xLiMe
FP7-ICT-2013-10
ICT-2013.4.1
Contract no.: 611346
www.xlime.eu

Deliverable D1.2

Prototype of Data Processing Infrastructure

Editor:	Ronald Denaux, ISOCO
Author(s):	Ronald Denaux, ISOCO; Nuria García, ISOCO; Andreas Thalhammer, KIT; Aditya Mogadala, KIT
Deliverable Nature:	Prototype (P)
Dissemination Level:	Public (PU)
Contractual Delivery Date:	M12 – 31 October 2014
Actual Delivery Date:	M12 – 31 October 2014
Suggested Readers:	Architects and developers involved in data publishing and consumption
Version:	1.0
Keywords:	Real-time, data processing, infrastructure, deployment

Disclaimer

This document contains material, which is the copyright of certain xLiMe consortium parties, and may not be reproduced or copied without permission.

In case of Public (PU):
All xLiMe consortium parties have agreed to full publication of this document.

In case of Restricted to Programme (PP):
All xLiMe consortium parties have agreed to make this document available on request to other framework programme participants.

In case of Restricted to Group (RE):
The information contained in this document is the proprietary confidential information of the xLiMe consortium and may not be disclosed except in accordance with the consortium agreement. However, all xLiMe consortium parties have agreed to make this document available to <group> / <purpose>.

In case of Consortium confidential (CO):
The information contained in this document is the proprietary confidential information of the xLiMe consortium and may not be disclosed except in accordance with the consortium agreement.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the xLiMe consortium as a whole, nor a certain party of the xLiMe consortium warrant that the information contained in this document is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using this information.

Full Project Title:	xLiMe – crossLingual crossMedia knowledge extraction
Short Project Title:	xLiMe
Number and Title of Work Package:	WP1 Processing Multilingual Multimedia Data
Document Title:	D1.2 - Prototype of Data Processing Infrastructure
Editor:	Ronald Denaux, ISOCO
Work Package Leader:	Andreas Thalhammer, KIT

Copyright notice

© 2013-2016 Participants in project xLiMe

Executive Summary

This document covers the prototype of the xLiMe data processing infrastructure. This prototype of the data processing infrastructure will be evaluated and extended throughout the project. The main goal is to create a robust infrastructure that can handle the volume and velocity of annotated multimedia and multilingual data produced by the various data providers and annotators in the xLiMe project. The data processing infrastructure is responsible for providing a stable solution for data collection, storing and indexing, thus relieving the various data providers and annotation components from the burden of implementing similar solutions. This document describes the implementation of the data processing infrastructure and gives examples of how the infrastructure can be used by data providers and annotation providers.

Table of Contents

Executive Summary 3
Table of Contents 4
Abbreviations 5
1 Introduction 6
1.1 Data Processing Infrastructure in xLiMe 6
2 Apache Kafka for Data Collection 8
3 CumulusRDF and MongoDB for Data Storage and Search 10
3.1 CumulusRDF 10
3.2 MongoDB 10
3.2.1 Indexing 11
3.2.2 Configuration 11
4 Example Uses 12
4.1 Data Provider Pushing Metadata to Kafka 12
4.2 Annotation Provider Listening to Kafka Topic and Pushing Annotations to New Topic 13
4.3 Data Consumer Querying Cumulus RDF 14
4.4 Data Consumer Querying MongoDB 16
5 Conclusion 17
References 18

Abbreviations

RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
URI	Unified Resource Identifier
URL	Unified Resource Locator

1 Introduction

In this document we will introduce the prototype of the xLiMe data processing infrastructure. This prototype will be evaluated and extended throughout the project. The main goal is to create a shared infrastructure which can be used by data providers, data annotators and client applications (e.g. user interfaces) to collect, store and search multimedia and multilingual (meta-) data and its annotations. The shared infrastructure facilitates the definition of clear interfaces for the various xLiMe stakeholders and relieves them from creating custom solutions for collecting, storing and searching the data.

In the remainder of the introduction, we will review and summarise the specifications identified for the data processing infrastructure as identified in D6.1, which described the overall architecture of the xLiMe toolkit. This will position the data processing infrastructure in the context of the xLiMe project.

The main part of this document describes the main components of the data processing infrastructure: a message broker for collecting data and two storage and indexing mechanisms: one is a graph database based on RDF and the other one is a NoSQL document store. Finally, we give examples of how the various xLiMe stakeholders can use the infrastructure to contribute to the xLiMe toolkit.

1.1 Data Processing Infrastructure in xLiMe

The xLiMe Toolkit Architecture [3] identifies three main components for the data processing infrastructure:

1. A **message broker** to resolve the issue of data collection from the various xLiMe stakeholders, i.e. the various data and annotation service providers.
2. A **distributed database** for storing and indexing of the multimedia and multilingual data produced by the xLiMe stakeholders. We assume that these databases provide enough search capabilities for retrieving relevant information (or that more advanced search capabilities can be built on top of the basic search capabilities).
3. A **streaming interface** for subscribing to a subset of the multimedia data or annotations.

The main purpose of the data processing infrastructure is to provide an extensible integration platform for the xLiMe stakeholders:

- Data providers such as VICO, JSI newsfeed, ZATOO and ECONDA, can focus on converting their data to conform to the xLiMe data model [2] and can then connect to the message broker to publish their data as it becomes available.
- Data annotators can connect to the message broker to listen to incoming multimedia (meta-) data in order to start or schedule annotation tasks. As annotations are computed, they can be sent back to the message broker to make them available to the rest of the xLiMe partners
- Data consumers, such as the use-case owners, can retrieve data from the xLiMe platform in various ways: by listening to the message broker directly, by subscribing to the streaming interface or by querying the distributed database.

This document describes the implementation of the data processing infrastructure at the end of the 1st year of the project. Based on the requirements for the Y1 prototypes, it was decided that only the message broker and the distributed database would be implemented this year, while leaving the streaming interface as an extension for the 2nd year.

The implemented components in the data-processing infrastructure, and their relations to the remaining components in the xLiMe architecture, are depicted in Figure 1. In the following sections, we describe the implemented and deployed components in more detail.

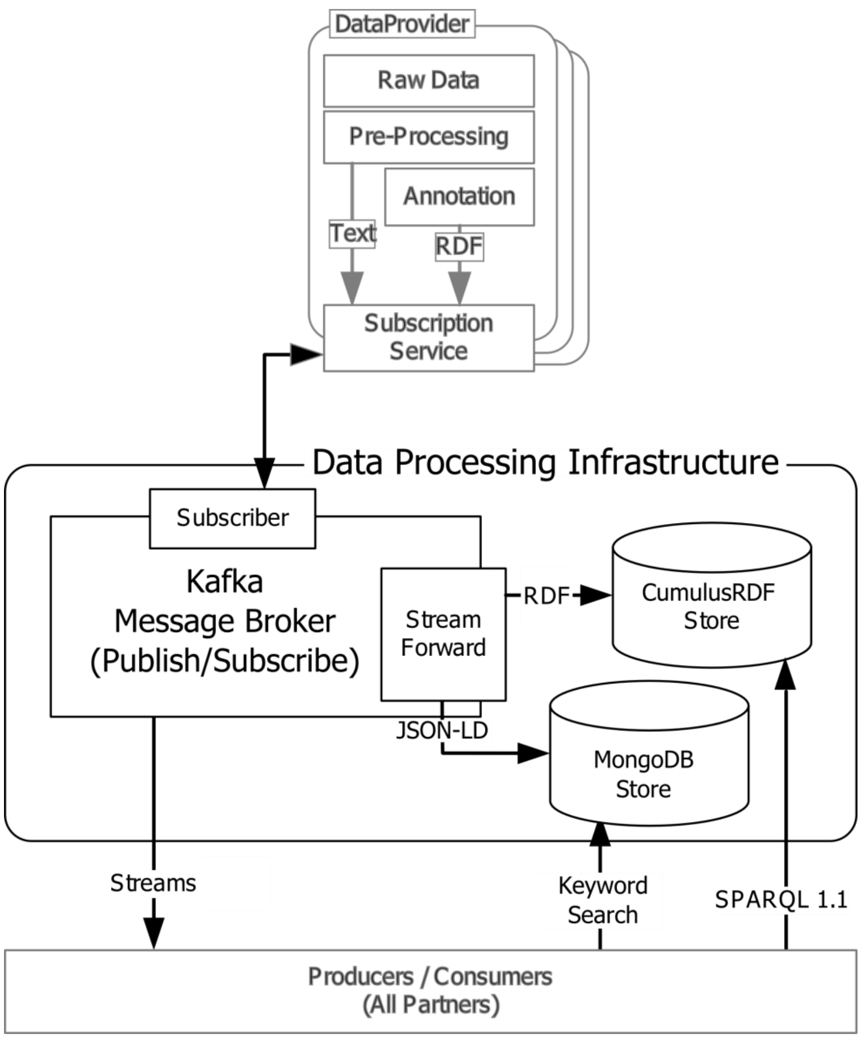

Figure 1: Deployment diagram for the xLiMe architecture, with the implemented data-processing infrastructure and its relation to the remaining xLiMe components.

2 Apache Kafka for Data Collection

As described in xLiMe deliverable “D6.1 – Toolkit Architecture Implementation”, there are a large number of enterprise quality message brokers available[1]. However, not all of them are suitable for the xLiMe Toolkit requirements. After careful evaluation of the various message brokers, we decided to use Apache Kafka as the message broker for the xLiMe data processing infrastructure. The main reasons for this choice were:

- Active development
- Cursor feature, which allows message consumers to catch up with the rest of the system. This allows message consumers to decide how and when to consume messages, rather than requiring consumers to be continuously connected to the message broker. Also, this feature means that if a consumer crashes, it can be restarted and it will start consuming messages from where it stopped, i.e. no or few messages will be lost.
- Easy installation and management
- It can scale horizontally to provide improved resilience
- Generous documentation.

As described in [3], Apache Kafka is a distributed publish/subscribe messaging system. It’s fast (a single broker can manage hundreds of megabytes of reads and writes per second from thousands of clients), scalable, durable, distributed by design, partitioned, and comes with a replicated commit log. It supports multiple subscribers and automatically balances the consumers if there are failures. It saves messages on disk, so it can be used in real time applications and in batch mode. Kafka has messages marked by the offset in the log versus the ids in traditional broker systems. This improves the performance, avoiding overhead. Other benefits are that Kafka doesn’t wait for acknowledges and sends messages as fast as the broker can handle, so this implies better performance, and Kafka has a more efficient storage format.

Apache Kafka is currently deployed on a single node with 4 x Intel(R) Xeon(TM) 3.00GHz CPUs, 4 GB of RAM, 1 Gbit Ethernet interface, and 1,2 TB available data space (next to the system components). The deployment retains the log for 168 hours which is equal to 7 days. The default number of partitions is set to 2. Currently, there are 15 topics deployed. As a matter of fact, the number of messages differs from topic to topic. In the remainder following we provide a snapshot of the current disk use of Apache Kafka in the xLiMe project:

As of 22 October 2014, the current total disk use (per week) is 5.5G which is split as follows[2] (note the two log partitions per topic):

2,9M	jsi-annotations-0
2,0G	jsi-newsfeed-0
2,0G	jsi-newsfeed-1
89M	KITRecommendations-0
4,0K	recovery-point-offset-checkpoint
4,0K	replication-offset-checkpoint
597M	socialmedia-0
589M	socialmedia-1
900K	tv-metadata-0

[1] http://queues.io/

[2] Empty topics removed.

```
0       tv-metadata-1
68K     tv-metadata-2
1,9M    tv-metadata-3
45M     zattoo-asr-0
45M     zattoo-asr-1
8,4M    zattoo-epg-0
8,4M    zattoo-epg-1
```

3 CumulusRDF and MongoDB for Data Storage and Search

The xLiMe Toolkit Architecture specifies that the data uploaded to the message broker should be stored in a distributed database. This database solves the following problems:

- It makes the data persistent. The information items will be available for longer than they are on the message broker (retention time).
- It aggregates the data. Message brokers typically have topics where similar data are kept together; hence clients need to know the location of specific types of data in order to access them. The database contains the various types of data, and provides a uniform manner of accessing that data.
- It provides indexing, query, and search mechanisms for easier access to the data.

3.1 CumulusRDF

Since the xLiMe data model is defined in terms of various RDF vocabularies (which make integration, interlinking and web-access easier), [3] identified various candidate RDF triple stores which can be used to implement the xLiMe database. For the 1st-year prototype of the data processing infrastructure, we have chosen CumulusRDF as the triple store. The main reasons for this choice were:

- Horizontal scalability due to the underlying technology (i.e. Apache Cassandra Key-value distributed NoSQL data store)
- In-house knowledge and development of the triple store. While some other triple stores with similar characteristics are available (e.g. BigData and Virtuoso Open Source Edition), installation and management of these other triple stores is cumbersome and lack of timely support may be an issue. CumulusRDF has been developed by one of the xLiMe partners (KIT), thus installation and management of this triple store is readily available. Furthermore, in case that new features are necessary as part of the xLiMe project, knowledge of how to implement these features is available.

CumulusRDF is an RDF store on a cloud-based architecture. It provides a REST-based API with CRUD (creation, read, update and delete) operations to manage RDF data. The current version uses Apache Cassandra (an open source nested key-value store) as storage backend. Its development is mainly driven by the Institute of Applied Informatics and Formal Desciption Methods (AIFB) but also involves further open source developers that are contributing actively.

CumulusRDF offers a highly scalable RDF store for write-intensive applications. It allows for fast and lightweight evaluation of triple pattern queries, has full support for triple and quad storage. Also, it comprises a Sesame SAIL interface implementation, contains a SPARQL 1.1 Endpoint and facilitates a web-bases service to manage the platform.

CumulusRDF is currently deployed on a single node with 8 x 2.0 GHz AMD Opteron "Warsaw" cores, 40GB of RAM, and 3TB of disk space. With more intensive usage, the head node is planned to be supported by three further replication nodes with 4 x 2.0 GHz CPUs, 8 GB of RAM and 1TB of disk space each.

The head node of the cumulus deployment is also hosting the Kafka2Cumulus data services. These are lightweight Tomcat Web services that are built to consume a specific Kafka topic each (on init). As the data formats on the Kafka topics are quads in valid TriG[3] format, the Kafka2Cumulus services consists of a topics consumer that pulls messages from Kafka and writes them to the CumulusRDF triple store.

3.2 MongoDB

Since some features in CumulusRDF are missing (i.e. full-text search), for the 1st year data processing infrastructure prototype, we have also implemented a secondary database and index based on MongoDB.

[3] http://wifo5-03.informatik.uni-mannheim.de/bizer/trig/ (access date: 22.10.2014)

MongoDB is a NoSQL database which supports document storage. It is schema-less and allows JSON documents having key and values. The advantage of MongoDB is that it is document-oriented.

It also supports ad-hoc queries and can be used for indexing. Since it can be replicated, it provides high availability. The horizontal scaling of MongoDB is also possible using sharding. For accessing the data stored in the MongoDB, we can use many front-end programming languages.

3.2.1 Indexing

The documents stored in MongoDB need to be index for search and other functionality. For content based similarity, we index the fields containing only text. To achieve it, the Text Indexing feature of MongoDB is used.

For the indexing we have specified which data fields for each document need to be taken into account, as each data item generated from different modalities contains different text fields. For example, the social media data (originating from VICO) is indexed based on the micropost's text field. For another example, the tv-metadata (obtained from Zattoo) is indexed using titles of the show and their descriptions. To eliminate the data which is redundant we remove the duplicates obtained in the stream.

3.2.2 Configuration

For the 1st year data processing infrastructure, MongoDB 2.6.4 is used.

The standard MongodB distribution provides utilities for running and managing the database: mongod is used to run the server on a Linux machine, while mongo is used for administration.

In order to deploy the MongoDB, first a database was created to store different streams coming from VICO, ZATOO and JSI. Then, three different collections inside the database were created: one for each of the data providers: VICO, ZATTOO and JSI. Next, a data loader was created, which subscribes to all of the topics in Kafka. Finally, each collection is indexed based on text fields for content based similarity as described above.

4 Example Uses

In this section, we give various examples to demonstrate how xLiMe stakeholders can use the data processing infrastructure described above.

4.1 Data Provider Pushing Metadata to Kafka

Data processing functionality needs that providers send their messages to the pipeline in order to parse and store them. These messages have to be pushed into the xLiMe messaging broker, Apache Kafka. To achieve this, we can use the command line or develop different Kafka client applications.

First of all, to push data into Kafka, we have to create a topic if it doesn't exist. To do this task it is necessary specifying the next input parameters (Table 1).

Input parameter	Description	Example value
Host	URL:Port, Address where Kafka is deployed to push the data	localhost:2181
Replication Factor	Controls how many servers will replicate each message that is written	1
Partitions	Controls how many logs the topic will be sharded	1
Topic	Name of the topic we're going to create	test

Table 1: Input parameters in Creation Topic task

An example to run this task in command line for xLiMe Apache Kafka broker could be:

```
bin/kafka-topics.sh --create --zookeeper kafka-server.example.com:2181 --
replication-factor 1 --partitions 4 --topic tv-metadata
```

Another useful command is listing the topics available in the broker:

```
bin/kafka-topics.sh --list --zookeeper kafka-server.example.com:2181
```

Now the output in our Kafka broker is as follows.

```
jsi-annotations
jsi-newsfeed
kitrec
social-media
socialmedia
test
topic
tv-asf
tv-asr
tv-metadata
tv-ocr
tv-visual
vico-socialmedia
zattoo-asr
zattoo-epg
```

Next, providers can send messages to Kafka. As we mentioned, it is possible using the command line or a client application. Both cases need several input parameters configuration (Table 2).

Input parameter	Description	Example value
Host	URL:Port, Address where Kafka is deployed to push the data	localhost:9092
Topic	Name of the topic we're going to send data	test

Table 2: Input parameters to send messages to Kafka

An example of pushing data into xLiMe broker:

```
bin/kafka-console-producer.sh --broker-list kafka-server.example.com:9092 --
topic tv-metadata
```

This command can take the message sent from a file or from standard input. By default each line will be sent as a separate message.
Besides, Apache Kafka can be used from a client application in different programming languages and platforms. Kafka wiki website provides several examples to configure this issue. A producer example in Java can be found in the Web (https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+Producer+Example)

4.2 Annotation Provider Listening to Kafka Topic and Pushing Annotations to New Topic

Consuming messages is another important task in xLiMe infrastructure. Providers can get data from Apache Kafka pipeline. This issue can be run in command line or client application as producer approach. The command line consumer will dump messages to standard output.

Several relevant input parameters can be specified (Table 3).

Input parameter	Description	Example value
Host	URL:Port, Address where Kafka is deployed to push the data	localhost:2181
Topic	Name of the topic we're going to consume data	test
Group Id	Id of the group who is going to consume data	groupid
--from-beginning	*(Only in command line)* Specifying user wants consume data from first message	
zk.sessiontimeout.ms	The zookeeper session timeout.	6000
zk.synctime.ms	Max time for how far a zookeeper follower can be behind a zookeeper leader	2000
autocommit.interval.ms	Frequency that the consumed offsets are committed to zookeeper	10000
autocommit.enable	If set to true, the consumer periodically commits to zookeeper the latest consumed	true

	offset of each partition.	
autooffset.reset	• smallest: automatically reset the offset to the smallest offset available on the broker. • largest: automatically reset the offset to the largest offset available on the broker. • anything else: throw an exception to the consumer.	smallest

Table 3: Input parameters to consume messages from Kafka

A command line example to consume data from an xLiMe broker is:

```
bin/kafka-console-consumer.sh --zookeeper kafka-server.example.com:2181 --topic
tv-metadata --from-beginning
```

A Java example to consume messages of Kafka is shown in the web. Below we write a code fragment showing a basic consumer configuration in Java to xLiMe specification. (https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example)

```
Properties props = new Properties();
 props.put("zookeeper.connect", " kafka-server.example.com:2181");
 props.put("group.id", "ISOCOTopicCaching-test5");
 props.put("zookeeper.session.timeout.ms", "4000");
 props.put("zookeeper.sync.time.ms", "2000");
 props.put("auto.commit.interval.ms", "1000");
 props.put("auto.commit.enable", "false");
 props.put("auto.offset.reset", "smallest");
```

Once providers have consumed the messages they can parse data, annotate it and upload it to Kafka using a new topic. To do this they have to follow the instructions listed in previous section 4.1.

4.3 Data Consumer Querying Cumulus RDF

As an RDF repository, CumulusRDF uses the SPARQL 1.1 [1] standard for queries. Below, we show several examples to get information from xLiMe Dataset. These examples answer competency questions specified in deliverable D1.1 [2].

First example returns data that satisfy the question *"What is the title of the show currently aired on SRF 1?"*.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ma: <http://www.w3.org/ns/ma-ont#>
PREFIX xlime: <http://xlime-project.org/vocab/>

SELECT ?title
FROM < http://xlime-project.org/data>
WHERE {
   ?uri a ma:MediaResource .
   ?uri ma:hasPublisher [ rdfs:label "SRF 1" ] .
   ?uri ma:date ?date.
   ?uri ma:title ?title.
   ?uri ma:duration ?duration.
   bind(now() as ?nowdate) .
   bind((?nowdate - ?duration) as ?compdate)
   FILTER (?date > ?compdate && ?date < ?nowdate).
}
```

"In which currently airing shows was the entity dbpedia:New_York_City mentioned in speech? "

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ma: <http://www.w3.org/ns/ma-ont#>
PREFIX xlime: http://xlime-project.org/vocab/>
PREFIX dbpedia: http://dbpedia.org/resource/>

SELECT ?uri
FROM < http://xlime-project.org/data>
WHERE {
   ?uri a ma:MediaResource .
   ?uri ma:hasTrack ?uri2 .
   ?uri2 a ma:AudioTrack .
   ?uri2 xlime:hasAnnotation [ xlime:hasEntity dbpedia:New_York_City ] .
   ?uri ma:date ?date.
   ?uri ma:duration ?duration.
   bind(now() as ?nowdate) .
   bind((?nowdate - ?duration) as ?compdate)
   FILTER (?date > ?compdate && ?date < ?nowdate).
}
```

Above examples get data from Zatoo provider. The next ones show a query looking information from NewsFeed or VICO providers. The query answers to the question *"At which position was Deichmann mentioned in a news article?".*

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ma: <http://www.w3.org/ns/ma-ont#>
PREFIX xlime: http://xlime-project.org/vocab/>
PREFIX kdo: <http://kdo.render-project.eu/kdo#>
PREFIX dbpedia: http://dbpedia.org/resource/>
PREFIX sioc: <http://rdfs.org/sioc/ns#>

SELECT *
FROM < http://xlime-project.org/data>
WHERE {
   ?uri a kdo:NewsArticle . (VICO -> ?uri a sioc:MicroPost . )
   ?uri xlime:hasAnnotation [ xlime:hasEntity dbpedia:Deichmann ;
                              xlime:hasPosition
                              [ xlime:hasStartPosition ?start ;
                                Xlime:hasStopPosition ?end ] ] .

}
```

"Which entities do news articles in different languages of the last 14 days cover?"

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ma: <http://www.w3.org/ns/ma-ont#>
PREFIX xlime: http://xlime-project.org/vocab/>
PREFIX kdo: <http://kdo.render-project.eu/kdo#>
PREFIX dbpedia: http://dbpedia.org/resource/>
PREFIX sioc: <http://rdfs.org/sioc/ns#>
PREFIX dcterms: <http://purl.org/dc/terms/>

SELECT ?lang ?entity
FROM < http://xlime-project.org/data>
WHERE {
   ?uri a kdo:NewsArticle .
```

```
    ?uri dcterms:language ?lang .
    ?uri xlime:hasAnnotation [ xlime:hasEntity ?entity ] .
    ?uri dcterms:created ?date .
    FILTER (?date > bind(now() - (14 * 86400)).
} GROUP BY ?lang
```

4.4 Data Consumer Querying MongoDB

Similar queries in MongoDB from previous examples in SPARQL (4.3) are the following.

"What is the title of the show currently aired on SRF 1? "

```
db.xlime.find( { type: "ma:MediaResource" ,
'ma:hasPublisher.rdfs:label': 'SRF 1'  ,
ma:date:{ $gt: { $subtract: [ new Date().getTime(), ma:duration ]}},
ma:date:{ $lt: new Date().getTime()}}, { ma:title: 1, _id: 0} )
```

"In which currently airing shows was the entity dbpedia:New_York_City mentioned in speech? "

```
db.xlime.find( { type: "ma:MediaResource" ,
ma:hasTrack: { type: "ma:AudioTrack",
'xlime:hasAnnotation.xlime:hasEntity': 'dbpedia:New_York_City' },
ma:date:{ $gt: { $subtract: [ new Date().getTime(), ma:duration ]}},
ma:date:{ $lt: new Date().getTime()}})
```

"At which position was Deichmann mentioned in a news article?".

```
db.xlime.find( { type: "kdo:NewsArticle",
'xlime:hasAnnotation.xlime:hasEntity': 'dbpedia:Deichmann' },
{ xlime:hasAnnotation: { xlime:hasPosition: {
xlime:hasStartPosition: 1, xlime:hasStopPosition: 1}}})
```

"Which entities do news articles in different languages of the last 14 days cover?"

```
db.xlime.aggregate(
{ $match: { type: "kdo:NewsArticle",
dcterms:created:{$gt:{$subtract:[new Date().getTime(),
$multiply:[ 14, 86400] ]}}}},
{ $group: { _id: 'dcterms:language',
'xlime:hasAnnotation.xlime:hasEntity': 1}})
```

5 Conclusion

In this deliverable we introduced the prototype of the xLiMe data processing infrastructure. We reviewed the xLiMe Toolkit architecture to place the data processing infrastructure in context. Afterwards we described the main components of the data-processing infrastructure, focusing on the reused components, the motivation for the chosen components, and the description of their configuration and deployment for the 1st year prototype. Finally, we showed examples of how the various xLiMe stakeholders can use the data processing infrastructure.

The data processing infrastructure described in this document serves as the basis for the various prototypes delivered at the end of the 1st year by the xLiMe project [4][5][6]. Detailed benchmark results of the data processing infrastructure (and the rest of the xLiMe toolkit) are currently being gathered and will be reported in D7.1.1.

References

[1] http://www.w3.org/TR/rdf-sparql-query/

[2] xLiMe deliverable “D1.1 – Prototype Meta Data Model”

[3] xLiMe deliverable “D6.1 – Toolkit Architecture Implementation”

[4] xLiMe deliverable “D6.2.1 – Early Prototype”

[5] xLiMe deliverable “D7.2.1 – Early Prototype and Validation Report SEARCH”

[6] xLiMe deliverable “D7.3.1 – Early Prototype and Validation Report MONITOR”