



NoTube

*Networks and ontologies for the transformation and unification of  
broadcasting and the Internet*

FP7 – 231761

---

## D4.4 Text content analysis components

---

**Coordinator: D. Kozhuharov (OT),**

**With contributions from: M. Yankova (OT)**

**Quality Assessor: Davide Palmisano (AS)**

**Quality Controller: Milena Yankova (OT)**

Document Identifier:	NoTube/2010/D4.4
Version:	0.4
Date:	20/02/2010
State:	Final
Distribution:	PU

## EXECUTIVE SUMMARY

This deliverable is a continuation of the deliverable D4.2. It describes the LUPedia text enrichment web service and how it was used to process the BBC use-case data (**Scenario 7c**). Since the Large Knowledge Base Gazetteer is the core component of the presented web service, the deliverable contains its overview, which covers its theoretical and structural details.

## DOCUMENT INFORMATION

<b>IST Project Number</b>	FP7 - 231761	<b>Acronym</b>	NoTube
<b>Full Title</b>	Text content analysis components		
<b>Project URL</b>	http://www.notube.eu/		
<b>Document URL</b>			
<b>EU Project Officer</b>	Francesco BARBATO		

<b>Deliverable</b>	<b>Number</b>	4.4	<b>Title</b>	Text content analysis components
<b>Work Package</b>	<b>Number</b>	4	<b>Title</b>	Metadata Acquisition and Enrichment












<b>Date of Delivery</b>	<b>Contractual</b>	M 13	<b>Actual</b>	
<b>Status</b>	version 0.3		final <input type="checkbox"/>	
<b>Nature</b>	prototype <input checked="" type="checkbox"/> report <input type="checkbox"/> dissemination <input type="checkbox"/>			
<b>Dissemination level</b>	public <input checked="" type="checkbox"/> consortium <input type="checkbox"/>			

<b>Authors (Partner)</b>	OT			
<b>Responsible Author</b>	<b>Name</b>	Yankova	<b>E-mail</b>	milena.yankova@sirma.bg
	<b>Partner</b>	OT	<b>Phone</b>	

<b>Abstract (for dissemination)</b>	This deliverable is a continuation of the deliverable D4.2. It describes the LUPedia text enrichment web service and how it was used to process the BBC use-case data ( <b>Scenario 7c</b> ). Since the Large Knowledge Base Gazetteer is the core component of the presented web service, the deliverable contains its overview, which covers its theoretical and structural details.
<b>Keywords</b>	Text Analysis

<b>Version Log</b>			
<b>Issue Date</b>	<b>Rev. No.</b>	<b>Author</b>	<b>Change</b>
25.01.2010	0.1	D.Kozhuharov	Draft of LKB Gazetteer component description
27.01.2010	0.2	D.Kozhuharov	Draft of LUPedia description
28.01.2010	0.3	D.Kozhuharov	Draft of BBC use-case handling
20.02.2010	0.4	M.Yankova	Post review changes

## PROJECT CONSORTIUM INFORMATION

Participants		Contact
Vrije Universiteit Amsterdam		Guus Schreiber Phone: +31 20 598 7739/7718 Email: <a href="mailto:schreiber@cs.vu.nl">schreiber@cs.vu.nl</a>
British Broadcasting Corporation		Libby Miller Phone: Email: <a href="mailto:libby.miller@bbc.co.uk">libby.miller@bbc.co.uk</a>
Asemantics SRL Uninomiale		Alberto Reggiori Phone: +39 0639 7510 78 Email: <a href="mailto:alberto@asemantics.com">alberto@asemantics.com</a>
Engin Medya Hizmetleri A.S.		Ron van der Heiden Phone: +31 6 2003 2006 Email: <a href="mailto:ron@engin.tv">ron@engin.tv</a>
Institut fuer Rundfunktechnik GmbH		Christoph Dosch Phone: +49 89 32399 349 Email: <a href="mailto:dosch@irt.de">dosch@irt.de</a>
Ontotext AD		Atanas Kiryakov Phone: +35 928 091 565 Email: <a href="mailto:naso@sirma.bg">naso@sirma.bg</a>
Open University		John Domingue Phone: +44 1908 655 014 Email: <a href="mailto:j.b.domingue@open.ac.uk">j.b.domingue@open.ac.uk</a>
RAI Radiotelevisione Italiana SPA		Alberto Morello Phone: +39 011 810 31 07 Email: <a href="mailto:a.morello@rai.it">a.morello@rai.it</a>
Semantic Technology Institute International		Lyndon Nixon Phone: +43 1 23 64 002 Email: <a href="mailto:lyndon.nixon@sti2.org">lyndon.nixon@sti2.org</a>
Stoneroos B.V.		Annelies Kaptein Phone: +31 35 628 47 22 Email: <a href="mailto:annelies.kaptein@stoneroos">annelies.kaptein@stoneroos</a>
Thomson Grass Valley France SA		Raoul Monnier Phone: +33 2 99 27 30 57 Email: <a href="mailto:raoul.monnier@thomson.nett">raoul.monnier@thomson.nett</a>
TXT Polymedia SPA		Sergio Gusmeroli Phone: +39 02 2577 1310 Email: <a href="mailto:sergio.gusmeroli@txtgroup.com">sergio.gusmeroli@txtgroup.com</a>
KT Corporation		Myoung-Wan Koo Phone: +82 2 526 5090 Email: <a href="mailto:mwkoo@kt.com">mwkoo@kt.com</a>

## **TABLE OF CONTENTS**

<b>LIST OF FIGURES .....</b>	<b>7</b>
<b>LIST OF ACRONYMS.....</b>	<b>8</b>
<b>1. INTRODUCTION .....</b>	<b>9</b>
1.1. SCOPE OF THIS DELIVERABLE.....	9
<b>2. OVERVIEW OF THE “LARGE KNOWLEDGE BASE GAZETTEER” .....</b>	<b>10</b>
2.1. THE GAZETTEER STORY IN A NUT SHELL .....	10
2.2. DEFINITIONS OF USED TERMS.....	10
2.3. THEORETIC BACKGROUNDS OF THE LKB GAZETTEER.....	11
2.4. INTERNAL GAZETTEER STRUCTURE OVERVIEW.....	15
2.5. EXTERNAL GAZETTEER STRUCTURE OVERVIEW.....	17
2.6. SPECIFIC BUSINESS VALUES .....	19
<b>3. LUPEDIA WEB SERVICE .....</b>	<b>20</b>
3.1. OVERVIEW.....	20
3.2. LUPEDIA INTERNAL STRUCTURE.....	21
3.3. SUPPORTED RESPONSE FORMATS .....	22
<b>4. BBC USE-CASE HANDLING .....</b>	<b>23</b>
4.1. BBC USE-CASE REFERENCE .....	23
4.2. LOADED DICTIONARY DATA.....	23
4.3. BBC PROGRAMS TEXT DATA.....	24
4.4. PROCESSED BBC PROGRAM TEXT DATA .....	24
<b>5. CONCLUSION .....</b>	<b>26</b>
<b>6. REFERENCES .....</b>	<b>27</b>

## List of Figures

Fig 1. LUPedia web user interface .....	20
Fig 2. Resulting page from LUPedia web user interface.....	22
Fig 3. Example of XML response .....	23
Fig 4. BBC programs rdf description .....	24
Fig 5. Original BBC web-site content .....	25
Fig 6. The User Interface for access to the processed BBC data.....	25

## List of Acronyms

<b><u>Acronym</u></b>	<b><u>Description</u></b>
<b>KB</b>	Knowledge Base
<b>LKB</b>	Large Knowledge Base
<b>URI</b>	Uniform Resource Identifier
<b>LOD</b>	Linking Open Data
<b>LDSR</b>	Linked Data Semantic Repository
<b>REST</b>	Representational State Transfer
<b>HTTP</b>	Hypertext Transfer Protocol

## 1. Introduction

Text content analysis components aim at processing human readable text content in order to produce formal knowledge and enrich the text with corresponding metadata. They utilize Semantic Annotation (SA) means to discover entities that are mentioned as well as their concrete position in the text.

One of the basic SA approaches is to discover the references to entities information for each is already accessible for the system in the text. It is performed using pre-existing knowledge base of objects and including their textual representations (e.g. main label, aliases, abbreviations, etc). Therefore it is important to have good data source in terms of accessibility and domain coverage in order to achieve good results.

Nowadays the amount of publicly accessible knowledge bases grows and one of the substantial sets of knowledge is the Linking Open Data (LOD) cloud <sup>[3]</sup>. We explore this knowledge base using Linked Data Semantic Repository (LDSR, <http://ldsr.ontotext.com>) that contains several of the most central datasets of LOD and offers a public SPARQL endpoint that allows for independent access to this information. One of the included sets is the DBPedia (<http://dbpedia.org>) data set. It was chosen to serve as a basic knowledge source due to its universality and wide coverage.

Laying the basis for the NoTube text enrichment components and following the best practices as described in D.4.1 [4] survey, we start with producing text lookups based on the 2.9 Millions things described in DBPedia. This is the necessary step for proceeding with more advanced enrichment techniques as Named Entity Recognition and Identity Resolution as described in D4.1. The challenge here is to design and implement a scalable algorithm that is capable to work with huge amount of data in acceptable time required for the other NoTube services that need real time text enrichment.

The core component of the presented lookup service called LUPedia is an implementation of in-memory lookup algorithm. It has been named after the complexity of used dataset – Large Knowledge Base Gazetteer (LKBG) and is currently tested on two data sets – DBPedia and FreeBase (<http://www.freebase.com/>). Their implementation showed clearly the big potential of using LKBG component loaded with other big volume data sets as the rest of the LOD cloud.

### 1.1. Scope of This Deliverable

The goal of this deliverable is to explain the algorithm behind LKBG and its usage in two scenarios (1) how the data set of DBPedia is utilized for discovery and enrichment and (2) how it is used for handling of the BBC use-case data.

Since the Large Knowledge Base Gazetteer is the core component of the web service presented here, the presentation begins with its overview. After some historic review and explanation of some terms the overview continues with explanation of the theoretic background behind the LKB Gazetteer. Then the internal and external structure of the Gazetteer is presented. Overview ends with a summary of the specific business values of the Gazetteer.

Chapter 3 discusses the LUPedia web service and how it integrates the DBPedia KB, the LKB Gazetteer and how this is available for usage. The following chapter 4 is about how the BBC use-case data (**Scenario 7c**) was processed using LUPedia web service and how it is made accessible. Finally we draw some conclusions.

## 2. Overview of the “Large Knowledge Base Gazetteer”

### 2.1. *The Gazetteer story in a nut shell*

The common usage of the word “gazetteer” is as: “... a reference for information about places and place names, used in conjunction with a map or a full atlas ...”<sup>(1)</sup>. Further in the past – the word originates from the profession of a man who prepares the news-paper content while using large indexes of geographic names.

In modern times the term was adopted by the text engineering society. It was then used to name software components that mark-up text elements with special meaning. Then with time the range of things recognized by the Gazetteer components expanded.

A substantial evolution was the idea to use the Gazetteers not only to recognize special words or abbreviations, but also publicly known names. Recognition of public names in text and the context knowledge it brings is important for the overall understanding of the content. This role shift for the Gazetteers imposed the differentiation of a specialized internal Dictionary unit. A separate process was established to fill the Dictionary unit with name data before operations.

The Large Knowledge Base Gazetteer is a next step in the evolution of these components. Its creation was motivated by the presence of large publically accessible bodies of formal knowledge. The primary goal was to make possible loading millions of dictionary data records and using them for look up while maintaining high levels of productivity. Later, when this functionality appeared applicable in many diverse use-cases, the flexibility and interoperability became another important goal for the LKB Gazetteer.

Before we step further into the structural and functional details here follows a short description of often used terms.

### 2.2. *Definitions of used terms*

- **Knowledge Base (KB)** – here a **Knowledge Base** will be considered a set of data that describes some objects through their properties and their mutual relations.
- **Entity** – an **Entity** is a known object described in a **KB** with its properties and relations.
- **Entity Alias** – the **Alias** is the textual value of a specific **Entity** property. This property must possess the potential to represent the Entity in texts.
- **Alias Redundancy Potential** – the **Redundancy Potential** represents the possibility for a given **Alias** to appear as property value of more than one **Entity**. The low **Redundancy Potential** depends also on the type of the property linking the **Alias** with the **Entity**.
- **Alias Predicate** – the **Predicate** of an **Alias** is the particular type of the property that linked the **Alias** to an **Entity**.
- **Entity Class** – the **Class** of an **Entity** is the value of particular type-property of the Entity. It links the **Entity** to a predetermined classification.
- **Alias Dictionary** – the internal dictionary of the **Gazetteer** which consists of **Alias** index linking aliases with small data-digests. It allows searching by given **Alias** and retrieval of the related data-digest.
- **Dictionary Record** – this is the basic information unit used in the Alias Dictionary. One Dictionary record contains the following fields:
  - the Alias;
  - the identifier of the Entity linked with the Alias;
  - the identifier of the Class of the linked Entity;
  - the identifier of the Predicate that linked the Alias with the Entity;

Here the Alias is the detectable part of the Dictionary Record and the other three fields are the deliverable part of the record.

- **Look-Up Parsing** – process of picking fragments of a text and performing with them look ups in the **Alias Dictionary**. The look up aims to match the given text fragment with the detectable part of some Dictionary Records.
- **Lookup** – a combination of a Dictionary Records returned by a look up in the Dictionary and couple of text offsets. The offsets justify the place in text where the text fragment for the Dictionary look up was retrieved.

### 2.3. Theoretic Backgrounds of the LKB Gazetteer

#### 2.3.1. Definition of the Main Gazetteer Function

The main function of a Gazetteer is to perform Look-Up Parsing of a given text and to produce a list of Lookups.

#### 2.3.2. Valid Lookup location

While parsing a text before every look up in the dictionary – the Gazetteer chooses a location and extracts the fragment of the text to pass it to the Dictionary.

The most delicate element here is the choice of rules that determine which fragments of a text should be considered valid Lookup locations. The main criterion for that choice is that the fragment taken alone must have the same meaning as it has as part of the whole text.

Typical approach is to split the set of characters used in text on two sub-sets:

- **Sense-bearing Characters** – this set contains the characters that are used to construct the substantial part of the Aliases that distinguish them by one another;
- **Delimiter Characters** – this set contains the characters that are used to delimit between Aliases in texts and if included in the Alias itself – they do not affect substantially its meaning.

So having these two sets – it is commonly accepted that:

The start and end offsets of a valid Lookup location must not break sequences of Sense-bearing Characters. This relates to the notion that a regular human-readable text is composed of separate words constructed of Sense-bearing characters.

For example if letters are sense-bearers and spaces are separators then we have:

“walking streets of New York” – is an invalid location example even though “king street” could be a valid name;

“walking streets of New York” – is a valid location example;

#### 2.3.3. Tokenization

- *Validity criteria diversity*

Even though we set some grounds to determine which locations in text could be valid Alias occurrences – there is a great variety of cases in practice. The variations could reflect as the content of the two character sets as also the rule for breaking of the Sense-Bearing characters.

For example the digits could be considered sense bearers as in the case: “ ... Apolo-1 **1** and Apolo-1**2** landed ...” or splitting could be allowed between lower-to-upper case letter transitions as in “TheTalesOf**NewYork**Zoo”.

- *Tokenization*

To reflect this complexity and retain flexibility Gazetteers use as separate preparatory process the Tokenization. This process transforms the text into sequence of Tokens which could be defined as:

- **Token** – an inseparable piece of the text which serves as the building block of text's semantic. That way if we split a Token this will alter the original text meaning;

Having a pre-tokenized text simplifies the task for determining the valid Lookup locations. They are all the locations which do not break apart Tokens.

#### **2.3.4. Token Phrases**

- *Phrase of Tokens*

If we review the definition of the Alias we could consider that it is a kind of name. So we could treat it as meaningful piece of text and hence apply Tokenization over it. Transforming an Alias into a sequence of Tokens shows that it could be consider as a Token Phrase.

- *Looking up by Tokens*

Having transformed the text and the Aliases into Token sequences we can concretize that:

***Look-Up Parsing*** process is picking Token Phrases from the Tokenized text and looking up such Token Phrases / Aliases in the Alias Dictionary.

This concretization allow to substantially reduce the number of look ups in the dictionary compared to just checking for any text fragment.

- *Phrase expansion*

As we defined that the Aliases are Token Phrases it comes apparent that the Look-Up Parsing process will not only check single Tokens in the Dictionary, but also multi-token phrases. So the Gazetteer makes a look up for a Token, then it must check the phrase comprised of that Token and the one next to it. Then iteratively it continues to add next Token and check for the new phrase. This process is called Phrase Expansion.

- *Valid Token Prefix Register*

From the fact that an Alias Token Phrase spans several Tokens while a Tokenized text can span hundreds of Tokens it comes that there have to be a reasonable criteria to limit the Phrase Expansion.

Such criteria would be if at every step of Phrase Expansion we could check if the Token sequence of the checked Phrase appears as beginning of the Token Phrase of an Alias stored in the dictionary.

This brings the need that the Alias Dictionary to supports a Valid Token Prefix Register. This register will be generated at dictionary population time. It will allow fast checks if a given Token Phrase happens to be any aliases prefix.

Having this function implemented by the dictionary allows for further substantial reduction of the number of look ups, which will increase productivity.

#### **2.3.5. Sense-bearing Tokens**

- *Term introduction*

Substantial for the overall productivity of the Gazetteer is the reduction of the number of dictionary look ups it makes. That is why in the LKB Gazetteer another

step is taken in that direction. The look up process is focused on the Sense-bearing Tokens. Here are some definitions:

- **Sense-bearing Token** – a token comprised entirely of Sense-bearing characters.
- **Delimiter Token** – a token comprised entirely of Delimiter characters.
- **Sense-bearing Token Phrase** – a Token Phrase comprised entirely of Sense-bearing characters. It is generated as a normalization of a regular Token Phrase by removing all Delimiter Tokens from it.

Here it is assumed that the Tokenization process is designed so that Sense-bearing and Delimiter characters are not mixed within a Token;

- *Looking up by Sense-bearing Tokens*

When the Gazetteer parses a text rich of Delimiter Characters it will generate a lot of look up requests to the Dictionary with Token Phrases that differ only in the Delimiter Tokens. All this requests actually aim at retrieving one and the same Alias.

For example having the text [... in "Table Inc.", ...] could result in look up requests for: ["Table Inc], ["Table Inc.], ["Table Inc."], ["Table Inc."], etc.

The approach chosen in the LKB Gazetteer is to ignore Delimiter Tokens and process the text only stepping on the Sense-bearing Tokens. Then the look up request to the dictionary will only have Sense-bearing Token Phrases. The Dictionary is modified so that it will return a short record-set with all the Aliases that have same as requested Sense-bearing Token Phrase.

Then for every Record returned by the Dictionary look up – additional check is made if its Alias matches exactly the text-area where the request Sense-bearing Token Phrases was extracted.

It is accepted that in most cases there will be just one record returned and the additional check for exact-matching of the Alias in the text is cheaper than a dictionary look up.

This look up approach also "leaves the door open" for implementation of "fuzzy look ups", where just matching of the Sense-bearing Token Phrase will be considered enough strong criteria that the Alias was mentioned in the text.

### **2.3.6. Text Identification Through Hash Codes**

- *Text comparison through hash codes.*

Hash functions are defined as:

"... any well-defined procedure or mathematical function that converts a large, possibly variable-sized amount of data into a small datum, usually a single integer ..."<sup>(2)</sup>

This makes them a great candidate for text compression method. Text fragments can be transformed to hash codes and then we could compare just the hash codes. This will allow to store in dictionary, the hash codes of the Aliases, which will greatly reduce needed dictionary storage space. On the other hand comparing integers is much faster than comparing strings.

But the great advantages always come on a price.

- *The Drawback*

"A hash function may map two or more keys to the same hash value."<sup>(2)</sup>

This function property will cause false-positive detections when a text fragment from the processed text generates the same hash-code as an Alias from the dictionary, while both corresponding texts are different.

- *Relieving the drawback*

A hash function will be used to transform the large set of all possible texts into the smaller set of the integer numbers. This makes it obvious that lots of texts will be mapped to one hash code.

The first relief here is that from all texts that map to one hash code just one or two will be meaningful. Others will be random character sequences.

The second relief is that Aliases are not of arbitrary length. They are of limited size, which in turn reduces their variety and hence the chance two meaningful names to have equal hash codes.

Now let's look at it statistically. Performing some tests with real world name data we will see that 1/1000 is good approximation for the amount of false-positive matches using an integer hash-code. This could be acceptable in some cases but unacceptable in others. To remedy that – the current implementation of the LKB Gazetteer uses generation of second independent hash-code. This will bring the amount of false-positive matches to 1/1000000.

This means that in a dictionary of 10 million names we will have 10 Aliases that could be false-positively recognized in text if their hash-code counterparts appear in the processed text.

Finally – here is a practical perspective on the problem. Maintaining high quality of data for dictionaries of millions of elements is very difficult. So naturally the errors inflicted by the usage of hash-codes will be an order of magnitude less than the errors due to dictionary data quality issues.

### **2.3.7. Alias Encoding Using Hash Codes**

- *The first hash-code*

As mentioned above – the LKB Gazetteer uses two independent hash-codes to identify a single Alias. The first hash-code is generated over the Sense-bearing Token Phrase of the Alias. This is inline with the search model based on the Sense-bearing Phrases. The first hash-code has the following usages:

- **To group Dictionary Records** – the internal structure of the Alias Dictionary is organized so that it groups the dictionary records with respect to their first hash-code. This grouping allows also that the hash-code value is stored just once for all dictionary records in the group.
- **As Dictionary look up criterion** – after the LKB Gazetteer locates a Sense-bearing Token Phrase it calculates its hash-code and uses it as look up criterion for a dictionary request.
- **To encode Valid Token Prefixes** – before storing an Alias .into the Dictionary – the LKB Gazetteer retrieves its Sense-bearing Token Phrase. Then all the Token Prefixes of that phrase are enumerated and their hash-code calculated and stored in the Valid Token Prefix Register.
- **To check for Valid Token Prefix** – after a hash-code has been calculated by the LKB Gazetteer to use it for Dictionary look up, it is used to check in the Valid Token Prefix Register. If the hash-code is found there – then Phrase Expansion is allowed.

- *The second hash-code*

The second hash-code is generated over the original text of the Alias. It is used for the secondary exact match check. This check verifies that the original Alias text together with the delimiter set characters exactly matches the area of the text that the gazetteer is inspecting.

- *Single dictionary record structure*

The Dictionary Record is a basic element that contains the payload data of the Dictionary. The record of the LKB Gazetteer contains four substantial fields:

- **Alias** – This is the detectable part of the record. It is originally a text fragment that will be localized in the texts. In the case of the LKB Gazetteer – the Alias text is represented by a couple of hash-codes.
  - **Entity** – This is the identity that stands behind the Alias. They have a certain type of relation and the Alias is representative for that Entity.
  - **Class** – This is a reference to a predefined classification. It states that the identity behind the Alias is of a particular Class.
  - **Predicate** – This is the type of the relation that links the Alias with its corresponding identity. The Predicate justifies the strength and nature of the link. This allows for better reasoning – what in fact the appearance of the Alias in a text means.
- *Valid Token Prefix Register*  
As mentioned above – the Valid Token Prefix Register for the LKB Gazetteer is based upon the first hash-code of the aliases. This allows for it to be implemented as a set of hash-codes (integer numbers), which produces a very small memory footprint and also very low check times.

The drawback is that relying on a single hash-code will increase the possibility of false positive check results. Never the less – the cost of this errors is one unnecessary Token Phrase Expansion, which is quite acceptable.

## **2.4. Internal Gazetteer Structure Overview**

After the last structural redesign of the LKB Gazetteer its code was divided in two major structures – the Internal and the External Gazetteer structures.

The Internal structure contains the core functionality that performs the Main Gazetteer Function as defined earlier. It is tuned for the maximum performance and is expected to remain relatively stable in time. This functional set is split on several units which are described further:

### **2.4.1. Parsing Frame unit**

The first function of this unit is to perform the Tokenization of text. The Tokenization is configured through a standard Java Regular Expression and performed by the standard Java Regular Expression Parser.

The texts are expected in UTF-8 format and the regular expression uses Java Character Classes for Letters and Digits. This makes the Tokenization and the LKB Gazetteer as a whole – language independent.

The second function is to allow crawling of the Sense-bearing Token Phrases. It exposes a “frame” interface that:

- *allows Frame to be slid over the text;*
- *allows Token Phrase Expansion;*
- *allows a complex view over the current content of the frame – this includes calculation of the first and second hash-codes;*

This unit is shared between the Alias Dictionary unit and the Look Up Parsing unit. This is done because the identical Tokenization of the Aliases and the Processed text is crucial for the proper matching.

### **2.4.2. Alias Dictionary unit**

This unit implements the functionality of a Gazetteer’s Dictionary. It implements the following functions:

- *Static Dictionary concept*

The standard scenario for usage of the Dictionary by the Gazetteer is that it is populated with data before the exploitation starts. This type of Dictionaries is

referred to as Static Dictionaries. Since the process of Dictionary generation could be quite heavy – a serialization to file is implemented. This allows in consecutive initializations of the Gazetteer – the Static Dictionary to be loaded faster from its serialized copy.

To address the cases of multi-process usage of the LKB Gazetteer, the Static Dictionaries are loaded as static memory resources. The serialization folder of the Static Dictionary is used as its unique identifier. A static Map with keys – the serialization folders holds the references of all dictionaries loaded in memory.

- *Dynamic Dictionary concept*

In the context of several use-cases emerged the idea of the Dynamic Dictionary. In contrast with the Static one – it is created empty just before the Gazetteer starts processing a document. Then the Dynamic Dictionary is incrementally filled just before every look up request from the Look Up Parsing engine using ad-hoc requests towards a data source. The dictionary is removed after the processing of the document ends.

Here is a list of the specific use-cases when usage of a Dynamic Dictionary could be beneficial:

- **Very Large Knowledge Base** – This is the case when the size of the knowledge base could not fit within the server's memory restrictions even with the high compression that Alias Dictionaries provide. This implementation scenario has a substantially reduced productivity, but allows the usage of the Gazetteer's framework and could be an enabling solution in extreme cases.
- **Dynamic Knowledge Base** – In this case the knowledge base is being constantly enriched and/or altered. It is also a needed that every next document be processed with the most complete and up to the minute knowledge. In this case the Dynamic Dictionaries are enabling technology.
- **Specific Context Knowledge** – In this case the document being processed carries additional information of the document's context. Also there is explicit need for specific knowledge to be applied for the document processing in the different contexts. In this case the Dynamic Dictionaries are also enabling technology.
- **Combined Static plus Dynamic data usage** – In real use-cases it is most probable that there will be no clear situation as the previous three. That is why the LKB Gazetteer supports the simultaneous usage of the two kinds of dictionaries. This allows usage of a large dictionary with static data together with small dynamically loaded dictionary.

- *Look-Up & Retrieval of Alias records from the Dictionaries*

There are two ways a look up could be performed.

The first is by providing a Parsing Frame unit loaded with given text. Then the look up is executed for the current content of the Frame.

The second is by directly providing the text of an alias. Then the Dictionary creates internally a Parsing Frame over the text and calls the first routine.

- *Additional Dictionary state information services*

- **Valid Token Prefix Check** – given a Parsing Frame – the Dictionary checks if the first hash-code exists in the register of Valid Token Prefix hash-codes.
- **Count of all Aliases/Entities** – returns the current count of the unique Dictionary Records (Aliases) or unique instances(Entities) stored in the Dictionary.
- **Stored Instance Checks** – given an instance URI – the Dictionary checks there was any data stored about this instance.

### 2.4.3. Look Up Parser unit

This unit implements the main Gazetteer's function. Here are some important features of its implementation:

- *Optimized Lookup algorithm*

As mentioned before – the look up algorithm of the LKB Gazetteer is optimized to do a minimal number of look up requests to the Dictionary. This is achieved through working only with Sense-bearing Tokens and having sophisticated criterion for Token Phrase Expansion limit.

After a look up request to the Dictionary returns a list of Dictionary Records, this unit performs the exact match check for each of them using their second hash-code. The records which pass this test are returned as Lookups.

- *Simultaneous usage of Dictionaries*

The Look Up Parser algorithm is modeled so that it allows parallel exploitation of a Static and a Dynamic dictionary.

### 2.4.4. Gazetteer Core unit

This unit is implemented as the single class **LKBGazCore**. It is responsible for the initialization and the configuration of all internal Gazetteer components. It also serves as the initial contact point with the External Gazetteer structure.

## 2.5. External Gazetteer Structure Overview

The External Gazetteer structure has the role of flexible envelop allowing integration of the LKB Gazetteer in wide range of applications. It consists of two Interfaces and an abstract class as also the conceptual description of their intended usage:

### 2.5.1. The Gazetteer Container Interface **LKBGazEmbedder** – This is a call-back type interface intended to allow flexible and light weight embedding of the core Gazetteer functionality.

- *Inversion-Of-Control principle*

The interface is designed in compliance with the Inversion of Control principle so the active side in the performance of the Gazetteer function is the Gazetteer itself and not its container. Here are the searched advantages of this design:

- The container does not have to know anything for the variety of execution modes supported by the Gazetteer supports;
- The container does not have to implement different approaches to calling the Gazetteer.
- The container does not have to manage data-structures for the Gazetteer.
- The container does not have to handle error states of the Gazetteer.

- *LKB Gazetteer Embedder duties.*

- **Instantiate a Gazetteer Core** – the embedder must create an instance of the Core Gazetteer class. The constructor of the core class takes a reference to **LKBGazEmbedder** instance so the Embedder must pass a self reference to the constructor.
- **Initialize the Gazetteer Core** – the embedder must initiate Gazetteer Core initialization when appropriate. At that moment embedder must be ready to provide all the resources the Gazetteer could request.
- **Provide Initialized DictionaryFeeder implementation** – one of the most important resources that the embedder must provide on request is an initialized instance of the **DictionaryFeeder** interface. At this point the embedder could also mate the Data Transformation adapter **FeedTransformer** to the **DictionaryFeeder** implementation and add

Feed Transformation Stages to it (see further description). The embedder must be ready to provide this resource at the moment when the initialization of the Gazetteer core is initiated.

- **Provide General Configuration Parameters** – the embedder is expected to provide values for a set of exploitation mode configuration parameters. Their values must be determined at the moment when the initialization of the Gazetteer core is initiated.
- **Initiate Document Processing** – the embedder must initiate the document processing when it is ready to provide the documents components.
- **Provide Document Content and Environment** – the embedder is expected to provide the text content (as UTF-8 string) and the environment (as key-value feature map) of the document to be processed. They must be ready for provision at the moment when the document processing is initiated.
- **Process Look-Up Findings** – the embedder must implement a call-back method that will receive sequential calls and receive the Lookups that the Gazetteer core has produced. Embedder must implement the utilization of the Lookups in the embedding system context.

**2.5.2. The Data Provider Interface DictionaryFeeder** – This is a combined interface intended to allow flexible implementation of data feeds over arbitrary data sources. The interface allows for the Core Gazetteer to order data feeding when it needs to build a new dictionary and to expose a call-back interface to the Feeder through which it can return data records one by one.

- *Very Needed Data Source Flexibility*

This interface was the first flexibility redesign of the LKB Gazetteer forced by the emerging use-cases. Every use-case has its variations not only over the storage of the dictionary data, but also over its dynamic and quality.

- *Dictionary Feeder duties.*

- **Initialize data sources** – the feeder must initialize and prepare for use its data-sources when init method is invoked.
- **Implement data feeding for Static Dictionaries (optional)** – the Feeder must implement the method for Static Dictionary data feeding. It must feed the whole dictionary data through the `EntityListener` helper interface when requested.
- **Implement data feeding for Dynamic Dictionaries (optional)** – the Feeder must implement the methods providing initialization execution and closure of the Dynamic Dictionary feeding.

**2.5.3. The Data Transformation Container Class FeedTransformer** – This is a container class that can hold a Feed Transformation Pipeline. It is designed to allow transparent insertion between a Dictionary Feeder implementation and the Core Gazetteer. The class defines an abstract helper class `FeedTransformer.Stage`, which serves as a bootstrap for creation of Feed Transformer Stages.

- *Flexible Data Transformation over the large scale*

As appeared from the practice – the common Knowledge Bases have a lot of imperfections and incompatibilities. At the same time the Dictionary Feeding is a process involving large data storage on the one end and hi compression Dictionary on the other end. This imposed the need for flexible and light way of inserting data transformation logic on the path of the data and working on the stream principle.

- *Automated dependency injection and integration*  
The Feed Transformer class exposes a pipeline integration method `addStage`. It takes a full class name as a parameter and implements all necessary actions to instantiate this class and mount it in the Feed Transformation pipeline. The provided class must have extended the bootstrap class `FeedTransformer.Stage`.
- *The Transformation Bootstrap Class `FeedTransformer.Stage`*  
This class appears as a comfortable working place for implementation of data-transformation routines. It offers automated integration and access to some Dictionary state information. The extending class should only implement a single abstract method `dictFeedEntity`.
- *The Transformation Stage Extender duties*  
There are three basic functions that a Transformation Stage could optionally implement:
  - *Data Modification* – this is when some of the components of the fed dictionary record must be changed. As a result the transformer passes further the modified version of the dictionary record.
  - *Data Enrichment* – this is when based on the current record and some equivalent transformations – some new dictionary records are generated. As a result the transformer passes further the original and all the generated dictionary records.
  - *Data Feeding* – this is when the transformer passes the dictionary records it produced to the next component of the feed. The transformer may apply filter logic which will stop from further processing some dictionary records.

## 2.6. Specific Business Values

Here are outlined the properties of the LKB Gazetteer that distinguish it.

### 2.6.1. Very High Dictionary Compression

The LKB Gazetteer includes specialized Alias Dictionary unit that allows extreme levels of compression. This is achieved through:

- *String-to-Hash-Code transformation of Aliases* – allows arbitrary length strings to be encoded as fixed size integers.
- *Entity URI name-space compression* – namespaces of Entity instances are encoded with single number
- *Class and Property URI encoding* – because of relatively small diversity of Class and Property URIs – they are encoded with single integers

Real world tests showed that an average dictionary record size varies between 35 and 45 bytes. This allowed loading and practical usage of the LKB Gazetteer with dictionaries containing tens of millions of entries.

### 2.6.2. Optimal Text Parsing Algorithm

Several measures are taken in the LKB Gazetteer to optimize the speed of the Look Up Parsing process:

- *Sense-bearing Token Parsing* – as described above – the parsing based on Sense-bearing Tokens reduces the number of Dictionary look ups.
- *Phrase Expansion Limitations* – maintaining a register of the valid Token prefixes gives very tight phrase expansion restriction. This allows for very sparing usages of the Dictionary look ups.

### 2.6.3. Support of Dynamic and Static Dictionaries

The support of the two ideologically different Dictionary types allows the LKB Gazetteer to be applicable both in general and in very specific use-case profiles.

### 2.6.4. Great Integration and Configuration Flexibility

The external structure of the LKB Gazetteer was designed under the push of real world use-cases. This made it very adaptable to the environment resources and to the production requirements.

- *Data Source Flexibility* – there is a dedicated interface allowing the LKB Gazetteer to be fed with data from any data source.
- *Data Transformation Flexibility* – there is a dedicated container class allowing implementation of arbitrary transformations of the dictionary data feed.
- *Embedding Flexibility* – there is a dedicated interface that allows the core Gazetteer functionality to be programmatically integrated in any larger software unit.

## 3. LUPedia Web Service

### 3.1. Overview

The LUPedia web service's main function is to recognize public names in texts relying on the Dictionary data retrieved from external KB. At present the Dictionary of the web service uses the DBPedia resources – loaded from the public access point of LDSR.

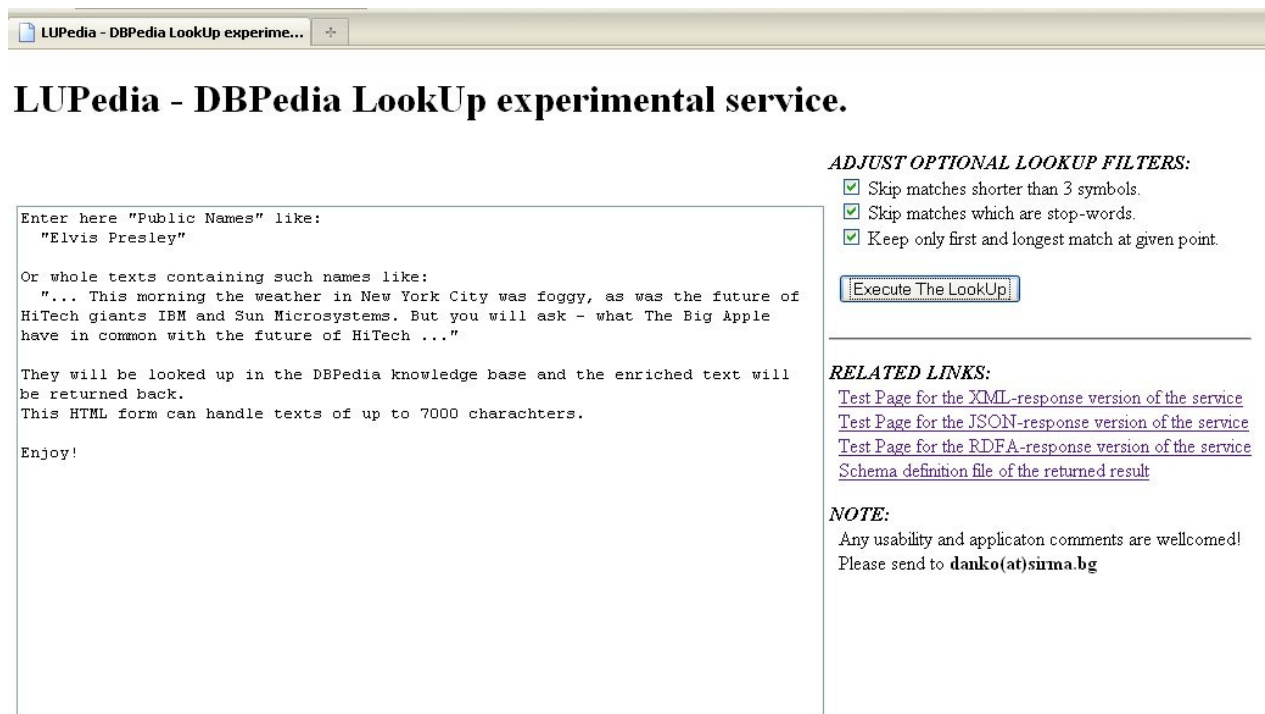


Fig 1. LUPedia web user interface

Linked Data Semantic Repository (LDSR, <http://ldsr.ontotext.com>) is probably the largest and most heterogeneous body of general factual knowledge that was ever used for logical inference. LDSR includes several of the most central datasets of Linking Open Data (LOD) cloud<sup>[3]</sup>. These are: DBPedia, Geonames, UMBEL, Freebase, Wordnet, CIA World Factbook, Lingvoj. It also includes the following schemata: Dublin Core, SKOS, RSS, FOAF. LDSR provides efficient mechanism to query data from multiple datasets and sources, considering their semantics, and represents a reason-able view to the web of data. It supports materialization with respect to the semantics of RDFS, extended with incomplete support of OWL Lite.

We have chosen DBPedia (<http://dbpedia.org/About>) as an RDF dataset derived from Wikipedia, designed and developed to provide as full as possible coverage of the factual knowledge that can be extracted from Wikipedia with a high level of precision. It actually serves as a hub for the LOD project and using in as starting point for lookups allows for further processing and enrichment of the original textual content.

LUPedia is accessible as REST service completed with a set of HTML test pages serving as web user interface (see Fig 1). Each of them contains a test form which allows submission of text and filter parameters to each of the four endpoints of the web service see the resulting page shown on Fig 2.

### 3.2. LUPedia Internal Structure

The LUPedia web service is build around the LKB Gazetteer and loaded with DBPedia resources from LDSR.

**3.2.1. LKBGazetteer Embedder class** – this is an implementation of the public `LKBGazEmbedder` interface, which allows for instantiation and utilization of the core gazetteer functionality.

**3.2.2. Configuration utilities** – this functionality allows usage of configuration files for adjustment of the web-service operational regime.

**3.2.3. Dictionary Feeder for Semantic Repositories** – this is an implementation of the public `DictionaryFeeder` interface, which is profiled for SPARQL-endpoint data sources.

**3.2.4. LKBCrawler utility** – this functionality allows “crawling” of publicly accessible SPARQL endpoints with reduced throughput. Usually public SPARQL endpoints have restrictions on the size of the results they can return. Utility could be configured to automatically split the result-set with the dictionary data to fit time-out or number-of-rows restrictions. Usage of this utility allows the LUPedia web service not to depend on availability of dedicated high-throughput SPARQL endpoints.

**3.2.5. Web-Service definition class** – this is a REST web service definition class that exposes the four end-points of the LUPedia service. The four endpoints represent the four different output formats that the web service supports.

**3.2.6. Base Response Generator class** – this class performs the primary text processing using the LKBGazetteer utility. Then it applies four predefined filters over the produced Lookup set in accordance with the provided filter configuration parameters.

**3.2.7. Response generation packages** – there are four packages for each of the response formats supported: HTML, XML, JSON and HTML+RDFa. Every package extends the base response generator class, takes its results and produces specifically formatted output.

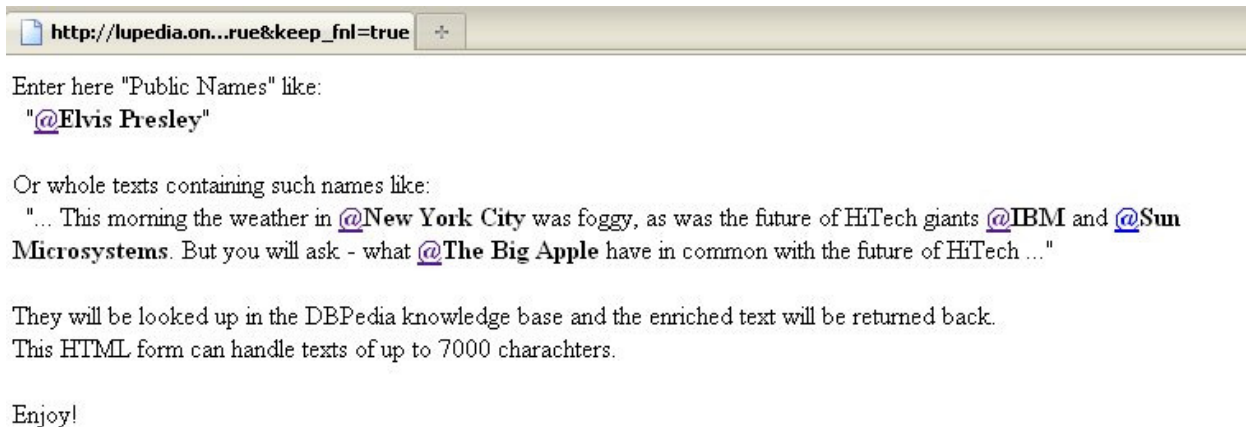
**3.2.8. BBC program data retrieval utility** – this utility uses the functionality of the `LKBCrawler` utility to crawl over a configured SPARQL endpoint and retrieve

the synopsis data of the BBC programs. The retrieved data is stored in local instance of Sesame semantic repository.

**3.2.9. BBC program data processing utility** – this utility processes the BBC program data from the local Sesame repository and stores back the info of the entities found in the text.

### 3.3. Supported Response Formats

**3.3.1. HTML Response** – returns HTML formatted version of the provided text where hyperlinks are added on the places of the found Lookups.



---

#### ACTIVATED FILTERS:

- Skip matches shorter than 3 symbols.
- Skip matches which are stop-words.
- Keep only first and longest match at given point.

#### CLASS: <http://dbpedia.org/ontology/Place>

- [http://dbpedia.org/resource/New\\_York\\_City](http://dbpedia.org/resource/New_York_City) - [New York City, The Big Apple]

#### CLASS: <http://dbpedia.org/ontology/Organisation>

- <http://dbpedia.org/resource/IBM> - [IBM]
- [http://dbpedia.org/resource/Sun\\_Microsystems](http://dbpedia.org/resource/Sun_Microsystems) - [Sun Microsystems]

#### CLASS: <http://dbpedia.org/ontology/Person>

- [http://dbpedia.org/resource/Elvis\\_Presley](http://dbpedia.org/resource/Elvis_Presley) - [Elvis Presley]

Fig 2. Resulting page from LUPedia web user interface

**3.3.2. JSON Response** – returns a JSON formatted list of the found Lookups.

**3.3.3. HTML+RDFa Response** – returns HTML formatted version of the provided text where RDFa enriched hyperlinks are added on the places of the found Lookups.

### 3.3.4. XML Response – returns a XML formatted list of the found Lookups (see Fig 3)

```
- <lookupResult>
  - <lookups>
    <startOffset>36</startOffset>
    <endOffset>49</endOffset>
    <instanceUri>http://dbpedia.org/resource/Elvis_Presley</instanceUri>
    <instanceClass>http://dbpedia.org/ontology/Person</instanceClass>
  </lookups>
  - <lookups>
    <startOffset>133</startOffset>
    <endOffset>146</endOffset>
    <instanceUri>http://dbpedia.org/resource/New_York_City</instanceUri>
    <instanceClass>http://dbpedia.org/ontology/Place</instanceClass>
  </lookups>
  - <lookups>
    <startOffset>193</startOffset>
    <endOffset>196</endOffset>
    <instanceUri>http://dbpedia.org/resource/IBM</instanceUri>
    <instanceClass>http://dbpedia.org/ontology/Organisation</instanceClass>
  </lookups>
  - <lookups>
    <startOffset>201</startOffset>
    <endOffset>217</endOffset>
    <instanceUri>http://dbpedia.org/resource/Sun_Microsystems</instanceUri>
    <instanceClass>http://dbpedia.org/ontology/Organisation</instanceClass>
  </lookups>
  - <lookups>
    <startOffset>243</startOffset>
    <endOffset>256</endOffset>
    <instanceUri>http://dbpedia.org/resource/New_York_City</instanceUri>
    <instanceClass>http://dbpedia.org/ontology/Place</instanceClass>
  </lookups>
</lookupResult>
```

Fig 3. Example of XML response

## 4. BBC use-case handling

### 4.1. BBC use-case reference

The BBC use-case is described as **Scenario 7c** at the wiki of the **NoTube** project. The main objective of this use-case is to follow user's behavior on various social networks and extracting user preferences to recommend content from the BBC programs. The main purpose of applying semantic enrichment services to this use-case is to enrich the original program description. The annotation provides highlights in the textual description and links the corresponding concepts - named entities - to a common knowledge representation that is accessible by the recommendation system. The actual usage of the annotated data is out of the scope of this deliverable and can be found in [5] and [6].

The main source of BBC program data in this experiment is a SPARQL endpoint available here: <http://api.talis.com/stores/bbc-backstage/services/sparql>

### 4.2. Loaded Dictionary Data

Looking for a good source of named entities and following consortium preferences we have chosen DBpedia for the initial experiments on which the LUPedia service can be

used in Wp7c. For processing of the BBC program data, the Dictionary of the LKB Gazetteer was loaded with data from [DBpedia](#). An attempt was made to download the DBpedia data from their public SPARQL endpoint but within a week period – only half of the needed data was retrieved. Finally the DBpedia data was crawled from the public SPARQL endpoint of [LDSR](#), which took 40 minutes.

The crawl retrieved information of over 1.3 million Named Entity aliases of the following DBpedia base classes: "Work", "Place", "Organisation", "Person" and "Event". The aliases are stored and are searched in texts in case-sensitive manner.

### 4.3. BBC programs text data

The BBC program text data that was crawled from the Talis public SPARQL endpoint consists of the Synopsis texts of the program descriptions. The crawled data was stored in a local semantic repository.

Due to existence of lot of corrupted entries in the Talis semantic repository – only part of the data was downloaded, however the rdf description is available also on BBC side ( see Fig 4)

```
-<rdf:RDF>
- <rdf:Description rdf:about="/programmes/b0094j2f.rdf">
  <rdfs:label>Description of the series Series 1</rdfs:label>
  <dcterms:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2008-02-24T03:19:56Z</dcterms:crea
  <dcterms:modified rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">2008-11-06T16:28:45Z</dcterms:m
  <foaf:primaryTopic rdf:resource="/programmes/b0094j2f#programme"/>
</rdf:Description>
- <po:Series rdf:about="/programmes/b0094j2f#programme">
  <dc:title>Series 1</dc:title>
  - <po:short_synopsis>
    Drama series set in the world of advertising in 1960s New York.
  </po:short_synopsis>
  - <po:medium_synopsis>
    Drama series which takes an unflinching look at the world of advertising in 1960s New York.
  </po:medium_synopsis>
  <po:genre rdf:resource="/programmes/genres/drama#genre"/>
  <po:masterbrand rdf:resource="/bbcfour#service"/>
  - <po:episode>
    - <po:Episode rdf:about="/programmes/b009364t#programme">
      <dc:title>Smoke Gets in Your Eyes</dc:title>
      <po:position rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</po:position>
    </po:Episode>
  </po:episode>
  - <po:episode>
    - <po:Episode rdf:about="/programmes/b0094yyp#programme">
```

Fig 4. BBC programs rdf description

### 4.4. Processed BBC Program text data

The BBC program text data was processed using the LUPedia web service and the results were stored back in the local Sesame repository. The original content as seen by the BBC web site visitors (see Fig 5) has been annotated in background and then made available for the other NoTube service.



Fig 5. Original BBC web-site content

The results can be reviewed through the Sesame query interface available at:  
<http://sparql-notube.ontotext.com/openrdf-workbench/repositories/notube-test/query>

Current Selections:  
 Sesame server: <http://192.168.130.72:8080/openrdf-sesame> [\[change\]](#)  
 Repository: OWLIM B5BM Repository ( notube-test ) [\[change\]](#)

## Explore (<<http://www.bbc.co.uk/programmes/b0094j2>>

Drama series set in the world of advertising in 1960s New York.

Subject	Predicate	Object	Context
< <a href="http://www.bbc.co.uk/programmes/b0094j2#programme">http://www.bbc.co.uk/programmes/b0094j2#programme</a> >	rdf:type	< <a href="http://purl.org/ontology/po/Series">http://purl.org/ontology/po/Series</a> >	< <a href="http://www.ontotext.com/explicit">http://www.ontotext.com/explicit</a> >
< <a href="http://www.bbc.co.uk/programmes/b0094j2#programme">http://www.bbc.co.uk/programmes/b0094j2#programme</a> >	rdfs:label	"Drama series set in the world of advertising in 1960s New York."@en	< <a href="http://www.ontotext.com/explicit">http://www.ontotext.com/explicit</a> >
< <a href="http://www.bbc.co.uk/programmes/b0094j2#programme">http://www.bbc.co.uk/programmes/b0094j2#programme</a> >	< <a href="http://www.w3.org/2004/02/skos/core#note">http://www.w3.org/2004/02/skos/core#note</a> >	"<HTML><HEAD></HEAD><BODY>Drama series set in the world of advertising in 1960s <b><a href="http://dbpedia.org/resource/New_York" title="New York">New York</a></b></HTML>"@en	< <a href="http://www.ontotext.com/explicit">http://www.ontotext.com/explicit</a> >
< <a href="http://www.bbc.co.uk/programmes/b0094j2#programme">http://www.bbc.co.uk/programmes/b0094j2#programme</a> >	< <a href="http://www.w3.org/2004/02/skos/core#related">http://www.w3.org/2004/02/skos/core#related</a> >	< <a href="http://dbpedia.org/resource/New_York">http://dbpedia.org/resource/New_York</a> >	< <a href="http://www.ontotext.com/explicit">http://www.ontotext.com/explicit</a> >

Resource: <<http://www.bbc.co.uk/programmes/b0094j2#programme>>

Fig 6. The User Interface for access to the processed BBC data

Also the results could be accessed through a SPARQL endpoint at:  
<http://sparql-notube.ontotext.com/openrdf-workbench/repositories/notube-test>

## 5. Conclusion

This Deliverable presented the LUPedia text analysis and enrichment web service. It detailed the grounds and the structure of its state of art main component – the LKB Gazetteer. Then it showed how its structure contributes for its automation and autonomy and how its functionality is made available for usage. The deliverable is wrapped up with the display of how the LUPedia web service was applied successfully on the BBC corpus of use-case data.

## 6. References

- [1] Arousseau, M. "On Lists of Words and Lists of Names," *The Geographical Journal* (Volume 105, Number 1/2, 1945): 61–67.
- [2] [http://en.wikipedia.org/wiki/Hash\\_function](http://en.wikipedia.org/wiki/Hash_function)
- [3] Linking Open Data W3C SWEO Community Project.  
<http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData/>,  
(2007).
- [4] D4.1 Requirement Analysis for Semantic Annotation of TV Content v.2,
- [5] D3.1. User and Context model Specification
- [6] D7c.1 Social TV: Use Case Specs, Design and first mock-up