



LarKC

The Large Knowledge Collider

a platform for large scale integrated reasoning and Web-search

FP7 – 215535

D2.1.2

Selection: Experimental Apparatus

H. Cunningham, A. Roberts, Y. Li, N. Aswani, V. Momchev, A. Kiryakov, J. Quesada, L. Schooler

Document Identifier:	LarKC/2008/D2.1.2/V1.0
Class Deliverable:	LarKC EU-IST-2008-215535
Version:	1.0
Date:	December 18, 2008
State:	final
Distribution:	internal

EXECUTIVE SUMMARY

Here we report the experimental setup and software apparatus for experimentation with retrieval and selection methods throughout the project. Deliverable D2.1.1 introduced a number of methods for selecting subspaces from large triple-based data sets. In this deliverable we discuss how we intend to operationalise that research as part of the LarKC software ecosystem, incorporating a diverse set of methods from various research streams into a single harness.

DOCUMENT INFORMATION

IST Project Number	FP7 – 215535	Acronym	LarKC
Full Title	Large Knowledge Collider		
Project URL	http://www.larkc.eu/		
Document URL	http://wiki.larkc.eu/LarkcProject/WP2		
EU Project Officer	Stefano Bertolo		

Deliverable	Number	2.1.2	Title	Selection: Experimental Apparatus
Work Package	Number	2	Title	Selection and Retrieval



Date of Delivery	Contractual	M6	Actual	31-Sept-08
Status	1.0		final <input type="checkbox"/>	
Nature	prototype <input type="checkbox"/> report <input checked="" type="checkbox"/> dissemination <input type="checkbox"/>			
Dissemination Level	public <input checked="" type="checkbox"/> consortium <input type="checkbox"/>			

Authors (Partner)	University of Sheffield			
Resp. Author	Hamish Cunningham		E-mail	hamish@dcs.shef.ac.uk
	Partner	University of Sheffield	Phone	+44 114 222 1891

Abstract (for dissemination)	Here we report the experimental setup and software apparatus for experimentation with retrieval and selection methods throughout the project. Deliverable D2.1.1 introduced a number of methods for selecting subspaces from large triple-based data sets. In this deliverable we discuss how we intend to operationalise that research as part of the LarKC software ecosystem, incorporating a diverse set of methods from various research streams into a single harness.
Keywords	Selection, retrieval, WP2, apparatus

Version Log				
Issue Date	Rev No.	Author		Change
3/June/2008	1	Hamish	Cun-	0.1
6/Sept/2008	2	Hamish	Cun-	1.0 draft
17/Sept/2008	3	Hamish	Cun-	1.0 incorporating internal reviewer comments and more input from all partners
18/Dec/2008	4	Hamish	Cun-	1.1 new introduction

PROJECT CONSORTIUM INFORMATION

Acronym	Partner	Contact
Semantic Technology Institute Innsbruck http://www.sti-innsbruck.at	STI 	Prof. Dr. Dieter Fensel Semantic Technology Institute (STI) Innsbruck, Austria E-mail: dieter.fensel@sti-innsbruck.at
AstraZeneca AB http://www.astrazeneca.com/	ASTRAZENECA 	Bosse Andersson AstraZeneca Lund, Sweden E-mail: bo.h.andersson@astrazeneca.com
CEFRIEL SCRL. http://www.cefriel.it/	CEFRIEL 	Emanuele Della Valle CEFRIEL SCRL. Milano, Italy E-mail: emanuele.dellavalle@cefriel.it
CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O. http://cyceurope.com/	CYCORP 	Michael Witbrock CYCORP, RAZISKOVANJE IN EKSPERIMENTALNI RAZVOJ D.O.O., Ljubljana, Slovenia E-mail: witbrock@cyc.com
HÄ́uchstleistungsrechenzentrum, Universitaet Stuttgart http://www.hlrs.de/	HLRS 	Dr. Georgina Gallizo HÄ́uchstleistungsrechenzentrum, Universitaet Stuttgart Stuttgart, Germany E-mail: gallizo@hlrs.de
Max-Planck-Institut für Bildungsforschung http://www.mpib-berlin.mpg.de/index.js.en.htm	MAXPLANCKGESELLSCHAFT 	Dr. Lael Schooler, Max-Planck-Institut für Bildungsforschung Berlin, Germany E-mail: schooler@mpib-berlin.mpg.de
Ototext Lab, Sirma Group Corp. http://www.ontotext.com/	ONTO 	Atanas Kiryakov, Ototext Lab, Sirma Group Corp. Sofia, Bulgaria E-mail: atanas.kiryakov@sirma.bg
SALTLUX INC. http://www.saltlux.com/EN/main.asp	Saltlux 	Kono Kim SALTLUX INC Seoul, Korea E-mail: kono@saltlux.com
SIEMENS AKTIENGESELLSCHAFT http://www.siemens.de/	Siemens 	Dr. Volker Tresp SIEMENS AKTIENGESELLSCHAFT Muenchen, Germany E-mail: volker.tresp@siemens.com
THE UNIVERSITY OF SHEFFIELD http://www.shef.ac.uk/	Sheffield 	Prof. Dr. Hamish Cunningham THE UNIVERSITY OF SHEFFIELD Sheffield, UK E-mail: h.cunningham@dcs.shef.ac.uk
VRIJE UNIVERSITEIT AMSTERDAM http://www.vu.nl/	Amsterdam 	Prof. Dr. Frank van Harmelen VRIJE UNIVERSITEIT AMSTERDAM Amsterdam, Netherlands E-mail: Frank.van.Harmelen@cs.vu.nl

<p>THE INTERNATIONAL WIC INSTITUTE, BEIJING UNIVERSITY OF TECHNOLOGY http://www.iwici.org/</p>	<p>WICI</p> 	<p>Prof. Dr. Ning Zhong THE INTERNATIONAL WIC INSTITUTE Maebashi, Japan E-mail: zhong@maebashi-it.ac.jp</p>
<p>INTERNATIONAL AGENCY FOR RESEARCH ON CANCER http://www.iarc.fr/</p>	<p>IARC2</p> 	<p>Dr. Paul Brennan INTERNATIONAL AGENCY FOR RESEARCH ON CANCER Lyon, France E-mail: brennan@iarc.fr</p>

CONTENTS

1	INTRODUCTION	1
2	AN OPERATIONAL MODEL OF RETRIEVAL METHODS FOR KB SELECTION	2
3	A NOTE ON RELEVANCE	4
3.1	Limitations of Current Search Technology	5
4	ANATOMY OF A SELECTION PLUGIN	7
4.1	Keywords, Vector Spaces and Selection	7
4.1.1	Related work	9
4.2	Example Code	10
4.2.1	The Example Class	14
5	MEASUREMENT	22
6	APPENDIX: DEVELOPMENT METHODS	31
6.1	Agile Software Development	31
6.1.1	Agile Teams and Scrum	32
6.1.2	Supporting the Programmer with XP	34
6.2	Agile Research	35
6.2.1	Process Summary	36
	BIBLIOGRAPHY	38

1 INTRODUCTION

Deliverable D2.1.1 introduced a number of methods for selecting subspaces from large triple-based data sets. In this deliverable we discuss how we intend to operationalise the work as part of the LarKC software ecosystem.

The problem is to incorporate a diverse set of methods from various research streams into a single harness, and one which is still evolving (the LarKC platform – see particularly WP5). Therefore the work at Month 6 is necessarily preliminary, but we hope to show three things:

1. what the baseline algorithm for a naïve first version of a selection plugin can look like (we do this in chapter 4)
2. the types of metrics we can use to measure performance of the different selection approaches (chapter 5)
3. how our agile software process can address the shifting and evolving requirement set during the rest of the project (appendix 1)

As preliminaries we briefly contextualise the work and its relevance to the project in chapters 2 and 3, to which we now turn.

2 AN OPERATIONAL MODEL OF RETRIEVAL METHODS FOR KB SELECTION

In 1995, when the number of "usefully searchable" Web pages was a few tens of millions, it was widely believed that "indexing the whole of the Web" was already impractical or would soon become so due to its exponential growth. A little more than a decade later, the GYM search engines—Google, Yahoo!, and Microsoft—are indexing almost a thousand times as much data and between them providing reliable subsecond responses to around a billion queries a day in a plethora of languages. (David Hawking, IEEE Computer, June 2006.)

Information retrieval (IR) technology has proliferated in rough proportion to the expansion of knowledge and information as a central factor in economic success. The main dimensions conditioning choice of search technology are:

- Volume. The GYM big three web search engines (Google, Yahoo!, Microsoft) deliver sub-second responses to hundreds of millions of queries daily over hundreds of terrabytes of data. At the other end of the scale desktop search systems can rely on substantial compute resources relative to a small data set.
- Value. The retrieval of high-value content (typically within corporate intranets or behind pay-for-use turnstiles) is often mission-critical for the business that owns the content. For example the BBC allocates a skilled staff member for eight hours per broadcast hour to index their most important content.

To process web-scale volumes GYM use a combination of one of the oldest and simplest retrieval data structures (an inverted file that relates search terms to documents) and a ranking algorithm whose most important component is derived from the link structure of the web. In general, when specifying a search, users enter a small number of terms as a query, based on words that people expect to occur in the types of document they seek. This gives rise to a fundamental problem, known as "index term synonymy": not all documents will use the same words to refer to the same concept. Therefore, not all the documents that discuss the concept will be retrieved by a simple keyword-based search. Furthermore, query terms may of course have multiple meanings; this problem is called "query term polysemy". The ambiguity of the query can lead to retrieval of irrelevant information and/or failure to retrieve relevant information.

High-value (or low-volume) content retrieval systems address these problems with a variety of semantics-based approaches that attempt to perform conceptual indexing and logical

querying. For example, the BBC system cited above indexes using a thesaurus of 100,000 terms that generalises over anticipated search terms. Life Sciences publication databases increasingly use rich terminological resources to support conceptual navigation (MeSH, the Gene Ontology, Snomed, the unified UMLS system, etc).

Our task in LarKC's selection and retrieval work package (WP2) is to show how the high-value techniques can scale to higher volumes than is currently feasible. This (Month 6) deliverable discusses an operational model for selection from semantic repositories using retrieval-based methods.

3 A NOTE ON RELEVANCE

European telecommunications companies (telcos) are among the strongest sectors of our economy and employ more than 1 million people across the union. Location-based services are an extremely important element of the future business models of the telcos, with more than 400 million EU citizens now owning mobile phones (and this figure is rising at around 15% per year). LarKC's Real Time City case study will demonstrate new possibilities for telco markets and contribute to Europe maintaining its strong position in the global economy. The involvement of Saltlux in the consortium will ensure the industrial relevance of the case study.

European pharmaceuticals and life sciences companies are positioned in the world's second largest drugs and health market (after the US) and have demonstrated strong profit growth in recent years. High technology investment in the sector remains high with more than 2000 biotechnology companies employing more than 100,000 people and generating over over €21.5 billion yearly revenue (europabio.be). In the UK pharma are the "biggest sector investor in R&D in the UK accounting for around 24% of total investment by business, valued at £3.2bn, about £9m a day in 2004" (DTI). LarKC's two life sciences case studies will be of direct relevance to European high-tech industry in this sector, and the involvement of the United Nations' World Health Organisation and AstraZeneca's research labs will show how the project's results can apply sector-wide.

Referring back to section 1 and the distinction between high-volume and high-value content and the search technologies that are typically applied in each case, it is clear that the US currently dominates the high-volume search market, where the big three are Google, Yahoo! and Microsoft. European initiatives such as Quaero are partly motivated by this dominance (and Siemens, a central LarKC partner, have been an active participant). With respect to high-value content the position is not so clear-cut, and European public research investment in new technology under the general heading of the Semantic Web is currently the highest in the world. Reasons why the high-value methods are more common in Europe include:

- the high quality of European cultural artefacts
- the multilingual character of the community
- the strong local scientific tradition in informatics

An important impact of LarKC will be to show how the methods currently becoming more common for high-value content can be applied to larger volumes, and this is a critical area for

the international competitiveness of European ICT (and for our society in general, because the owners of the gateways to the web are an increasingly central mediator between public and culture).

3.1 Limitations of Current Search Technology

(An earlier version of this material appeared in (Bon06b).)

In general, when specifying a search, users enter a small number of terms in the query. The query describes the information need, and is commonly based on the words that people expect to occur in the types of document they seek. This gives rise to a fundamental problem, known as "index term synonymy": not all documents will use the same words to refer to the same concept. Therefore, not all the documents that discuss the concept will be retrieved by a simple keyword-based search. Furthermore, query terms may of course have multiple meanings; this problem can be called "query term polysemy". As conventional search engines cannot interpret the sense of the user's search, the ambiguity of the query leads to the retrieval of irrelevant information.

Technically, the above two problems can be explained as follows: search engines that match query terms against a keyword-based index will fail to match relevant information when the keywords used in the query are different from those used in the index, despite having the same meaning. This problem can be overcome to some extent through thesaurus-based expansion of the query; this approach increases the level of document recall, but it may result in significant precision decay, i.e. the search engine returning too many results for the user to be able to process realistically.

Users can partly overcome query ambiguity by careful choice of additional query terms. However, there is evidence to suggest that many people may not be prepared to do this. For example, an analysis of the transaction logs of the Excite WWW search engine (Jansen00) showed that web search engine queries contain on average 2.2 terms. Comparable user behaviour can also be observed on corporate intranets: an analysis of the queries submitted to the intranet search engine of BT over a 4-month period between January and May 2004 showed an average query length of only 1.8 terms.

In addition to difficulties in handling synonymy and polysemy, conventional search engines are of course unaware of any other semantic links between terms (or, more precisely, the concepts which the terms represent). A major limitation of non-semantic IR approaches is that they cannot handle queries which either require knowledge and data which are not

available in the documents; or require extraction, explicit structuring, and reasoning about some data. Consider for example, the following query:

"telecom company" Europe "John Smith" director

The information need appears to be for documents concerning a telecom company in Europe, a person called John Smith, and a board appointment. Note, however, that a document containing the following sentence would not be returned using conventional search techniques:

"At its meeting on the 10th of May, the board of London-based O2 appointed John Smith as CTO"

In order to be able to return this document, the search engine would need to be aware of the following semantic relations:

- O2 is a mobile operator, which is a kind of telecom company;
- London is located in the UK, which is a part of Europe;
- A CTO is a kind of director.

These are precisely the kinds of relations which can be represented and reasoned over using semantic web technology.

4 ANATOMY OF A SELECTION PLUGIN

This section describes the implementation seeds that we have worked on in the first six months of LarKC. These seeds are intended to grow into the baseline plugin and the quantum informatics plugin. We approach the problem from two directions:

- top-down in section 4.1 by describing an approach derived on vector space retrieval models
- bottom-up in section 4.2 by describing prototype code (which is available on SourceForge and via LarKC's GForge repository)

This is a Month 6 deliverable, and it is worth noting that we are in the early stages of the work, and that the API that we have adopted has very definitely **not** been harmonised with either the work on the API from Innsbruck/HLRS or the work on the metareasoning platform from Cyc/JSI. Discussions with these partners (on the WP5 list) and within WP2 concluded that it is too early to try and unify our different models because none of them are stable or complete. Therefore we will proceed with the knowledge that the APIs will certainly change, but nonetheless in the hope that our prototyping experience will be valuable both for the first working plugin versions due at Month 12 and as inputs to future LarKC API definition work.

4.1 Keywords, Vector Spaces and Selection

As discussed in deliverable D2.1.1 (selection and retrieval parameters), we will develop a range of plugins for the LarKC platform. To give us a sound basis to evaluate our progress we will start by developing a **baseline** plugin using a very straightforward approach and then build on this work to investigate a range of spreading activation, geometrical spaces and streaming databases approaches.

The simplest method for subsetting a set of RDF triples relative to a SPARQL query is key word matching: first collect key words from the triples and create an inverted index from words to all triples associated with the word. Then, given a query, extracts key words from the query and retrieve from the inverted index those triples which associate with some of the query's key word.

The word matching method is quick and efficient in storage, but it is not flexible in the sense that it just produces a fixed subset for one query and cannot take into account the

constraints represented by context and quality of service criteria. One method for adding flexibility to the word matching approach is to rank those triples selected by matching.

For example, we can assign a weight (or an importance indicator) to each key word in a triple and store the weights in the inverted index of key words and triples. Then we can rank the triples selected by the word matching method for a given query, based on the weights of the key words of the query in those triples. Another method for ranking is to create vectors (e.g. based on TFIDF scores and the weights of key words) for each triple and query and then rank the triples based on the inner product or cosine between the triples' and query's vectors. For convenience, we may call the first extension of the word matching approach as weighted word matching, and the second extension as vector ranking.

Another concern with the word matching is where to find the key words to associate with a triple. We can assume that the local names of the three components of a triple are important key words. We can also extract the content words from some fields of a triple's URI as key words for a triple using the URI reference, though those content words may be less important than the local names of the URI reference for characterising the triple. If a triple has a link back to the source document from which it was generated, then those content words in the source document can also be regarded as the key words for the triple (especially those proximate to the point of reference).

Once we create a vector space model by associating a vector with a triple or query, we can explore vector space retrieval methods which are potentially richer and more sensitive than the key word methods. For example, we may cluster the triples based on the vector representation for more efficient subsetting (with a first step that selects clusters, i.e. that starts to move away from the requirement to consider all triples for each selection task). We may also map the vector space to a higher dimensional one or even infinite-dimensional Hilbert space e.g. via kernelisation, for more effective comparison between query and triples.

One particular implementation of the method is for the case that all triples are generated from documents and each triple has a link back to the source document. If we only want to use the content words from the source documents to characterise the triples, then we have a subsetting method based on the conventional document based IR methods. It first selects those documents which are relevant to a query and then retrieves all the triples generated from those documents as a subset for the query. Note that this method cannot distinguish the key words from different components of a triple, though this kind of distinction is available from both the triples and the SPARQL query.

Previous work on searching and summarising triple spaces (see below) has demonstrated promising results. We will adopt a related approach:

- for each triple and for each SPARQL query derive a vector (inputs: the lexemes associated with the URLs of the nodes; a gloss or description if it exists in the ontology; annotated documents perhaps)
- to increase recall we can expand the inputs to the vector model using inheritance (and maybe other) relations within the ontology
- standard vector space retrieval to get the triples that best match the query (i.e. dot product pairwise comparison)

The standard naive comparison approach is inefficient (which is a reason why web search engines often use inverted file type approaches instead). Therefore although early versions may use 1 vector per triple, with a later version we may end up doing some clustering and hierarchical matching (to make it more efficient by reducing the number of pairwise comparisons – no doubt there is also plenty of other prior work on optimising VS retrieval, which we should certainly investigate before undertaking this).

After exploring the baseline solutions and their variants in a standard vector space, a later version will then contrast this with a Hilbert space.

To tie things up with the use cases and to give us task-based evaluation possibilities, we will seek to use a KB generated from LLD (Linked Life Data - a very large ontology/KB derived from lots of other life sciences resources and represented in ORDI) and from annotation of Medline. This will allow us to compare directly with the baseline plugin; for evaluation of the annotation work we should also look into using other gold standards out there, e.g. Briscoe's Flybase work at Cambridge, GENIA, others.

4.1.1 Related work

[25] extracted a virtual document for each URI reference in an RDF triple store (or equivalently each node in a RDF graph). This virtual document contains the local name and labels of the URI reference, other associated literals such as those in `rdfs:comments`, and the names of neighbour nodes in the RDF graph.

[5] used these virtual documents to create a system for searching and browsing RDF triples using a vector space retrieval model. Basically the system uses a keyword matching method to compare a query against the virtual documents associated with a triple in order to determine the relevancy of the triple to the query.

[14] presented an A-Box summarization technique for efficiently making inference over the A-box. The technique used one summarized node to represent all individuals belonging to one concept, unless two individuals were explicitly stated to be different. In the latter case two more summarized nodes were used to represent the two individuals. The relation between two individuals were assigned to the two corresponding summarized nodes.

4.2 Example Code

This section documents our initial API models for selection plugins. It should be read in conjunction with the auto-generated software documentation at `larkc.sf.net`.

The code is structured around three packages:

eu.larkc.selection This package defines the core interfaces which plugins will implement, defining the top-level API and the various data structures that are passed between the calling environment (the future LarkC platform) and the plugins themselves. Key classes are `SelectionPlugins`, which perform `SelectionTasks` relative to `ReasoningTasks` over `DataSets` in particular `SelectionContexts`.

gate.larkc.selection This package contains first implementations of the Baseline and Quantum selection plugins, and an `Example` class that creates a plugin, contextual data and performs an example task. Also implements the default `SelectionContext`.

com.ontotext.larkc.selection This package contains first implementations of the `ReasoningTask` and `DataSet` classes, and in future will contain the `SpreadingActivationSelectionPlugin`.

The current set of classes and interfaces comprises:

- Top-level interfaces (`eu.larkc.selection`):

eu.larkc.selection.SelectionPlugin Essentially the mission of a selection plugin is to reduce the scale of reasoning tasks by performing a subset operation on a semantic repository. These operations are resource-bounded (they will time out after a specific duration, at which point partial results will be returned). They are relative to context – for details see `SelectionTask`.

eu.larkc.selection.SelectionTask `SelectionTasks` encapsulate a `ReasoningTask`, a `DataSet`, a `TripleSet` and a `SelectionContext` in order to provide a convenient object to pass to plugins and also for storage in a (future) selection history.

- eu.larkc.selection.ReasoningTask** Reasoning tasks are essentially SPARQL queries (LarKC is a ‘SPARQL endpoint on steroids’ as Frank has it). This class has a `lexicalise` methods which returns a set of lexemes derived heuristically from the reasoning task (more on this in relation to the baseline and geometrical semantics plugins).
- eu.larkc.selection.DataSet** DataSets are as described in D2.1.1 (parameters): graphs that integrate multiple RDF graphs in a such way that the graphs can still be distinguished and managed separately.
- eu.larkc.selection.TripleSet** TripleSets are as described in D2.1.1 (parameters): part of a DataSet, i.e. a multi-graph represented as a subset of its quadruples. TripleSets are also named, i.e. each TripleSet is associated with a unique name. This component of a selection task may be null, in which case the whole DataSet is used.
- eu.larkc.selection.SelectionContext** Contexts include elements like the activation history of the triplestore; annotated corpora (medline, for example; the users desktop, perhaps).
- eu.larkc.selection.ResourceConstraints** This class encapsulates constraints on the runtime resource usage of a selection plugin. For example plugins may be required to execute a selection task within a fixed period of time, or using a specific set of cluster nodes, etc.
- eu.larkc.selection.SelectionException** Models exceptional conditions and errors.
- Implementations of Sheffield plugins and GATE-related data structures (`gate.larkc.selection`):
 - gate.larkc.selection.Example** This class contains example code, which is described in detail below.
 - gate.larkc.selection.BaselineSelectionPlugin** The (naive, simple) baseline against which other plugins may be measured.
 - gate.larkc.selection.QuantumSelectionPlugin** A plugin exploring complex geometrical spaces for selection.
 - gate.larkc.selection.SelectionContextImpl** A default implementation of selection contexts that uses GATE’s Document abstraction (which in turn implements standoff annotation facilities and is at the core of a large number of semantic annotation systems).
 - gate.larkc.selection.SelectionTaskImpl** A default task implementation. Parameters: `reasoningTask` – a ReasoningTask is in effect a query (from which the baseline plugin will derive lexemes with which to do a standard IR query over a corpus);

dataset – initially DataSet will wrap an ORDI class; tripleSet – the data from the repository from which we will select. If this target set parameter is null, then the whole dataset DataSet should be used; selectionContext – a context encapsulates additional information from which to derive query terms – it might be a set of documents currently active on a desktop, for example, or the activation record of the input tripleset.

- Implementations of OntoText plugins and OWLIM/ORDI/SAR-related data structures (com.ontotext.larkc.selection):

com.ontotext.larkc.selection.DataSetImpl This class provides a default implementation of Data Sets. It is a wrapper around SarSearcher. It allows manipulating data in the repository.

com.ontotext.larkc.selection.TripleSetImpl This class provides a default implementation of TripleSets as a wrapper around ORDI and Sesame datasets (as an extension of DataSetImpl).

com.ontotext.larkc.selection.ReasoningTaskImpl A default implementation taking care of deriving a list of lexemes (to form IR keyword queries in the Baseline-Plugin, for example) from a SPARQL query.

com.ontotext.larkc.selection.SpreadingActivationSelectionPlugin This plugin will explore the use of spreading activation in weighted RDF graphs for selection purposes. For details of the approach see D2.1.1 (parameters).

Diagrammatically:

To show how these classes combine in action, the next section presents code from the `gate.larkc.selection.Example` class.

4.2.1 The Example Class

The `gate.larkc.selection.Example` class looks like this:

```
1 package gate.larkc.selection;
2
3 import java.io.File;
4 import java.util.HashMap;
5 import java.util.Set;
6
7 import org.openrdf.model.URI;
8
9 import com.ontotext.larkc.selection.DataSetImpl;
10 import com.ontotext.larkc.selection.ReasoningTaskImpl;
11 import com.ontotext.larkc.selection.TripleSetImpl;
12
13 import eu.larkc.selection.DataSet;
14 import eu.larkc.selection.ReasoningTask;
15 import eu.larkc.selection.SelectionTask;
16 import eu.larkc.selection.TripleSet;
17 import gate.ir.annic.sar.FillAStore;
18 import gate.util.Benchmark;
19 import gate.util.GateRuntimeException;
20
21 /**
22  * Example code for selection plugins.
23  */
24 public class Example {
25     /**
26      * This example class:
27      * <ul>
28      * <li> constructs or loads a GATE corpus (indexed)
29      * <li> ditto an ORDI KB
30      * <li> constructs ReasoningTask, SelectionContext, SelectionTask and so on
31      * <li> illustrates a SeclectionPlugin.select call with the
32      * <li> BaselineSelectionPlugin
33      * <li> ...
34      * </ul>
35      */
36     static public void main(String[] args) {
37
38         // benchmark id – to be used for logging messages
```

```
39     String benchmarkId = Example.class.getName();
40
41     // folder to store index definition
42     File indexFile = new File("./indexDir");
43     if(!indexFile.exists() || !indexFile.isDirectory()) indexFile.mkdirs();
44     indexFile.deleteOnExit();
45
46     // datastore path
47     String dsPath =
48         new File("resources/docs/annotated/medline-xml-datastore")
49             .getAbsolutePath();
50     String indexPath = indexFile.getAbsolutePath();
51
52     // enabling benchmarking
53     System.setProperty("gate.enable.benchmark", "true");
54     HashMap<String, String> benchmarkFeatures = new HashMap<String, String>();
55
56     // indexing documents with sam/sar
57     // ids of documents are used as named graphs
58     long start = Benchmark.startPoint();
59     String[] applicationArgs = new String[]{indexPath, dsPath, "output"};
60     try {
61         FillAStore.main(applicationArgs);
62     }
63     catch(Exception e) {
64         throw new GateRuntimeException(e);
65     }
66     Benchmark.checkPoint(start, benchmarkId + "_indexing_docs", Example.class
67         .getName(), benchmarkFeatures);
68
69     // example of how to use BaselineSelectionPlugin
70     BaselineSelectionPlugin plugin = new BaselineSelectionPlugin();
71     plugin.setBenchmarkId(benchmarkId);
72
73     DataSet ds = new DataSetImpl();
74     SelectionContextImpl sc = new SelectionContextImpl();
75
76     // find out documents with annotations of type Gene or Protein
77     String exampleQuery = "Disease, Protein";
78     ReasoningTask rt = new ReasoningTaskImpl(exampleQuery);
79     ((ReasoningTaskImpl)rt).setBenchmarkId(benchmarkId);
80
81     // documentIds to use in context
82     String[] contextDocIds =
83         new String[]{"med-2.txt.xml_0001B__1220439525188__8894",
84             "med-1.txt.xml_0001A__1220439524435__5277",
```

```

85     "med-3.txt.xml_0001C__1220439525557__6095"};
86   for(int i = 0; i < contextDocIds.length; i++) {
87     sc.addDocument(contextDocIds[i]);
88   }
89
90   // create an appropriate object of selection task
91   start = Benchmark.startPoint();
92   SelectionTask task = SelectionTask.create(rt, ds, null, sc);
93   Benchmark.checkPoint(start, benchmarkId + "_creating_selection_task",
94     Example.class.getName(), benchmarkFeatures);
95
96   // actual stuff – provided a task without any constraints
97   // and asking the plugin to select the relevant tripliset
98   start = Benchmark.startPoint();
99   TripleSet set = plugin.select(task, null);
100  Benchmark.checkPoint(start, benchmarkId + "_selecting_tripliset",
101    Example.class.getName(), benchmarkFeatures);
102
103  // returns two out of three docs as match
104  Set<URI> namedGraphs = ((TripleSetImpl)set).getNamedGraphs();
105  System.out.println("Found Documents for:\" + exampleQuery + "\");
106  for(URI aNamedGraph : namedGraphs) {
107    System.out.println("\t" + aNamedGraph.toString());
108  }
109
110  // refining search, though using the same documents in the context.
111  // This time though seaching for documents with annotations which have one
112  // of the two feature values listed below
113  exampleQuery = "4267,7645"; // gene database identifiers
114  start = Benchmark.startPoint();
115
116  rt = new ReasoningTaskImpl(exampleQuery);
117  ((ReasoningTaskImpl)rt).setBenchmarkId(benchmarkId);
118
119  // lexicalising query
120  benchmarkFeatures.put("reasoningTaskQuery", exampleQuery);
121  Benchmark.checkPoint(start, benchmarkId + "_lexicalising_query",
122    Example.class.getName(), benchmarkFeatures);
123  benchmarkFeatures.remove("reasoningTaskQuery");
124
125  // creating a task
126  start = Benchmark.startPoint();
127  task = SelectionTask.create(rt, ds, set, sc);
128  Benchmark.checkPoint(start, benchmarkId + "_creating_selection_task",
129    Example.class.getName(), benchmarkFeatures);
130

```

```
131 // actual stuff – selecting an appropriate set of tripless those are relate
132 // to the task
133 start = Benchmark.startPoint();
134 set = plugin.select(task, null);
135 Benchmark.checkPoint(start, benchmarkId + "_selecting_tripleset",
136     Example.class.getName(), benchmarkFeatures);
137
138 // returns one of the two named graphs found in the previous call
139 namedGraphs = ((TripleSetImpl)set).getNamedGraphs();
140 System.out.println("Found Documents for:\" + exampleQuery + "\"");
141 for (URI aNamedGraph : namedGraphs) {
142     System.out.println("\t" + aNamedGraph.toString());
143 }
144 } // main(String[])
145 } // Example
```

The runtime output of the example will look something like this:

```
[untar] Expanding: /home/hamish/ext/larkc-sf/selection/resources/docs/
annotated/datastore.tar into /home/hamish/ext/larkc-sf/
selection/resources/docs/annotated
[java] usage: java FillAStore <indexDirAbsPath> <datastoreAbsPath> <SetToIndex>
[java] Using /home/hamish/gate as GATE home
[java] Using /home/hamish/gate/plugins as installed plug-ins directory.
[java] Using /home/hamish/gate/gate.xml as site configuration file.
[java] Using /home/hamish/.gate.xml as user configuration file
[java] Using /home/hamish/.gate.session as user session file
[java] CREOLE plugin loaded: file:/home/hamish/ext/gate/plugins/ANNIE/
[java] CREOLE plugin loaded: file:/home/hamish/ext/gate/plugins/Ontology_Tools/
[java] 3 documents found
[java] med-2.txt.xml_0001B___1220439525188___8894 (1 of 3)...Indexing :
med-2.txt.xml_0001B___1220439525188___8894 ...
[java] ruleSet=owl-max, partialRdfs=true
[java] num pages=4000, page size=2729
[java] 454...
[java] done
[java] med-1.txt.xml_0001A___1220439524435___5277 (2 of 3)...Indexing :
med-1.txt.xml_0001A___1220439524435___5277 ...
[java] 410...
[java] done
[java] med-3.txt.xml_0001C___1220439525557___6095 (3 of 3)...Indexing :
med-3.txt.xml_0001C___1220439525557___6095 ...
[java] 552...
```

```
[java] done
[java] NumberOfStatements = 8338
[java] NumberOfExplicitStatements = 8162
[java] NumberOfExplicitlyInsertedStatements = 4102
[java] NumberOfEntities = 1764
[java] Restoring entity hash table. . .
[java] Done in 583 ms.
[java] ruleSet=owl-max, partialRdfs=true
[java] num pages=4000, page size=2729
[java] Found Documents for:"Disease, Protein"
[java]      http://ordi.ontotext.com/sar#
           med-1.txt.xml_0001A___1220439524435___5277
[java]      http://ordi.ontotext.com/sar#
           med-3.txt.xml_0001C___1220439525557___6095
[java] Found Documents for:"4267,7645"
[java]      http://ordi.ontotext.com/sar#
           med-3.txt.xml_0001C___1220439525557___6095
```

The BaselineSelectionPlugin looks like this:

```
1 package gate.larkc.selection;
2
3 import java.util.HashMap;
4 import java.util.HashSet;
5 import java.util.List;
6 import java.util.Map;
7 import java.util.Set;
8
9 import org.openrdf.model.URI;
10
11 import com.ontotext.larkc.selection.TripleSetImpl;
12 import com.ontotext.ordi.sar.server.handlers.NamingUtility;
13
14 import eu.larkc.selection.*;
15 import gate.util.Benchmark;
16 import gate.util.Benchmarkable;
17
18 /**
19  * The baseline selection plugin.
20  */
21 public class BaselineSelectionPlugin implements SelectionPlugin, Benchmarkable
22     /**
23      * benchmark ID
24      */
```

```
25     protected String benchmarkId;
26
27     /**
28      * benchmark Features
29      */
30     protected Map<Object, Object> benchmarkFeatures;
31
32     /**
33      * Constructor
34      */
35     public BaselineSelectionPlugin() {
36         benchmarkId = this.getClass().getName();
37         benchmarkFeatures = new HashMap<Object, Object>();
38     }
39
40     /**
41      * The baseline select operation goes like this: we get an indexed gate Corpus
42      * from the SelectionTask; the documents in this Corpus are referred to by the
43      * DataSet; we use the ReasoningTask to create an IR query on the Corpus; we
44      * then pick out all the triples in the DataSet that refer to the Documents
45      * returned by the IR query
46      */
47     public TripleSet select(SelectionTask selectionTask,
48         ResourceConstraints resourceConstraints) {
49
50         DataSet ds = selectionTask.getDataset();
51         TripleSet set = selectionTask.getTripleSet();
52         List<String> toSearch = selectionTask.getReasoningTask().lexicallise();
53         SelectionContext sc = selectionTask.getSelectionContext();
54
55         // 1. generate sparql query from the given list of words
56         // this query will be issued using SAR on the documents
57         // specified in the selection context
58         StringBuilder sparql_selectPart = new StringBuilder();
59         StringBuilder sparql_wherePart = new StringBuilder();
60         StringBuilder fromPart = new StringBuilder();
61
62         sparql_selectPart.append("SELECT ?graph_uri \n");
63
64         // specifying the documents in which the query should be
65         // searched for
66         SelectionContextImpl scImpl = (SelectionContextImpl)sc;
67         for(String aDocId : scImpl.getDocumentIds()) {
68             String namedGraph = NamingUtility.documentUri(aDocId);
69             fromPart.append("FROM NAMED <" + namedGraph + ">\n");
70         }

```

```

71
72 // variable definitions in the sparql wherePart
73 sparql_wherePart.append("WHERE {\n");
74 sparql_wherePart.append("GRAPH ?graph_uri {\n");
75 sparql_wherePart.append("?subject ?predicate ?value .\n");
76
77 // filtering out results those do not match the lexems obtained
78 // from the reasoning task
79 sparql_wherePart.append("FILTER (");
80 for(int i = 0; i < toSearch.size(); i++) {
81     String string = toSearch.get(i);
82     if(i != 0) {
83         sparql_wherePart.append(" || ");
84     }
85     sparql_wherePart.append("regex(?value,\"" + string + "\")");
86 }
87 sparql_wherePart.append(")\n}\n}");
88
89 long start = Benchmark.startPoint();
90 // 2. issue query in the dataset with constraints of
91 // named graphs mentioned in the sparql query
92 TripleSetImpl result =
93     (TripleSetImpl)ds.getTripleSet(sparql_selectPart.toString()
94         + fromPart.toString() + sparql_wherePart.toString());
95 Benchmark.checkPoint(start, getBenchmarkId() + "_obtaining_tripleset",
96     this, benchmarkFeatures);
97
98 // 3. obtain the named graphs, which are relevant to the sparql query
99 Set<URI> namedGraphs = new HashSet<URI>(result.getNamedGraphs());
100
101 // 4. intersection of the sets of named graphs.
102 if(set != null) {
103     // obtain the named graphs which are in result set
104     Set<URI> tripleSetNamedGraphs = ((TripleSetImpl)set).getNamedGraphs();
105
106     // and remove those from the namedGraphs which do not exist in the
107     // tripleSetNamedGraphs
108     namedGraphs.retainAll(tripleSetNamedGraphs);
109 }
110
111 // a new triples set that has named graphs for only
112 // those documents which matched the above specified query
113 TripleSetImpl setToReturn = new TripleSetImpl();
114 setToReturn.addNamedGraphURIs(namedGraphs);
115 return setToReturn;
116

```

```
117     } // select(SelectionTask, ResourceConstraints)
118
119     /**
120      * Predict the length of time that a selection task may take.
121      */
122     public Long predictCost(SelectionTask selectionTask) {
123         return new Long(0);
124     } // predictCost(SelectionTask)
125
126     public String getBenchmarkId() {
127         return benchmarkId;
128     }
129
130     public void setBenchmarkId(String arg0) {
131         benchmarkId = Benchmark.createBenchmarkId(this.getClass().getName(), arg0);
132     }
133 } // BaselineSelectionPlugin
```

Note that we've used GATE's benchmarking library to automate the collection of runtime statistics (which becomes a non-trivial problem in service-based architectures). In the longer-term this will need to harmonise with WP5's Quality of Service API.

5 MEASUREMENT

4. Measurement

This section describes our empirical methods and evaluation protocols. Five preliminary comments are necessary.

First, the LarKC platform will exist in a relatively large number of different instantiations depending on domain and task. Therefore evaluation will need to cover a range of different scenarios, the specific implementations of which will be elucidated later in the project. In other words the measurement picture will be complex, and is likely to become more so as the project progresses.

Secondly, the subsetting task may well be too abstract to be amenable to measurement. How do we know a subset is successful? Success can only be measured with respect to a task that has obvious relevance for humans. For example, a question answering system can be evaluated by checking how many answers it gets right; although a lot trickier than it seems [1], humans have intuitions about whether an answer is correct or not. Subsetting is not like question answering in that sense: humans have no intuitions whether a subset is 'good' or not, mainly because due to scale and lack of readability (see next points), they would not be able to transport the original KB and the subset into their minds to compare them. In fact, subsetting is just a step on which many other tasks depend (question answering being one example). In this sense, subsetting is explanatorily prior, but harder to evaluate in isolation. We can evaluate other more concrete tasks that depend on subsetting, but not subsetting itself.

Thirdly, measurement is extremely expensive, and the costs have to be balanced against development work in this WP (and elsewhere). Therefore in each of the various methods we use for evaluation it is almost inevitable that more could (and, in an ideal world, should) be done. To compensate for this partiality, the best method is to be part of established 3rd-party evaluation programmes such as those in TREC [2], MUC, DUC, ACE etc. At present this is not possible: none of these programmes are evaluating this type of task to the best of our knowledge. We will, however, track closely the availability of appropriate activities arising such programmes throughout the project lifetime, and seek to become involved wherever possible.

Fourthly, there are important differences with classical information retrieval. In information retrieval we distinguish between structured information and unstructured information. Semantic web data are by definition structured, whereas classical IR uses the bag of words approach where the units of analysis are the type/token and the doc. In fact, in classical IR the document plays a central role. But in an RDF-centric approach, the document plays a very minor role, if any; RDF triples are independent from the document they came from (although sometimes they can keep provenance information if it will be needed later).

Last, working on RDF knowledge bases (KB) means that people won't be able to read materials, because RDF is simply not designed for humans. That makes several evaluation methods unusable, because they require humans to read materials. There is an alternative notation for

RDF called N3 [3] that improves readability. However, N3 does not make RDF readable to the untrained eye of the general population.

Thus, reusing the literature on classical (unstructured) IR evaluation becomes impractical. New alternative method need to be created, and in this section we address our first steps to meet this challenge.

In the rest of this section we summarise five types of parameter upon which we will base our evaluation methods in WP2. These methods relate to:

- computational resources (especially overall speed)
- accuracy (a useful measure but hard to implement on LarKC-sized data sets)
- task-based, making use of the use case WPs and our end-user partners
- pipeline-based, making use of the abstraction and reasoning evaluation protocols
- qualitative and impressionistic measures relating to the overall success of the use cases, platform take-up and so on (and relatively weak from the point-of-view of directing algorithmic development in WP2, but strongly relevant to the project as a whole)

4.1. Speed

This is the most straightforward measurement. We can compute statements / second, i.e. statements considered for inclusion in our output subset, per second. The same way, another relevant measure statements removed per second. These need to be relative to computational resource usage (number and nature of cluster nodes, for example). The entire LarKC project would be subject to Quality of service (QoS) terms. That is, each plugin will need to make sure it can to guarantee a certain level of performance when in a data flow. Plugins will have to identify themselves and present their QoS profile. That is, speed measurement will be relevant for every plugin in the project, not only the selection plugins.

4.2. Accuracy

Traditional measures of accuracy are precision and recall. However, these require knowing the size of the (limited) dataset in advance, and the exact set that solves the query (i.e., which are relevant documents). This is not the case for larKC-sized KBs: it would be very hard to know which are the relevant documents (or triples) for all but the simplest, smallest KBs. Size is not the only obstacle: the KB is dynamic, and so are the subsets that one can sample from it. A dynamic dataset means that we can only approximate its size in a particular point in time. Also, it makes it even harder to have an estimation of the number of relevant triples.

For the subsetting problem recall (eq. 1) is more important than precision because subsetting is the first step for other processes. If some relevant information (that may contain the answer, or a key triple for reaching the right conclusion) is missed by the subsetting algorithm, then the rest of the system will not succeed. For example, imagine that someone builds a LarKC-based

application to address question answering. Obtaining a list of triples is only the first step in the question answering process.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|} \quad (1)$$

In most use cases for LarKC retrieval plugins, the number of relevant documents will not be known. Additionally, the concept of document –as central as it is in classical, unstructured information retrieval- is not meaningful for structured data such as RDF. As of today, the current interfaces for most plugins do not use any document-based arguments (Note: the interface may well change in the future).

Since triples are the logical unit of analysis for structured knowledge such as the one used by LarKC plugins, let us assume that recall can be computed on triples instead of documents. Determining an exhaustive set of relevant triplets in advance is not trivial for all but the simplest KB. So at LarKC-scale sizes the classical precision/recall metric is problematic, so we can perform this type of measurement on small test sets only, with manual checking precision and recall of selected triples. Additional issues come from the fact that it's not straightforward for humans to classify RDF triples as relevant/non-relevant. This basic operation is key for the entire measurement operation. Common sense may flag a triple as irrelevant for a human, but that same triple may be crucial for the successful operation of a reasoner plugin.

The subsetting problem may be isomorphic with machine summarization on structured data; the main difference is that subsetting is summarization at much larger scale. The key One could add some concepts from social networks (centrality, connectivity, etc) to pick up RDF that really captures the gist of the KB. The problem is how to validate it. You cannot ask a human to read the entire KB, much less in RDF form, and say if the summary is good. Other similar summary quality evaluation techniques depend on humans being able to read the resulting summary [4].

4.3. Task-based

The contribution of selection can be measured by running various end-user, platform-based analyses with and without selection; if we get exactly the same results we have 100% accuracy of the selection process.

In this section we will focus on two example problems described in the use case from WP7: The one will be the Monographs find-the-expert task. The task is described in the use case (requirements) for WP7b; we present a summary here.

Cancer Expert-finding

The International Agency for Research on Cancer (IARC) is a section of the world Health organization (WHO) that specializes on cancer research. The IARC monographs are a series of

volumes evaluating potential carcinogens. They are definitive, and encyclopedic. To create these volumes, IARC scientists need to recruit the experts to write each monograph. Since it is not trivial to find who are the most authoritative figures, one of the objectives for WP7 is to do a network analysis to help identify experts. We will use this task as a prototype for measurement in selection plugins.

Materials (Description of the monographs)

Each monograph contains a list of experts that were selected by a group of specialists, in part, based on searching PubMed for researchers relevant to the monograph's topic. Since this group's decision takes time and effort, one of the goals of WP7b is to automatically find experts. Testing the expert-finding algorithm is the goal of this example for testing.

Each monograph contains a working group of experts, plus sometimes some invited specialists. The combined number of experts (working group + specialists) per monograph is an average of 24. See table 1 for a sample of five of those monographs.

Monograph title	Url	working group	Invited specialists	observers
Combined estrogen-progestogen contraceptives and combined estrogen-progestogen menopausal therapy	http://monographs.iarc.fr/ENG/Monographs/vol91/index.php	16	4	
Non-Ionizing Radiation, Part 1: Static and Extremely Low-Frequency (ELF) Electric and Magnetic Fields	http://monographs.iarc.fr/ENG/Monographs/vol80/index.php	21		4
Tobacco Smoke and Involuntary Smoking	http://monographs.iarc.fr/ENG/Monographs/vol83/index.php	24		3
Inorganic and Organic Lead Compounds	http://monographs.iarc.fr/ENG/Monographs/vol87/index.php	23	1	2
Some Traditional Herbal Medicines, Some Mycotoxins, Naphthalene and Styrene	http://monographs.iarc.fr/ENG/Monographs/vol82/index.php	29		6

Thus, our testing procedure involves generating a list of about 24 authors from a knowledge base that consists of all PubMed at the time the monograph was created. This is the gold standard.

Method

A subsetting plugin should return the authors in the list when using the title of the monograph as a query. This is the basic retrieval test. An obvious question is, If PubMed (or Web of Science) was wrapped as a selection plugin, would it return the authors in the list when using the title of the monograph as a query? Of course the selection would be valid only in the time slot the original group was created.

We plan to select a random sample of monographs and extract the expert list. Then, for every expert selected by the IARC, we will query PubMed (or Web of Science) with the topic for which the expert was selected. PubMed (or Web of Science) can produce a ranked list of authors for every topic query terms. Since the the expert group is already known, doing a medline analysis relative to the timepoint at which the group was selected, and measuring the difference between the suggested and actual compositions is a good proxy for the accuracy of a selection plugin. Note that the IARC does more than just search; there is a certain amount of paper reading, delivering, and discussion between researchers. If all this could effort is saved by a smart selection plugin, it would be a good solution.

This testing method works under the following assumptions: (1) We have access to the repositories in question (PubMed or Web of Science), and (2) We can select a time slice that corresponds to the time the particular monograph was written.

On the basic test bench we are testing PubMed or Web of Science's own selection mechanisms, but as soon as selection plugins are developed, they can be tested in the same conditions. Ideally, they should produce better precision and recall than the current state of the art.

Since there is a know limited set of results, we can tell in advance which search results are relevant, so we can apply precision and recall measures.

Dependent variables:

1 - number of relevant results returned (and corresponding transformations, precision and recall).

2- cycles needed given a certain threshold for precision/recall

3- Triplets that are processed

4 – Use of Metareasoning. According to most research, metareasoning is computationally expensive and may be an important bottleneck [5]; plus, there's a trade-off between the costs of meta-computation and the savings it produces. Thus, this is an optimization situation for meta-computation. Too little and too much meta-reasoning will produce suboptimal performance [5]. thus, a combination of plugins that requires an optimal amount of metareasoning and should be preferred. The additional advantage is that the less metareasoning, the easier to debug a system is.

Independent variables

1 – Type of selection plugin

2 – type of reasoning plugin (baseline: null reasoner).

All combinations of selection and reasoning plugins will be tested. If we ever have to deal with a combinatorial explosion (thanks to a large number of plugins), there are techniques to sample the most informative comparisons that can be used.

Measurement of precision and recall for the combinations of selection and reasoning plugins would lead to standard factorial designs; analysis of variance (ANOVA) is an appropriate method.

Multiple regression models, where factors are combinations of plugins are another possibility. Precision and recall's variance might be explained by each plugin pair and their interactions. We can maximize correlation to a gold standard: the past experts selected in the existing monographs.

Expected results

Without having even preliminary plugins implemented, it is hard to venture hypotheses. However, one general prediction would be that type of reasoner and selection plugins will interact; some combinations will be more efficient than others. Such a result may be improved by iteration with new designs as variations of the most successful combinations.

ROC curves also have proved useful for the evaluation of machine learning techniques and could be employed here.

Evaluation of Single Nucleotide Polymorphisms (SNPs)

A second evaluation of a selection plugin that is also specific to the WP7b storyboard concerns detection of genes that could potentially be carcinogenic, given a small sample of carriers. Current statistical methods can be used to detect genes in extremely small proportions. Of course, small n and a large array of possible matches makes effect sizes small, and significance levels vary in consequence.

In a Gene expression profiling experiment, the expression levels of thousands of genes are simultaneously monitored to study the effects of certain diseases. For example, microarray-based gene expression profiling can be used to identify genes whose expression is changed in response to pathogens or other organisms by comparing gene expression in infected to that in uninfected cells or tissues.

Imagine a Genome Wide Association study (GWA) population will have a number of relevant SNPs raised below the significance threshold.

We would expect any improved method based on a successful selection plugin to raise these same SNPs below the threshold with a smaller population. A study that finds 2 SNPs over 1000 patients might be expected to find the same 2 SNPs in 500 patients. That is, selection plugins will increase the statistical power of the test. When applied to the full population of 1000, it should find the original 2 SNPs and additional ones that had not been previously seen. We can perform the experiment with and without selection in various combinations to evaluate the effect of the selection plugin.

Both storyboards (WP7a, WP7b) have a set of known good solutions. These can be used to construct the test. If we can increase the statistical power of the test, then the selection Plugin is working: dropping information still gets us the same results. This could have important economic consequences, as certain GWA population data are expensive to collect.

4.4. Pipeline-based

Each of the stages in the LarKC pipeline will measure success in various ways. As these mechanisms emerge, we may hope to gain additional measures for selection by performing them with and without the selection processes operational. For example,

- pipeline speed/resource consumption with/without selection
- result of pipeline with/without selection

Here we will piggyback on the abstraction and reasoning evaluations (which have to quantify the success of their parts of the pipeline), and evaluate one cycle behind their release cycle.

4.5. Qualitative

Qualitative and impressionistic measures relating to the overall success of the use cases, platform take-up and so on (and relatively weak from the point-of-view of directing algorithmic development in WP2, but strongly relevant to the project as a whole).

4.6. Other factors

In each of these cases there may also be an additional dimension relating to different combinations of methods. For example, we imagine testing various associative mechanisms with various temporal mechanisms. It could well be that the best combination is OntoText's spreading activation or Sheffield's Quantum Semantics with ACT-R's decay mechanisms. Where possible we will evaluate the various cross-pollination combinations.

Finally, in our software development process we make use of unit tests and benchmarking code (For a more detailed description of agile methods see also WP2's D2.1.2).

References

- [1] J. Lin and B. Katz, "Building a Reusable Test Collection for Question Answering," *Journal-american society for information science and technology*, vol. 57, p. 851, 2006.
- [2] E. Voorhees and D. Tice, "Building a question answering test collection," *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 200-207, 2000.
- [3] W3C, "definitive N3 specification," <http://www.w3.org/DesignIssues/Notation3.html>, 1998.

- [4] I. Mani, "Summarization Evaluation: An Overview," *Proceedings of the Second NTCIR Workshop on Research in Chinese & Japanese Text Retrieval and Text Summarization*, 2001.
- [5] F. Van Harmelen, "A Model of Costs and Benefits of Meta-Level Computation," *Proceedings of the Fourth Workshop on Meta-programming in Logic (META'94)*, vol. 883, pp. 248-261, 1994.

6 APPENDIX: DEVELOPMENT METHODS

We include here the section on development methods from D7b1.1b (Methodology) as it is also relevant for WP2.

6.1 Agile Software Development

To start with the obvious, building software is hard. How do people cope with hard construction problems? They hire an architect, who draws a lot of complicated pictures (or, more likely these days, builds a complex computer model), and, for big projects, builds a small scale prototype to give the commissioning customer the feel of the thing. This design process iterates until all appears to be well, and off we go to the cement pourers and the concrete reinforcers.

Software engineering, both as craft and later as discipline, began by making a similar analogy, and so adopted a commission-design-build lifecycle conception of the ideal software development process. The first mature set of methods used techniques like flow charts, entity-relationship modelling and structure diagrams to capture the architectural team's vision of the system to be build (e.g. [32]). Later incarnations merged the data structures with the algorithms and created *object-oriented* design (e.g. [3]). These modelling approaches were then used to produce as comprehensive a design as possible, which was then turned over to implementation teams lower down the food chain (the skilled work was concentrated in architectural hands while the programmer's work taken to be almost as routine as brick-laying). Because of the step-by-step procession from high to low these methods were often called *waterfall* development.

And everyone lived happily ever after. Well, not really. The problem with the architectural metaphor for software engineering is that software and the contexts within which it is constructed have turned out to be rather unlike houses and bridges, and more like evolving viruses or cantankerous animals. This is probably due of a number of factors, including:

- The act of building software changes the needs and the ideas of the people who are going to use it.
- The working systems that software is meant to support or replace are often so complex that no single person possesses a comprehensive understanding of them.
- Software increasingly sits at the bleeding edge of technology's interface with society, and the markets that condition which parts of this interface expand and contract are

chaotic and fast-moving. In other words the context is both a strong determinant of success and changing very rapidly.

- Verification of the correct working of software is an extremely difficult problem to solve theoretically. Whereas we can be quite confident of the properties of a wall or joist built to certain well-established principles, the equivalent level of certainty with respect to software is much more difficult to achieve (and in the general case can only be approached via testing and not theoretical proofs¹).

Therefore it has become a truism that the best way to build a particular piece of software is to have built a very similar one before [15]. Obviously this is often impossible, and especially so in an R&D context. The waterfall methods have failed² most often in dynamic contexts; the piles of design artefacts can end up out-of-date before they are finished, and the synchronisation of design and code a never-ending process which makes the nature of the executable system ever more opaque.

Over the last decade or so a new family of software engineering methods have arisen based on the types of insight sketched above: the *agile methods*; below we discuss what we may usefully learn from this experience for our research work in projects like LarKC in general, and WP7b in particular, beginning with a short summary of two of the most popular members of the agile family, Scrum and XP (eXtreme Programming).

6.1.1 Agile Teams and Scrum

In their book *Agile Software Development with Scrum* Ken Schwaber and Mike Beedle [26] recall visiting a DuPont factory and talking with process engineers engaged in managing complex chemical processes. They described the types of activity they routinely encountered in software development and the types of (non-agile) design methods in use. The DuPont engineers laughed: they couldn't see how the methods matched up with the process because the methods were, from their point-of-view, appropriate only in much more defined

¹Automatic verification is a lively area of research but not yet widely applicable in practice.

²How to construct a failing software project: option 1: work for the UK government; option 2: define your plan as:

- first we (highly-paid developers) decide what to build (requirements)
- then we (middle-level developers) decide how to build it (design)
- then we (lower-level developers) implement it (coding)
- then we (support staff) deploy it (disaster)

A research equivalent: I hereby promise to invent The Next Big Thing at 3.27 PM on the 23rd of February 2010 in order to produce Deliverable D2,345 revision 63.

and predictable circumstances. They suggested that for the chaotic and fast-moving software world a different approach based on constant readjustment and flexibility was more appropriate. Scrum is:

...a process skeleton that includes a set of practices and predefined roles. The main roles in Scrum are the ScrumMaster who maintains the processes and works similar to a project manager, the Product Owner who represents the stakeholders, and the Team which includes the developers.

During each sprint, a 15-30 day period (length decided by the team), the team creates an increment of potential shippable (usable) software. The set of features that go into each sprint come from the product backlog, which is a prioritized set of high level requirements of work to be done. Which backlog items go into the sprint is determined during the sprint planning meeting. During this meeting the Product Owner informs the team of the items in the product backlog that he wants completed. The team then determines how much of this they can commit to complete during the next sprint. During the sprint, no one is able to change the sprint backlog, which means that the requirements are frozen for a sprint.

There are several implementations of systems for managing the Scrum process which range from yellow stickers and white-boards to software packages. One of Scrum's biggest advantages is that it is very easy to learn and requires little effort to start using.

<http://en.wikipedia.org/wiki/Scrum> Wikipedia 2/Sept/2008

Pictorially: see figure 6.1.

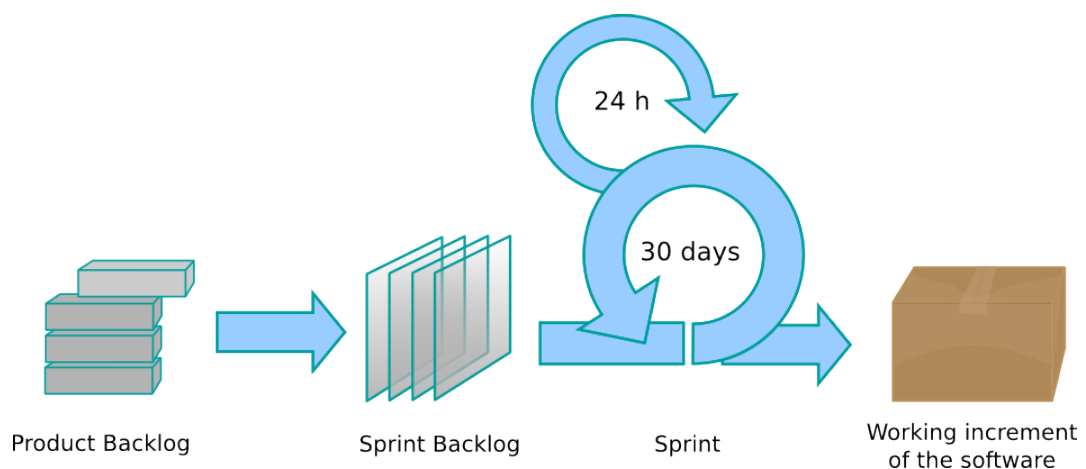


Figure 6.1: Scrum in pictures

6.1.2 Supporting the Programmer with XP

In 1999 Kent Beck wrote a book called *Extreme Programming Explained: Embrace Change* [1] wrote a book that advocated dispensing with most design artefacts and implementing a programmer-centric division of labour. The philosophy was to accept the dynamism and complexity of the software context and to make a virtue out of a weakness by focussing on flexibility and the ability to change course at any point. Key techniques were to keep code quality high (to *refactor* at every opportunity, building libraries of reusable code instead of continually reinventing incrementally different solutions), maximise communication between programmers and between programmers and clients, and to test, test, and test some more as the main method of verifying functionality.

The original list of practises was:

Small Releases: Put a simple system into production quickly, then release new versions on a very short cycle.

Metaphor: Guide all development with a simple shared story of how the whole system works.

Simple Design: The system should be designed as simply as possible at any given moment. Extra complexity is removed as soon as it is discovered.

Testing: Programmers continually write unit tests, which must run flawlessly for development to continue. Customers write tests indicating that features are finished.

Refactoring: Programmers restructure the system without changing its behaviour to remove duplication, improve communication, simplify, or add flexibility.

Pair Programming: All production code is written with two programmers at one workstation.

Collective Ownership: Anyone can change code anywhere in the system at any time.

Continuous Integration: Integrate and build the system many times a day, every time a task is completed.

40-hour Week: Work no more than 40 hours a week as a rule. Never allow overtime for the second week in a row.

On-site Customer: Include a real, live customer on the team, available full-time to answer questions.

Coding Standards: Programmers write all code in accordance with rules emphasizing communication throughout the code.

[1, page 54]

Regarding not working too much overtime, we have refined the protocol as follows:

- on Monday we recover from the weekend
- on Tuesday we get ready to work
- on Wednesday we work
- on Thursday we recover from work
- on Friday we get ready for the weekend

This keeps most of the team happy, although when we first raised the idea one member raised their hand and asked “Does this mean we’re going to work *every* Wednesday?”

Next we look at the promise of the new methods in a research context.

6.2 Agile Research

What of research? Here’s how *not* to do it:

Project Proposal: We love each other. We can work so well together. We can hold workshops on Greek islands together. We will solve all the problems of AI that our predecessors were too stupid to manage.

Analysis and Design: Stop work entirely, for a period of reflection and recuperation following the stress of attending the kick-off meeting in Luxembourg.

Implementation: Each developer partner tries to convince the others that program X that they just happen to have lying around on a dusty disk-drive meets the project objectives exactly and should form the centrepiece of the demonstrator.

Integration and Testing: The lead partner gets desperate and decides to hard-code the results for a small set of examples into the demonstrator, and have a fail-safe crash facility for unknown input (“well, you know, it’s still a prototype...”).

Evaluation: Everyone says how nice it is, how it solves all sorts of terribly hard problems, and how if we had another grant we could go on to transform information processing the World over (or at least the European business travel industry).

[9]

Seriously, there are significant risks hidden under the contractual carpet in modern ICT research, especially in shared-cost collaborative projects with many partners, many agendas, many pieces of background software that theoretically will be integrated and work smoothly together to demonstrate some radical new proposition regarding the possibility of the next high-tech revolution. For obvious reasons of financial probity these projects begin with the authoring of a lengthy annex to the funding contract that specifies several acres of milestones, deliverables and other checkpoints. Conformance to the plans set out in this annex (the *description of work* in current parlance) is measured by periodic expert peer review in the normal (academic) manner.

This is all well and good, but from a software development perspective it does tend to encourage staying with the type of waterfall methods which have been shown to be less than perfect in dynamic environments. How to cope?

In LarKC WP7b we have specified our outputs as a series of three iterations, and we have adopted an agile process for the development of these prototype instances. The rest of the section summarises this process.

6.2.1 Process Summary

Reprising the above, XP is a way to keep focussed on user needs and to ensure stability through testing. Scrum is a way to organise regular deadlines and to encourage teamwork towards clearly defined objectives, while avoiding lock down to inflexible targets. Scrum is less about the code and more about how to split development time into manageable chunks, how to structure implementation iterations, and how to filter and prioritise ideas for new features and technical changes.

This section summarises the agile process in use for LarKC WP7 software development. The main elements of XP and Scrum that we have adopted are:

- Test-driven development.
- Deliver early, deliver often.

- Only document where necessary (minimise design artefacts because they go out of date or cost lots to maintain).
- Maintain our library-based approach and refactor code into the frameworks in and around GATE. (Resist the temptation to reassure bureaucrats and marketing people of the novelty of the work by continually inventing new names.)
- Maintain our continuous integration suite with (minimally) nightly builds.
- Development done by self-organising teams.
- Development done on a bi-weekly or monthly cycle; no new work can be taken on by a team after a cycle starts.
- The list of features, functions, technologies, enhancements and bug fixes that would ideally be performed are prioritised and listed in a “backlog” which represents a snapshot of the ever-changing requirements and plan.
- For each development cycle pick the highest priority items to do next.
- There are no bad ideas for new features, only low priority features.
- Meet every 2 working days to briefly report: what’s been done since last meeting; what is planned for next; what has been a barrier. These are called “scrums” and should last around 15 minutes.

The process:

- Develop a prioritised list of functionality, technology changes, bug fixes etc.
- Select a month’s work (or a week or a fortnight, depending on external factors).
- Do a month’s work (a “sprint”).
- Review.
- Repeat.

During a sprint new work items cannot be assigned to the developers unless they are of an urgent, show-stopper nature. (New items that are identified are added to the backlog instead.) The team can reduce or increase the amount of work in the sprint if it needs to, but tries to avoid reductions if possible.

As new functionality becomes available it is rolled into the applications and user feedback sought.

Artefacts:

- Applications.
- Backlog. This lists all new and changed functionality that we've decided we need to develop to improve the applications. Items at the top of the list should be detailed enough and fine-grained enough to be implemented.
- Test suite. All the backlog items that the team produces are delivered as a set of tests.
- User guide. This has to contain enough information for users to understand what the team outputs and to exploit it effectively.
- Developer guide. This is written relatively informally as a set of notes that point into the test suite javadocs/java2html.
- Documentation process: the sequence is: vision; backlog to-do; backlog done, linked to developer or user documentation.

Meetings:

- First Monday of the month: sprint end review, sprint backlog elements selection. (Changes when a sprint finishes early. Sprints can't finish late: if the work isn't complete when they end we just stop, replan, and start again.) A sprint end review looks at each backlog item that was planned and asks "Do tests exist that cover this item, and do they succeed?" and "Is the customer using this, and if not then when?"
- Monday/Wednesday/Friday bi-weekly and Tuesday/Thursday bi-weekly alternating: team (scrum) meeting. The team meeting is very short; each member summarises what they did since the last meeting, what they plan to do next, and anything that is a problem (technological, organisational, whatever).
- Alternate Thursdays: management meeting (analyse problems raised in team meetings, check backlog priorities, review code).
- As required: user meetings. When we're the users or when the users are remote we have to simulate them. In XP style new development is rolled into the applications as quickly as possible.

BIBLIOGRAPHY

- [1] K. Beck. *eXtreme Programming eXplained*. Addison-Wesley, Upper Saddle River, NJ, USA, 2000.
- [2] K. Bontcheva, V. Tablan, D. Maynard, and H. Cunningham. Evolving GATE to Meet New Challenges in Language Engineering. *Natural Language Engineering*, 10(3/4):349—373, 2004.
- [3] G. Booch. *Object-Oriented Analysis and Design 2nd Edn*. Benjamin/Cummings, 1994.
- [4] S. Chakrabarti. *Mining the Web – Discovering Knowledge from Hypertext Data*. Morgan Kaufmann Publishers, 2003.
- [5] G. Cheng, W. Ge, and Y. Qu. Falcons: Searching and Browsing Entities on the Semantic Web. In *Proceedings of WWW2008*, pages 1101–1102, 2008.
- [6] S. Clark, B. Coecke, and M. Sadrzadeh. A compositional distributional model of meaning. In *Proceedings of the Second Quantum Interaction Symposium (QI-2008)*. College Publications, 2008.
- [7] H. Cunningham. GATE, a General Architecture for Text Engineering. *Computers and the Humanities*, 36:223–254, 2002.
- [8] H. Cunningham. Information Extraction, Automatic. *Encyclopedia of Language and Linguistics, 2nd Edition*, pages 665–677, 2005.
- [9] H. Cunningham and K. Bontcheva. Computational Language Systems, Architectures. *Encyclopedia of Language and Linguistics, 2nd Edition*, pages 733–752, 2005.
- [10] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, 2002.
- [11] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.
- [12] M. Dowman, V. Tablan, H. Cunningham, and B. Popov. Web-assisted annotation, semantic indexing and search of television and radio news. In *Proceedings of the 14th International World Wide Web Conference*, Chiba, Japan, 2005. <http://gate.ac.uk/sale/www05/web-assisted-annotation.pdf>.

- [13] D. Fensel, F. van Harmelen, B. Andersson, P. Brennan, H. Cunningham, E. D. Valle, F. Fischer, Z. Huang, A. Kiryakov, T. K. Lee, L. School, V. Tresp, S. Wesner, M. Witbrock, and N. Zhong. Towards larkc: a platform for web-scale reasoning. In *Proceedings of the IEEE International Conference on Semantic Computing (ICSC 2008)*, Santa Clara, CA, USA, 2008. IEEE Computer Society Press.
- [14] A. Fokoue, A. Kershenbaum, L. Ma, E. Schonberg, and K. Srinivas. The summary abox: Cutting ontologies down to size. In *Proc. of the Int. Semantic Web Conf. (ISWC2006)*, pages 136–145, 2006.
- [15] Jr. Frederick P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1978.
- [16] J. Karlgren and M. Sahlgren. From words to understanding. In Y. Uesaka, P. Kanerva, and H. Asoh, editors, *Foundations of Real-World Intelligence*, pages 294–308. Stanford: CSLI Publications, 2001.
- [17] A. Kiryakov, B. Popov, D. Ognyanoff, D. Manov, A. Kirilov, and M. Goranov. Semantic annotation, indexing and retrieval. *Journal of Web Semantics, ISWC 2003 Special Issue*, 1(2):671–680, 2004.
- [18] Atanas Kiryakov. OWLIM: balancing between scalable repository and light-weight reasoner. In *Proc. of WWW2006*, Edinburgh, Scotland, 2006.
- [19] Y. Li, K. Bontcheva, and H. Cunningham. SVM Based Learning System For Information Extraction. In M. Niranjan J. Winkler and N. Lawrence, editors, *Deterministic and Statistical Methods in Machine Learning*, LNAI 3635, pages 319–339. Springer Verlag, 2005.
- [20] Y. Li, K. Bontcheva, and H. Cunningham. Using Uneven Margins SVM and Perceptron for Information Extraction. In *Proceedings of Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, 2005.
- [21] Y. Li, K. Bontcheva, and H. Cunningham. Hierarchical, Perceptron-like Learning for Ontology Based Information Extraction. In *16th International World Wide Web Conference (WWW2007)*, pages 777–786, May 2007.
- [22] D. Nelson and C. McEvoy. Entangled Associative Structures and Context. In *Proceedings of the AAAI Spring Symposium on Quantum Interaction*. AAAI Press, 2007.
- [23] D. Pavlovic. On quantum statistics in data analysis. In *Proceedings of the Second Quantum Interaction Symposium (QI-2008)*. College Publications, 2008.

- [24] B. Popov, A. Kiryakov, A. Kirilov, D. Manov, D. Ognyanoff, and M. Goranov. KIM – A semantic platform for information extraction and retrieval. *Natural Language Engineering*, 10:375–392, 2004.
- [25] Y. Qu, W. Hu, and G. Cheng. Constructing virtual documents for ontology matching. In *Proceedings of WWW2006*, pages 23–31, 2006.
- [26] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.
- [27] K. van Rijsbergen. *The Geometry of Information Retrieval*. Cambridge, 2004.
- [28] D. Widdows. Orthogonal negation in vector spaces for modelling word-meanings and document retrieval. In *The 41st Annual Meeting of the Association for Computational Linguistics*, pages 136–143, 2003.
- [29] D. Widdows. Semantic vector products: Some initial investigations. In *Proceedings of the Second Quantum Interaction Symposium (QI-2008)*. College Publications, 2008.
- [30] D. Widdows and P. Bruza. Quantum information dynamics and open world science. In *Proceedings of the First Quantum Interaction Symposium (QI-2007)*. AAAI Press, 2007.
- [31] D. Widdows and S. Peters. Word vectors and quantum logic: Experiments with negation and disjunction. In *Eighth Mathematics of Language Conference*, pages 141–154, 2003.
- [32] E. Yourdon. *Modern Structured Analysis*. Prentice Hall, New York, 1989.