SEVENTH FRAMEWORK PROGRAMME

Networked Media

*Specific Targeted Research Project*

# SMART

(FP7-287583)

# Search engine for MultimediA environment generated contenT

## D4.2 Social Networks and Sensor Networks Integration

Due date of deliverable: 01-05-2013

Actual submission date: 05-06-2013

Start date of project: 01-11-2011 Duration: 36 months

## Summary of the document

| | |
|---|---|
| **Code:** | **D4.2 Social Networks and Sensor Networks Integration** |
| **Last modification:**<br>**State:** | 31/05/2013<br>Review comments being addressed |
| **Participant Partner(s):**<br>**Author(s):**<br>**Fragment:** | AIT, Imperial, Telesto, S3Log<br>A. Babaee, R. Riveret, A. Pnevmatikakis, Th. Alexiou, M. Tarquini<br>No |
| **Audience:** | [✓ public]<br>☐ restricted<br>☐ internal |
| **Abstract:** | This deliverable provides the techniques for data access and representation of social networks information within the SMART search engine and how this data is combined with the sensor network data to infer high level events. A description of the components involved is presented and where relevant examples are given of use cases. |
| **Keywords:** | • Edge node<br>• Social networks<br>• Metadata<br>• Fusion<br>• APIs |
| **References:** | See end of document |

# Table of Contents

# 1 Executive Summary

## 1.1 Scope

The SMART system offers users information from both physical sensors and social networks. Accordingly, integrating and combining the data gathered from social networks and the physical world is a major component of SMART. Here, we present how the data from social networks of interest such as Twitter and Facebook can be collected, managed and used for reasoning on scenarios such as events (e.g. a celebration).

## 1.2 Audience

The target audience for this deliverable is anyone who is interested in the mechanisms by which SMART is collecting and managing social network data and using it later to combine/integrate with physical world data for reasoning. This document is of interest for both SMART project members and the open source community, especially with regards to social network components that can be later added to the project.

## 1.3 Summary

The integration of social networks and sensor networks is the main topic of this document. To achieve this, different SMART components, namely the Social Network Manager (SNM) and a reasoning engine are used. These components are described in the following chapters.

In particular we will describe how using SNM will allow users and other components of SMART to have uniform access to social network data.

We will also elaborate on how the reasoning is performed to find higher level inferences by combining the social network and sensor data and the theory behind the reasoning engine (i.e. Markov Logic Networks).

## 1.4 Structure

The document is structured as follows:

- Chapter 2 provides an introduction to the document and explains the role of social networks as sources of SMART metadata.
- Chapter 3 identifies the social networks of interest to SMART and details the components of the Social Network Manager (SNM).
- Chapter 4 describes the metadata provided by the SNM as feeds to the edge node database, but also demonstrates that these feeds indeed correspond to RDF triples..
- Chapter 5 discusses the rationale behind fusing together social and physical metadata for particular live news and security use cases. It then proceeds to describe the fusion happening within the reasoning engine.
- Chapter 6 concludes the deliverable.

## 2 Introduction

An edge node is a SMART component, where data from local sensors, linked data and other information from social networks are collected and fused into high-level information or events [3]. An edge node is meant to cover a particular location such as a neighbourhood or a part of a city centre.

The overall architecture of an edge node has already been given in the deliverable D4.1, "SMART Distributed Knowledge Base and Open Linked Data Mechanisms" [3] and in the Web site of SMART, http://www.smartfp7.eu/. As a quick recap and an account of what is present in the first software release (see D6.2 [6]), the block diagram of the edge node and its three layers is given in Figure 1.

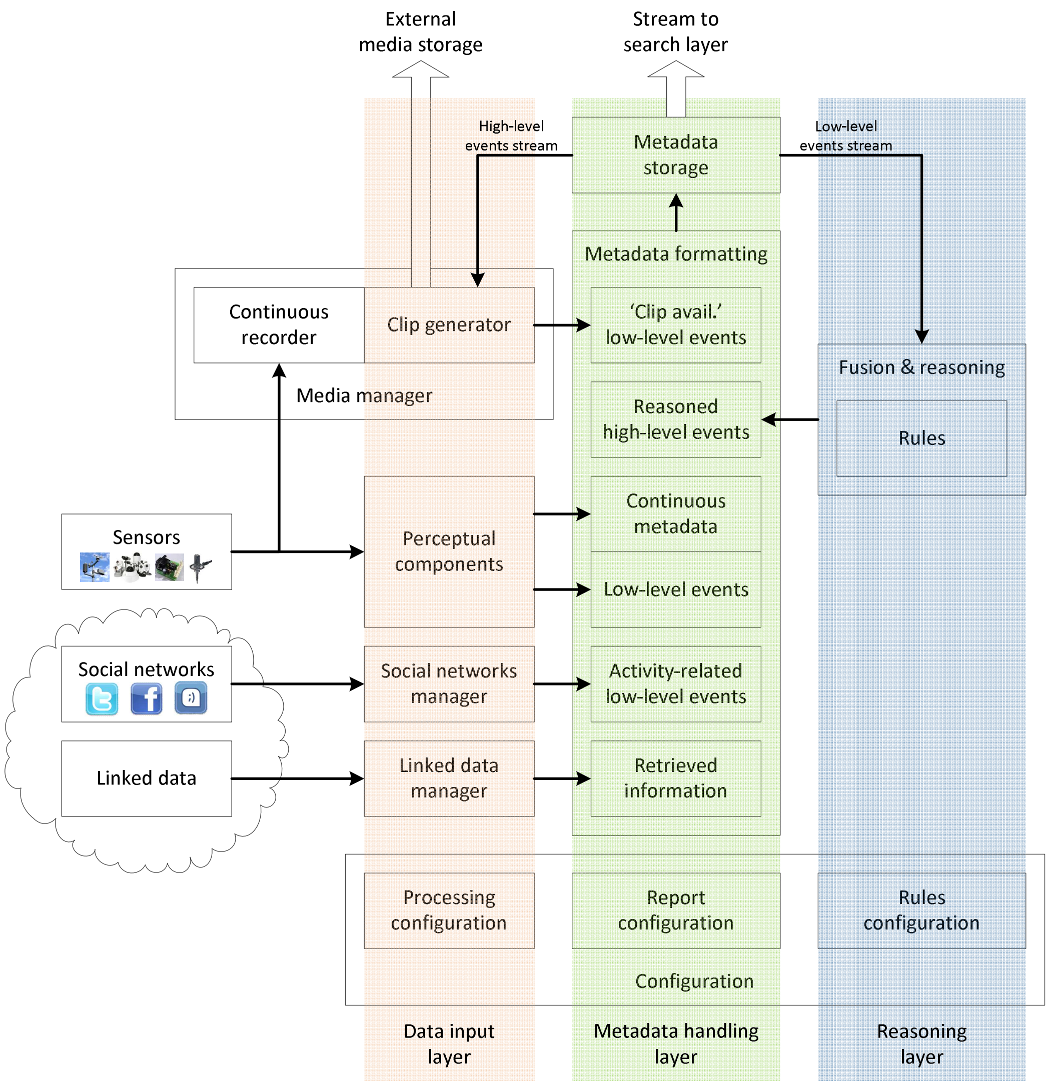


**Figure 1: Edge node architecture.**

The **data input layer** is meant to provide the mechanisms to abstract sensors, social networks and the linked data cloud as sources of data. Its components process the signals from the physical sources (cameras, microphones, weather stations, chemical sensors, etc.) or virtual ones (linked data cloud or social networks) in order to extract metadata.

The **metadata handling layer** handles the metadata streams from the data input layer and makes them available both to the reasoning layer and to the SMART search engine. This is achieved by storing every feed in a CouchDB database.

The **reasoning layer** encapsulates the Intelligent Fusion Manager (IFM). The IFM has two main goals: (i) infer high-level information from the collected data using some rule-based patterns and (ii) learn those patterns.

After a brief introduction of the role of social networks as information sources in the SMART edge node, this deliverable focusses on the way social network data are generated and handled, but also on how they are fused together with physical data by the reasoning layer.

## 2.1 Social networks within SMART

The way social network information is accessed by the various SMART entities is shown in Figure 2.

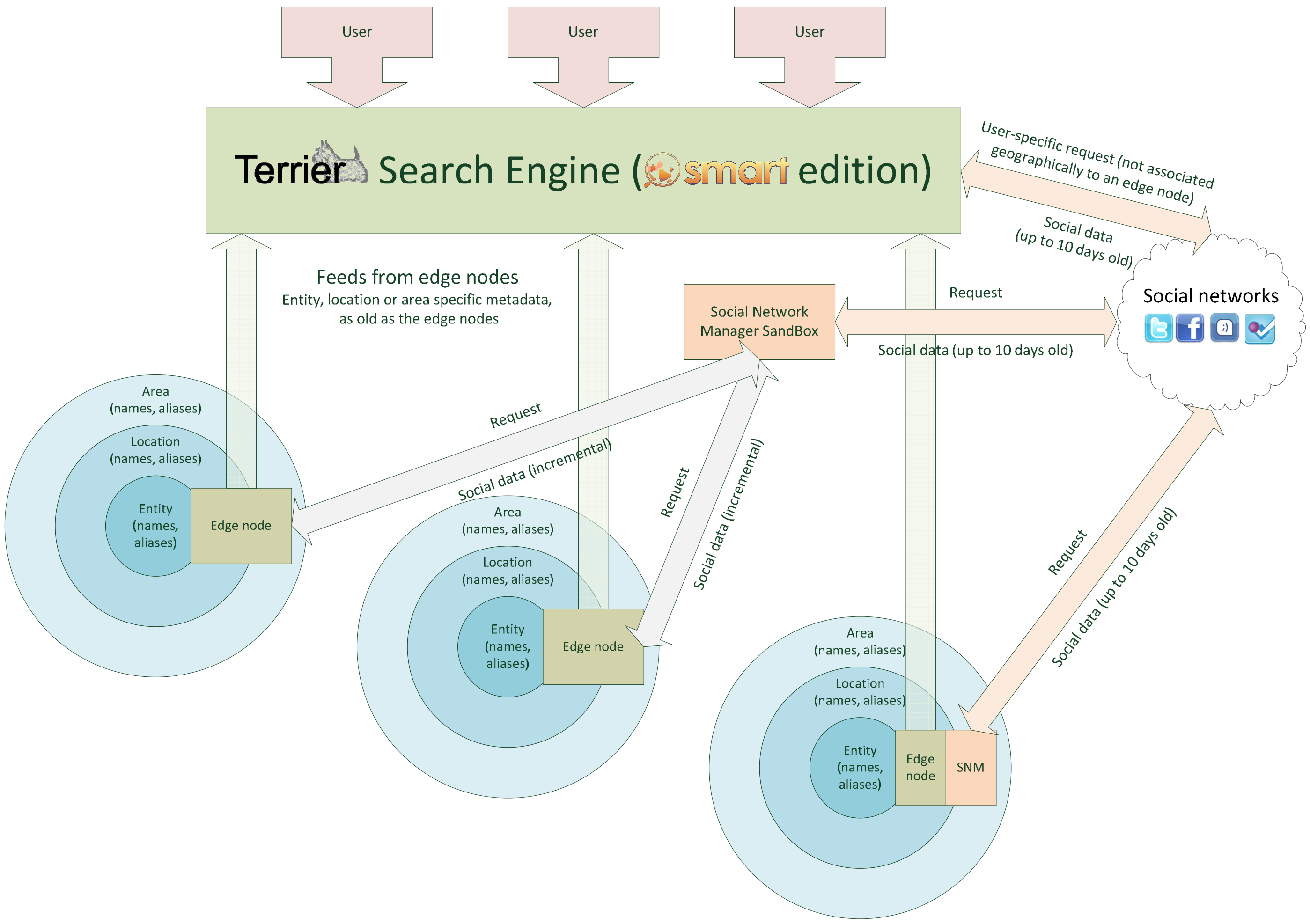

User
User
User
Terrier Search Engine (smart edition)
User-specific request (not associated geographically to an edge node)
Social data (up to 10 days old)
Feeds from edge nodes
Entity, location or area specific metadata, as old as the edge nodes
Social Network Manager SandBox
Request
Social data (up to 10 days old)
Social networks
Area (names, aliases)
Location (names, aliases)
Entity (names, aliases)
Edge node
Request
Social data (incremental)
Request
Social data (incremental)
Area (names, aliases)
Location (names, aliases)
Entity (names, aliases)
Edge node
Request
Social data (up to 10 days old)
Area (names, aliases)
Location (names, aliases)
Entity (names, aliases)
Edge node
SNM

**Figure 2: Accessing social networks' information in SMART.**

The social networks are accessed by different components in SMART:

- The edge nodes receive from the social networks information associated to them. The association is based on geography, either via GPS coordinates, or via the various names and aliases that are related to the edge node. This relation can be with the entity hosting the edge node (e.g. Athens Information Technology is hosting the AIT edge node), with the location of the edge node (e.g. the AIT edge node is on Markopoulou Ave.), or with the broader area (e.g. the AIT edge node is in the town of Peania). The social networks provide information (social posts) associated to an edge node after being queried directly or indirectly:
    - o The edge node is querying the social networks directly when it uses its own built-in Social Network Manager (SNM). This is shown at the bottom-right edge node in Figure 2.

  - o The edge node is querying the social networks indirectly when it uses the SNM built by TELESTO to serve many edge nodes and as a SandBox to new users of SMART. This is shown at the top-left and the middle edge nodes and the SNM SandBox of Figure 2.
- The SMART search layer queries the social networks for user-specific requests that cannot be geographically associated to any edge node. This is shown by the bidirectional arrow connecting the SMART search layer and the social networks' cloud in Figure 2.

Both means of access to social networks actually provide a wrapper around the individual social networks APIs, both simplifying the procedure and removing temporal constraints imposed by the APIs. E.g. the Twitter API only allows access to tweets that are up to 10 days old, whereas the process of indexing tweets of interest make them available since the life span of the SMART system or the particular edge node.

Why do we need this dual mode for accessing social networks information? Because the search engine does not know the edge nodes as well as the people that are managing the edge nodes. So we follow the decentralised approach, of having the edge node managers define what information is related to the edge node and try to collect it from the social networks. Since most of the edge nodes are expected to have their own SNM modules, this approach takes off the burden imposed by the large amount of global requests the search engine makes to a single instance of the SNM, namely the SandBox operated by TELESTO located at http://telesto.zapto.org:81/SocialNetworkManager/.

The edge node access to the social networks is the topic of this deliverable. This involves:

- The social network manager that handles information requests (Chapter 3),
- The social metadata streams (Chapter 4), and
- The fusion of social metadata with physical (sensor) metadata (Chapter 5).

# 3 Social Networks in SMART

In this chapter we discuss the social networks we currently use within SMART and then we detail the Social Network Manager that populates social feeds in the SMART Edge Nodes.

## 3.1 Social networks of interest

### 3.1.1 Twitter

Twitter is an online social networking service and a micro-blogging service that enables its users to send and read text-based messages of up to 140 characters, known as "tweets". It was created in March 2006 by Jack Dorsey and by July 2006, the social networking site was launched. The service rapidly gained worldwide popularity, with over 500 million registered users as of 2012, generating over 340 million tweets daily and handling over 1.6 billion search queries per day [6].

Information can be accessed through the Twitter API and it mainly consists of Tweets, which are the basic atomic building block of all things, also known more generically as "status updates". Also, Users (can be anyone or anything), Entities (provide metadata and additional contextual information about content posted on Twitter such as Hash-tags, Media, or URLs) and Places (specific named locations with corresponding geo coordinates) can be queried and retrieved through the API.

The Social Network Manager uses Twitter4J, an unofficial Java library to easily integrate with the Twitter service and communicate with the API [7].

### 3.1.2 Facebook

Facebook is a popular free social networking website that allows registered users to create profiles, upload photos and videos, send messages and keep in touch with friends, family and colleagues. It was founded by 2004 by Harvard student Mark Zuckerberg and began as a college networking website. Facebook users create a profile page that shows their friends and networks information about themselves. The choice to include a profile in a network means that everyone within that network can view the profile. The profile typically includes the following: Information (Gender, Current Location, Education etc.), Status, Friends, Friends in Other Networks, Photos, Notes, Groups, and The Wall.

Facebook’s API is called “The Graph API” and is the primary way to get data in and out of Facebook's social graph. It's a low-level HTTP-based API that cen be used to query data, post new stories, create check-ins or any of the other tasks that an application associated with its user’s Social Media might need to do. Facebook's Open Graph allows you to define new objects and actions in a user's social graph, and the way that you create new instances of those actions and objects is via the Graph API. The API provides the ability to query nodes in the graph, but also query connections between nodes in the graph. Those connections are what allows to post to a user's timeline, create location check-ins or work with a user's photos.

Facebook provides a large amount of information for a user such as age and gender to user status or user likes. Also, general information, such as Facebook pages and posts, can be queried.

The Social Network Manager uses RestFB, a simple and flexible Facebook Graph API client written in Java, to communicate with the Facebook API service. ResftFB is an open source software released under the terms of the MIT License.

### 3.1.3 Foursquare

Foursquare is a location-based social networking website for mobile devices, such as smartphones. Users "check in" at venues using a mobile website, text messaging or a device-specific application by selecting from a list of venues the application locates nearby. Location is based on GPS hardware in the mobile device or network location provided by the application. Each check-in awards the user points and sometimes "badges". The service was created in 2009 by Dennis Crowley and Naveen Selvadurai.

The Foursquare API provides methods for accessing a resource such as a venue, tip, or user, at a canonical URL. Given a resource, the client can then drill into a particular aspect of the resource such as the tips of a given venue or a series of actions associated with it can be accessed such as liking the given venue if the client has already connected with an active foursquare account.

Social Network Manager uses Foursquare V2 API for Java, an open source easy library implementing the use of the foursquare API using the Java programming language.

## 3.2 Social Network Manager

Social Network Manager is an end-point social data gathering tool for the SMART Edge Node. It is implemented through specific drivers for each supported Social Network accessed via a restful API which offers Social Data querying and gathering from Social Networks around the internet.

### 3.2.1 Overview

A major target of SMART is the combination of information stemming from sensors as well as social networks, in order to provide accurate and useful answers to queries related to the physical world at all times according to the imported data. Part of this imported data comes from multiple social networks and for the extraction and fusion of it, SMART Edge Nodes incorporate a component named Social Network Manager. The architecture adopted by the Social Network Manager focuses on allowing the edge node, external users, or external software clients to query, filter and retrieve social network data in a uniform manner.

The clients that use the Social Network Manager will be mainly other SMART components requesting social data and elaborating on them or using them as input for more complicated computations. However, a Social Network Manager client can also be a person accessing directly the restful API through an easy to use Web Application Interface called the SNM SandBox .

### 3.2.2 Architecture

Figure 3 illustrates the architecture of the Social Network Manager. The Social Network Manager consists of **Drivers**, each implementing the use of a specific Social Network API for each of the supported Social Networks, **Filters** that implement filtered information pulling such as the Twitter DateTime Extractor filter providing date and time information about the requested keyword or sentence and the **General Social Network Rest API** that is used to call the Drivers and the Filters in a Restful way and perform the requested queries and Social Data gathering. As a result the Rest API is the component giving access to all the Social Network Manager functionalities and provides the connection interface with the other SMART components.

### 3.2.3 Social Network Manager API Implementation

The Social Network Manager uses the **REST** architecture for its Application Programming Interface to implement the communication between Client and Server. **Representational State Transfer** (**REST**) is a style of software architecture for distributed systems such as the World Wide Web. REST has emerged as a predominant Web service design model using mainly, but not limited to, HTTP methods such as GET, PUT, POST, DELETE.

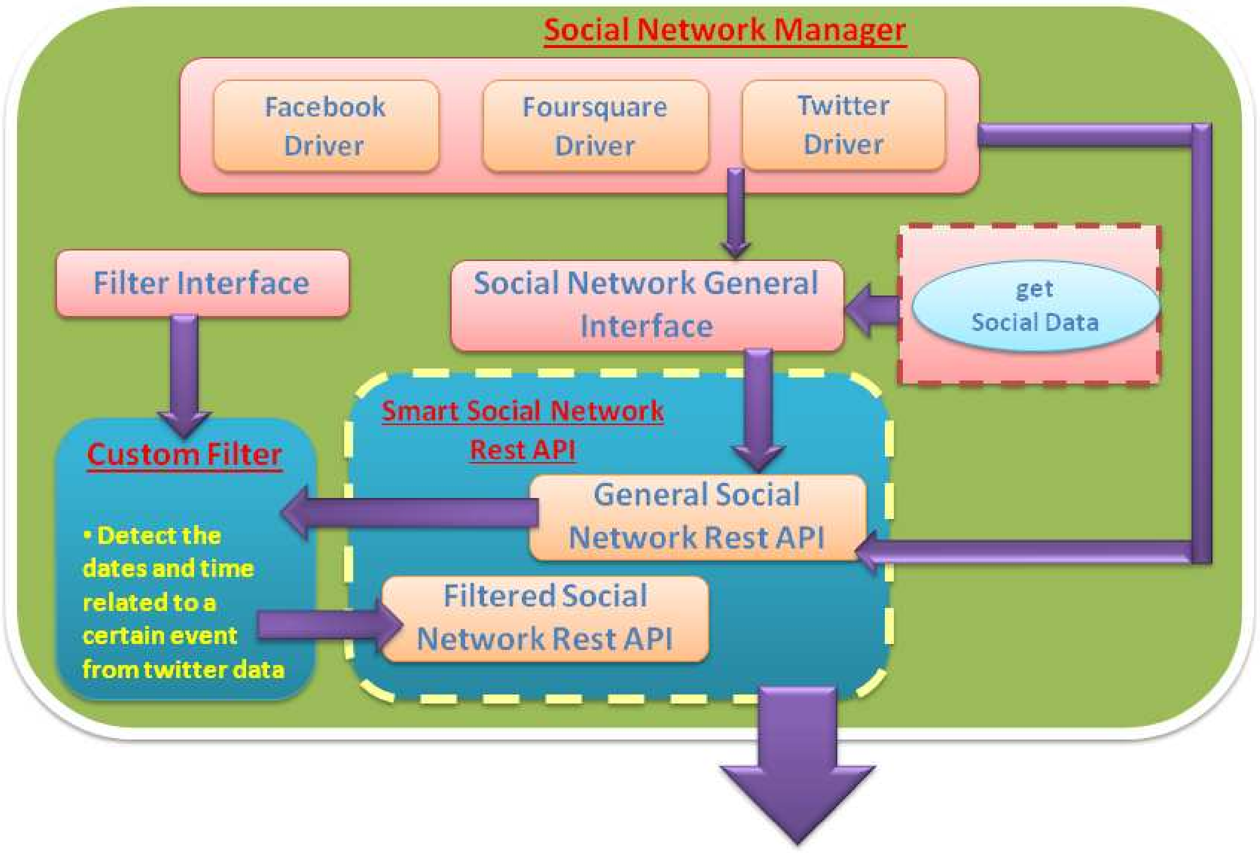

Social Network Manager
Facebook Driver
Foursquare Driver
Twitter Driver
Filter Interface
Social Network General Interface
get Social Data
Custom Filter
• Detect the dates and time related to a certain event from twitter data
Smart Social Network Rest API
General Social Network Rest API
Filtered Social Network Rest API

**Figure 3: SMART Social Network Manager Architecture Overview.**

### 3.2.4 General Social Network API

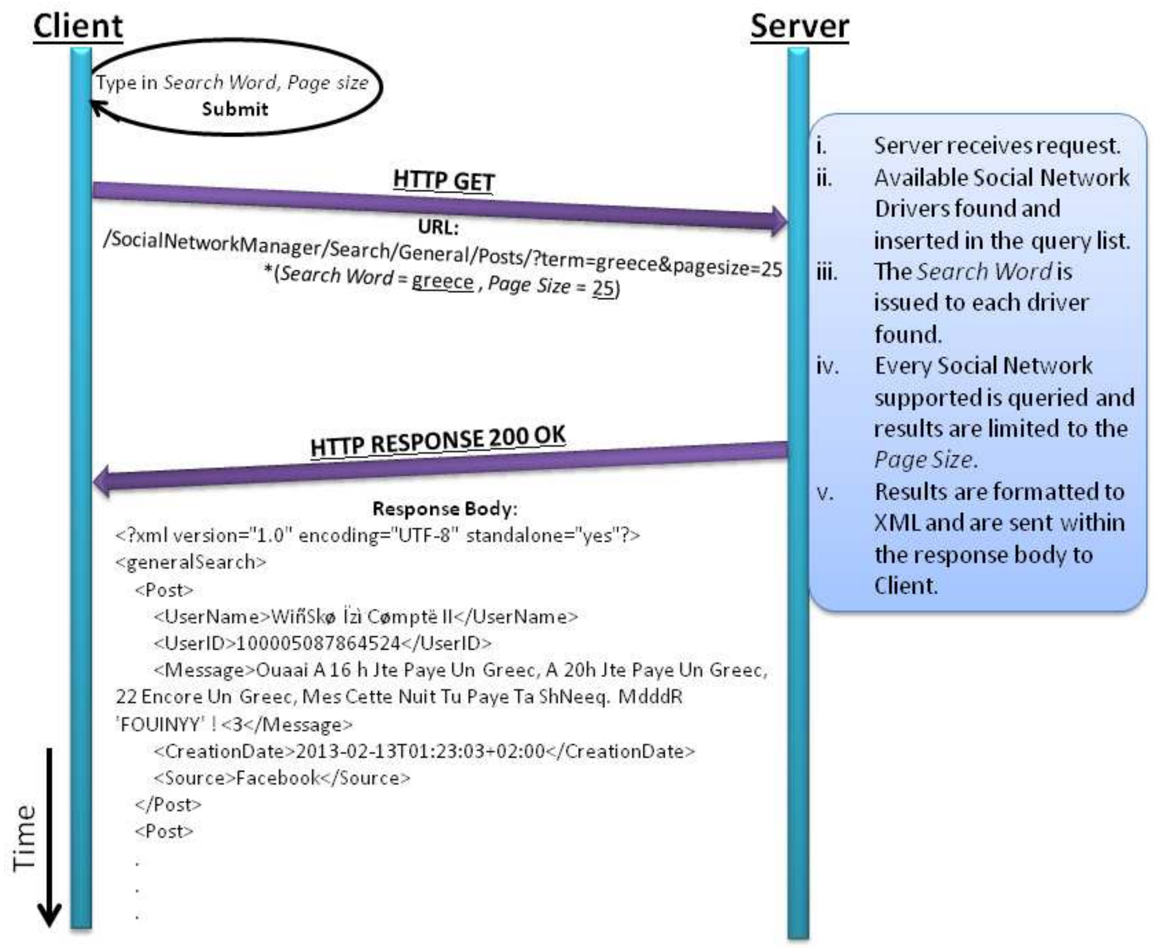
The General Social Network API focuses on allowing applications using the Social Network Manager to perform queries to access Social Data. The rest of this section explains the usage of the calls included in this API:

#### 3.2.4.1 Querying Social Data on all available social networks with one call

The General Social Network API allows clients query all the attached social networks using one call for convenience purposes. The parameters of the query must include the queried term as well as the number of the retrieved results per page. Results are encoded in both RDF and XML formats. Following the implementation, Figure 4 depicts an example flow diagram of a general query.

**Implementation:**

| |
|---|
| ***Query social data from all the supported Social Networks*** |
| **Http Request Type**: GET |
| **Header**: Accept:text/xml |
| **Parameters**: term-> Queried Term, pagesize->Number of Results per page |
| **Body:** Null |
| **URL**: /SocialNetworkManager/Search/General/Posts/?term='provide_a_term'&pagesize=provide_a_number |
| **Description**: Enables users to query social data from **all** the supported social networks using the aforementioned call |


Client
Server
Type in *Search Word*, *Page size*
Submit
HTTP GET
URL:
/SocialNetworkManager/Search/General/Posts/?term=greece&pagesize=25
*(*Search Word* = greece , *Page Size* = 25)
HTTP RESPONSE 200 OK
Response Body:
Time

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<generalSearch>
  <Post>
    <UserName>WiñSkø İzi Cømptë II</UserName>
    <UserID>100005087864524</UserID>
    <Message>Ouaai A 16 h Jte Paye Un Greec, A 20h Jte Paye Un Greec,
22 Encore Un Greec, Mes Cette Nuit Tu Paye Ta ShNeeq. MdddR
'FOUINYY' ! <3</Message>
    <CreationDate>2013-02-13T01:23:03+02:00</CreationDate>
    <Source>Facebook</Source>
  </Post>
  <Post>
  .
  .
  .
```

i. Server receives request.
ii. Available Social Network Drivers found and inserted in the query list.
iii. The *Search Word* is issued to each driver found.
iv. Every Social Network supported is queried and results are limited to the *Page Size*.
v. Results are formatted to XML and are sent within the response body to Client.

**Figure 4: Query Example Flow Diagram.**

### 3.2.4.2 Querying a certain social network with one call

Additionally, the General Social Network API allows clients to query a specific social network through its driver. The parameters of the query must include the queried term as well as the number of the retrieved results per page. Results are encoded in both RDF and XML formats as well. The contents of the results are again limited to posts.

**Implementation:**

| |
|---|
| ***Query social data from one of the supported Social Networks*** |
| **Http Request Type**: GET |
| **Header**: Accept:text/xml |
| **Parameters**: **Preferred Social Network** -> the Name Of the Social Network Driver, **term**-> Queried Term, **pagesize**->Number of Results per page |
| **Body:** Null |
| **URL**: SocialNetworkManager/Search/General/{Preferred Social Network}Driver/Posts/?term={ term }&pagesize={ pagesize } |
| **Description**: Enables users to query social data from **one** of the supported social networks using the aforementioned call |

### 3.2.4.3 Browsing through pages of results

Since the information pulled by the social network manager is sometimes quite long, a paging mechanism is used to handle the load. Therefore, the Social Network Manager API provides an interface that allows users to browse through the retrieved data.

**Implementation:**

| |
|---|
| ***Retrieve Next Page of Results*** |
| **Http Request Type**: GET |
| **Header**: Accept:text/xml |
| **Parameters**: (Optional) **Preferred Social Network** -> the Name Of the Social Network Driver |
| **Body:** Null |
| **URL**:<br>/SocialNetworkManager/Search/General/{Preferred Social Network}Driver/Posts /nextpage<br>or<br>/SocialNetworkManager/Search/General/Posts/nextpage |
| **Description**: Enables users to retrieve the next page of results |

| |
|---|
| ***Retrieve Previous Page of Results*** |
| **Http Request Type**: GET |
| **Header**: Accept:text/xml |
| **Parameters**: (Optional) **Preferred Social Network** -> the Name Of the Social Network Driver |
| **Body:** Null |
| **URL**:<br>/SocialNetworkManager/Search/General/{Preferred Social Network}Driver/Posts /previouspage<br>or<br>/SocialNetworkManager/Search/General/Posts /previouspage |
| **Description**: Enables users to retrieve the previous page of results |

### 3.2.4.4 Using custom information pulling functionality

Since the functionalities of Social Networks may vary, the Social Network Manager Drivers can be extended using custom information pulling functionalities to meet the user's demands. The API provides access to these functionalities through a call that is used to send xml descriptions of them. Users developing custom information pulling functionalities must specifically encode their results in XML and RDF.

**Implementation:**

| |
|---|
| ***Call a non-standard driver method*** |
| **Http Request Type**: PUT |
| **Header**: Content Type: text/xml |
| **Parameters**: none |
| **Body:**<br>&lt;?xml version="1.0" encoding="UTF-8" standalone="yes"?&gt;<br>&lt;driverSpecificCall&gt; |

| |
|---|
| &lt;ClassName&gt;{Here you need to specify the name of the Driver class that contains the method}&lt;/ClassName&gt;<br>&lt;MethodName&gt;{Here you need to specify the name of method}&lt;/MethodName&gt;<br>&lt;ArgTypes&gt;{Here you need to specify the first Argument Type of the method}&lt;/ArgTypes&gt;<br>&lt;ArgTypes&gt;{Here you need to specify the second Argument Type of the method}&lt;/ArgTypes&gt;<br>…<br>&lt;ArgTypes&gt;{Here you need to specify the N Argument Type of the method}&lt;/ArgTypes&gt;<br>&lt;ArgValues&gt;{Here you need to specify the Value for the first Argument}&lt;/ArgValues&gt;<br>&lt;ArgValues&gt;{Here you need to specify the Value for the second Argument}&lt;/ArgValues&gt;<br>…<br>&lt;ArgValues&gt;{Here you need to specify the Value for the N Argument}&lt;/ArgValues&gt;<br>&lt;/driverSpecificCall&gt; |
| **URL**: SocialNetworkManager/Search/General//NonStandard/DriverMethod |
| **Description**: Enables users to retrieve data by calling a non standard method included in a specific driver. |

### 3.2.5 Filtered Social Network API

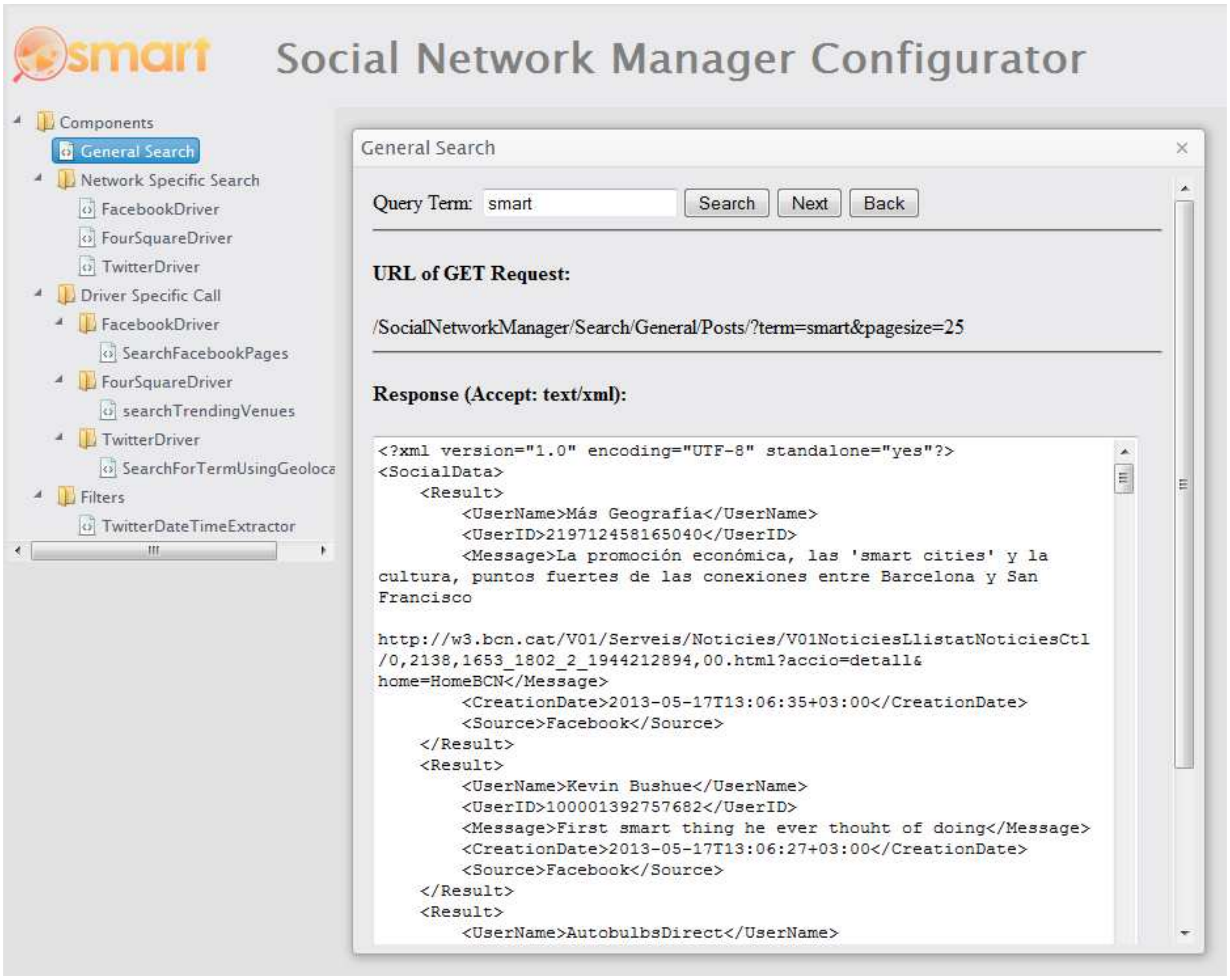
The Filtered Social Network API provides a mechanism for accessing data from custom filters. The URL of each filter consists of a static part and a dynamic part that changes depending on the class name of the requested filtered. Data retrieved from filters is encoded in XML or RDF.

**Implementation:**

| |
|---|
| **Retrieve Data From Filter** |
| **Http Request Type:** GET |
| **Header: Accept:** text/rdf |
| **Parameters: (Optional) Filter Name ->** the Name Of the Social Network Filter |
| **Body:** Null |
| **URL:** /SocialNetworkManager/Search/General/Filter/{FilterName} |
| **Description:** Enables users to retrieve data from an attached filter. Data will be encoded in RDF |

### 3.2.6 The Social Network Manager SandBox

For testing purposes, as well as the ease of use for human clients, a Web Application Interface is built based on Kendo UI, a comprehensive HTML5 - JavaScript framework which provides an easy way to access the restful API. The application front-end structures the available drivers and functions on a TreeView on the left side of the screen and upon a click on the functions, it provides a separate window as an interface for each function. All the Rest API calls are automatically created and sent on demand based on the arguments provided on the specific fields. Figure 5 shows a snapshot of the Kendo UI to access the restful API of the Social Network Manager.



**URL of GET Request:**

/SocialNetworkManager/Search/General/Posts/?term=smart&pagesize=25

**Response (Accept: text/xml):**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SocialData>
    <Result>
        <UserName>Más Geografía</UserName>
        <UserID>219712458165040</UserID>
        <Message>La promoción económica, las 'smart cities' y la
cultura, puntos fuertes de las conexiones entre Barcelona y San
Francisco

http://w3.bcn.cat/V01/Serveis/Noticies/V01NoticiesLlistatNoticiesCtl
/0,2138,1653_1802_2_1944212894,00.html?accio=detall&
home=HomeBCN</Message>
        <CreationDate>2013-05-17T13:06:35+03:00</CreationDate>
        <Source>Facebook</Source>
    </Result>
    <Result>
        <UserName>Kevin Bushue</UserName>
        <UserID>100001392757682</UserID>
        <Message>First smart thing he ever thouht of doing</Message>
        <CreationDate>2013-05-17T13:06:27+03:00</CreationDate>
        <Source>Facebook</Source>
    </Result>
    <Result>
        <UserName>AutobulbsDirect</UserName>
```

**Figure 5: Social Network Manager SandBox usage example.**

### 3.2.7 Supporting multiple social networks

To provide support for multiple social networks, the SMART Social Network Manager uses Drivers. Custom Drivers may also be added to support new social networks. Although drivers must adhere to a certain template, their functionalities are not limited by it. This freedom is granted to Driver developers, since most social networks have some special features. Therefore, in order to support these features, Social Network Drivers can be extended using functionalities that are not included in the template. However, the basic functions defined in the Social Network General Interface, which acts as the Driver Template, should be implemented by all the attached drivers. At this point, we should stress the fact that the Social Network General Interface, acts as an abstraction layer, hiding the heterogeneities caused by the multiple sources of incoming data. In order to perform social queries, clients of the Social Network Manager component may access the implementations of the standard information pulling functions included in the Social Network General Interface using the General Social Network Rest API. This API also enables them to access non-standard functionalities that are tied to a certain Social Network and are included within a certain driver. However, to do so, they must provide an XML description of that call through the REST API using an HTTP Put Request. The aforementioned approach should enable developers that write new Drivers for the social network manager to:

- Extend the system with non-standard functionalities that have not been predicted.
- Extend the system without having to involve themselves with the code of the Social Network Manager.

- Make these non-standard functionalities accessible through the General Social Network Rest API, without having to make any modifications to the code of the Social Network Manager.

### 3.2.8 Custom Social Network Data Filtering

To benefit from further processing of Social Network Data, the Social Network Manager allows the inclusion of custom filters. These filters have the form of Java .class files that can be directly injected within the .WAR package of the Social Network Manager in order to extend it. Their main purpose is the filtering of social network content for the detection of events or other useful information in social media content. As with the custom Drivers, the custom filter classes adhere to the SocialNetworkFilter.class as an Interface to begin with and can be extended to the developer's needs. Data produced by the filters, is accessible though the Filtered Social Network API and is encoded in XML or RDF.

Social Network Manager already contains such a Filter. The Twitter DateTime Extractor filter currently uses simple if then statements in order to segment time and date related text from the text contained in twitter posts. This information is then encoded in RDF format along with the queried word. Additionally information about whether the post describes an active event is inferred by the dates extracted from the text and is included in the results in the form of a Boolean value. The user may retrieve the formalized filter output through the equivalent Social Network Manager API call. In order to encode data in RDF the JENA library has been utilized. An example use of the filter along with results is provided in Section 4.1.

# 4 Social Network Data in the in the SMART Edge Node

Chapter 3 described the social networks already considered in SMART and the SNM that captures information from them. This chapter is about the metadata extracted from the social networks. It begins with the streams the various components of the SNM provide and then groups these metadata into two families of RDF triples.

## 4.1 Social information collected

Three different streams of social metadata are currently supported:

- Applications are able to query Twitter and Facebook APIs for tweets, Facebook posts, users and pages containing a word or sentence of interest through the Social Network Manager.

  *Example:*

  **URL** of **GET** Request**:**

  /SocialNetworkManager/Search/General/Posts/?term=nature&pagesize=25

  Returned information:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SocialData>
    <Result>
        <UserName>Mein Garten und mehr.</UserName>
        <UserID>286121554794299</UserID>
          <Message>Mai, das Wonnemonat in der Natur - morgen, Freitag 18 Uhr ist
          Ausstellungseröffnung von "Lieblingsstücken" bei uns im Garten. Öffnungszeiten
          sind Mittwoch bis Sonntag immer von 14-18 Uhr. Mehr Information:
          https://www.facebook.com/galeriepluskunst</Message>
        <CreationDate>2013-05-09T10:54:57+03:00</CreationDate>
        <Source>Facebook</Source>
    </Result>
    <Result>
        <LocationName>Denver Museum of Nature and Science</UserName>
        <LocationID>4ae59e3ef964a52016a121e3</UserID>
        <LocationType>Science Museum</LocationName>
        <GeoLatitude>39.74810479621085</GeoLatitude>
        <GeoLongitude>-104.94275541615056</GeoLongitude>
        <Source>FourSquare</Source>
    </Result>
    <Result>
        <UserName>Yancho04</UserName>
        <UserID>1036935042</UserID>
          <Message>RT @PrayInFaith: All are but parts of one stupendous whole, Whose
          body Nature is, & God the soul. -Alexander Pope, 1734</Message>
        <CreationDate>2013-05-09T10:59:29+03:00</CreationDate>
        <Source>Twitter</Source>
    </Result>
</SocialData>
```

- The TwitterDateTimeExtractor filter component returns dates and times that come with the term of interest in Twitter as well as if the particular event is currently active or not. A sample text analysis is planned to be added to this component by taking into account words like “now” , “start – begin” , “finish – end” to help identifying even better if the event is currently active and connect the date/times found with the beginning or the ending of the event.

  *Example:*

  **URL** of **GET** Request**:**

  /SocialNetworkManager/Search/General/Filter/TwitterDateTimeExtractor?term=football

  Returned information:

```
<rdf:RDF
```

```
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:j.0="http://SocialNetworkManager/Event/">
    <rdf:Description rdf:about="http://SocialNetworkManager/activity">
        <j.0:hasName>football</j.0:hasName>
        <j.0:isActive>false</j.0:isActive>
        <j.0:hasDate>09-05-2013</j.0:hasDate>
    </rdf:Description>
</rdf:RDF>
```

- The SNM also provides querying on tweets based on geo-location given by the client along with the term/phrase of interest.

  *Example:*

  *(Driver Specific Call)*

  **URL** of **PUT** Request:

  SocialNetworkManager/Search/General/NonStandard/DriverMethod

  User provides the following **Body** of **PUT** Request:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
     <driverSpecificCall>
     <ClassName>TwitterDriver</ClassName>
     <MethodName>SearchForTermUsingGeolocation</MethodName>
     <ArgTypes>java.lang.String</ArgTypes>
     <ArgTypes>java.lang.Integer</ArgTypes>
     <ArgTypes>java.lang.Double</ArgTypes>
     <ArgTypes>java.lang.Double</ArgTypes>
     <ArgTypes>java.lang.Double</ArgTypes>
     <ArgValues>science</ArgValues>    // Term
     <ArgValues>25</ArgValues>         // Max Result Pages
     <ArgValues>37.975556</ArgValues>  // Geo Latitude
     <ArgValues>23.734722</ArgValues>  // Geo Longitude
     <ArgValues>2000</ArgValues>       // Radius
</driverSpecificCall>
```

  Returned information:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<SocialData>
    <Result>
        <UserName>PhilosL</UserName>
        <UserID>814716962</UserID>
          <Message>AMENDMENT Workshop: Internal Critique, University of Essex, 10/11
          May:     Please     note     the     amended     schedule     for     Sa...
          http://t.co/916610zXfU</Message>
        <CreationDate>2013-05-09T12:41:37+03:00</CreationDate>
        <Source>Twitter</Source>
    </Result>
          .
          .
          .
</SocialData>
```

## 4.2 Social information as RDF triples

It is evident from the examples of the information collected from the social networks in Section **Error! Reference source not found.** that social metadata are obtained from two components, one that gets results (posts) and another that extracts descriptions for certain activities (currently temporal information).

The edge node reasoning mechanism for social and physical metadata (presented in Chapter5) expects the information in RDF triples, namely in the form:

*<predicate>*(*<subject>*,*<object>*)

where <predicate> is the attribute that needs quantifying, <subject> is the entity the attribute refers to and <object> is the quantity the attribute receives.

In the following two sub-sections, we consider the information in RDF form, by defining subjects for the two components and their associated possible predicates.

### 4.2.1 Collecting social posts (results)

We support two ways of receiving results from the SNM (i.e. posts from the social networks) into the edge node:

- **By keyword**. Here we specify keywords of interest to the edge node. These usually include names of important entities at or near the edge node.
- **By area description**. Here we give GPS coordinates and a radius of interest around them, or a location name. The second option is less reliable, since there can be different names at different levels of area description, but we assume that the person managing the edge node has this information.

The information we receive from the SNM concerns the subject <Result>, and is organized as objects to all or some of the predicates in Table 1.

**Table 1: Predicates related to the subject <Result> and their meaning.**

| Subject | Predicate | Notes |
|---|---|---|
| Result | UserName | Human & machine readable name of entity issuing the post |
| | UserID | |
| | Message | Text of post |
| | CreationDate | Date of publication |
| | Source | Social network of publication |
| | LocationName | Description of location of publication by name or GPS coordinates |
| | GeoLatitude | |
| | GeoLongitude | |

### 4.2.2 Collecting event descriptions

We can get a description for any activity. Currently, we support the extraction of temporal information from posts referring to some activity via the DateTimeExtractor filter (see Section 3.2.8). The activity of interest is specified by a keyword. The information we receive from the SNM concerns the subject <Description>, and is organized as objects to all or some of the predicates in Table 2.

In addition to the predicates shown in Table 2 (see below), we are adding some more describing term frequencies, or equivalent term mentions in the social media during a past interval of interest.

The details of this group of predicates are not yet finalized.

**Table 2: Predicates related to the subject <Description> and their meaning.**

| Subject | Predicate | Notes |
|---|---|---|
| Description | hasName | Activity name (the keyword) |
| | isActive | True if there is indication in the extracted temporal hints that the activity is currently running |
| | hasDate | Date and time obtained as temporal hints, not associated with the start or end of the activity |
| | hasTime | |
| | hasStartDate | Date and time associated with the start of the activity |
| | hasStartTime | |
| | hasEndDate | Date and time associated with the end of the activity |
| | hasEndTime | |

# 5 Social Network Data in the Reasoning Layer

In this chapter we discuss why (Section 5.1) and how (Sections 5.2 and 5.3) the fusion of social and physical metadata is performed in the reasoning engine of the SMART edge node.

## 5.1 Rationale

The following two subsections detail our live news and security scenarios for fusing social with physical metadata.

### 5.1.1 Live news scenario

Social networks can inform us on something that is about to happen, or is happening now. Consider some example tweets:

S1. “Now at Santander City Hall square listening to a street band” tells us that a street band is playing, but not if there is a significant audience, nor if they are enjoying it.
S2. “Off to Santander City Hall square for some live music” tells us there will be some band playing music there, but not exactly when.

There are also the physical sensors in place, with the SMART perceptual algorithms processing` their signals. Here are three possible sets of low level events:

P1. Audio event classification reporting repeating intervals of music, applause and crowd noises. Video crowd analysis reporting large crowd density without significant motion.
P2. Audio event classification reporting sparse crowd noises and occasional music intervals. Video crowd analysis reporting medium crowd density with significant motion along the horizontal direction.
P3. Audio event classification reporting only sparse crowd noises. Video crowd analysis reporting medium crowd density with significant motion along the horizontal direction.

Now what information can we infer by combining the social and the physical information? Here are examples:

- Combining the social information S1 with the physical information P1 tells us that the on-going event of a band playing has attracted lots of attention and the people are having a good time.
- Combining the social information S2 with the physical information P3 tells us that the event of a band playing has either not started or has ended.
- Combining the social information S2 first with the physical information P1 and then with P3 tells us that the event of a band playing has attracted lots of attention but now has ended.
- Combining the social information S1 with the physical information P2 tells us that the on-going event of a band playing has attracted little attention.

Obviously by combining low level events, like sequences of audiovisual observations and social information can give SMART a better understanding of what is happening.

### 5.1.2 Security scenarios

From the security viewpoint, data from social networks can be used both to provide decision support and to prevent risks. To better understand how to use data from social networks, we identify two different scenarios.

The first scenario involves crowd reacting to a security-related event. The situation at hand is either crowd panic (following events like explosion, fire, earthquake), or gathering and attending at the event scene (following milder events like accident, crime).

SMART can understand such events by the following sequence of observations:

- Initial observation of abrupt physical evidence at event scene (abrupt changes of sound, temperature or light levels, audio event classification).
- Followed by progressive build-up of more physical evidence after the event (crowd analysis: formation, direction, audio analysis: crowd noises).

- Accompanied by social evidence (density of posts, emotion in posts).

In this first scenario, SMART cannot be used to prevent the risk, but rather to assess it: a security operator can use physical and social feeds from a specific location to better identify what happened, possible damages or victims. Such information is useful in planning the intervention.

The second scenario involves crowd actually causing a security-related event. The situation at hand can be about hooligans.

SMART can understand such events by the following sequence of observations:

- Initial observation of progressive build-up of physical evidence (crowd analysis: formation, direction, audio analysis: crowd noises).
- Accompanied by social evidence like density of posts, emotion in posts and keywords (e.g. team names) in posts.

In this second scenario SMART can be used to prevent the event: a security operator can use physical and social feeds to understand the risk associated with the event, like crowd mood and next moves.

## 5.2 Intelligent Fusion Manager

Each SMART edge node has an Intelligent Fusion Manager (IFM) where the reasoning about incoming data takes place. At the heart of the IFM there is the Alchemy reasoning engine which is an implementation of Markov Logic Networks (MLN) [11].

In learning mode, the engine takes a bunch of input data and outputs some rule-based patterns accounting for these data. In the inference mode, given some sensors' data and some rule-based patterns, the engine derives new information with some attached probabilistic weight. An example is given in the next section.

The SMART system is meant to fuse and reason over data from heterogeneous sources. For example, we have shown in [12] how to combine data from audio or video with data from sensors like thermometers to enrich the reasoning about a situation. Figure 6 shows how data is diffused within the reasoning engine.

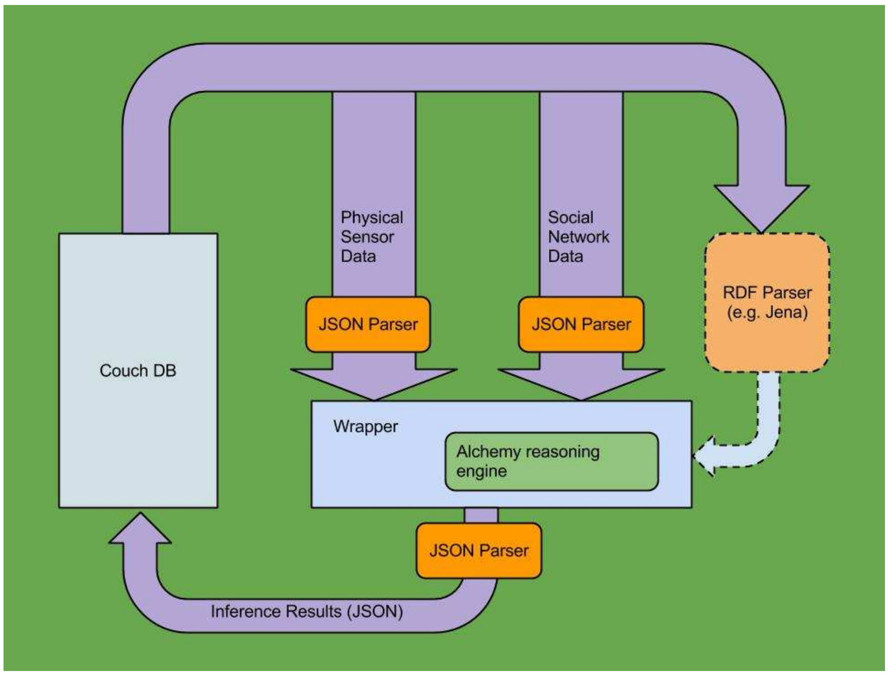


**Figure 6: Flow of data within the reasoning engine.**

This provisional configuration pinpoints the fact that while using XML parsers can help with regards to extracting the relevant information from the Social Networks and Physical sensors data, a Semantic Web framework as JENA can be used to handle valid W3C RDF data (more information on Jena can be found in [13]). This can then create the potential for using additional linked data sources (including open source data).

Both the data from sensors and from reasoning engine are meant to be stored in the database of the edge node (CouchDB, see [14]).

### 5.2.1 Markov Logic Networks

As stated before the reasoning engine uses Markov Logic Networks (MLN). It is an approach based on combining first-order logic (FOL) and graphical models into a single representation. Here, we give a simple presentation of MLN from [15] where the reader can find an exhaustive documentation.

In the most common settings, a first-order knowledge base is a set of hard constraints on the set of possible worlds i.e. if a world violates one formula, it will have zero probability. In MLN, these constraints are softened so that a world violating a formula is still possible but less probable. Equivalently, the fewer formulas a world violates, the more probable it is.

In practice, a MLN is a graphical representation of a first-order knowledge base with a weight attached to each formula. Each node of the graph is a grounded atom. There is an edge between two nodes if and only if the corresponding ground atoms appear together in at least one grounding of a formula.

A reproduced example that illustrates the basic idea of MLN can be found in Table 3 and Figure 7.

**Table 3: MLN example.**

| English | First-Order Logic | Clausal Form | Weight |
|---|---|---|---|
| Friends of friends are friends. | $\forall x \forall y \forall z\ \mathtt{Fr}(x,y) \wedge \mathtt{Fr}(y,z) \Rightarrow \mathtt{Fr}(x,z)$ | $\neg\mathtt{Fr}(x,y) \vee \neg\mathtt{Fr}(y,z) \vee \mathtt{Fr}(x,z)$ | 0.7 |
| Friendless people smoke. | $\forall x\ (\neg(\exists y\ \mathtt{Fr}(x,y)) \Rightarrow \mathtt{Sm}(x))$ | $\mathtt{Fr}(x, g(x)) \vee \mathtt{Sm}(x)$ | 2.3 |
| Smoking causes cancer. | $\forall x\ \mathtt{Sm}(x) \Rightarrow \mathtt{Ca}(x)$ | $\neg\mathtt{Sm}(x) \vee \mathtt{Ca}(x)$ | 1.5 |
| If two people are friends, either | $\forall x \forall y\ \mathtt{Fr}(x,y) \Rightarrow (\mathtt{Sm}(x) \Leftrightarrow \mathtt{Sm}(y))$ | $\neg\mathtt{Fr}(x,y) \vee \mathtt{Sm}(x) \vee \neg\mathtt{Sm}(y)$, | 1.1 |
| both smoke or neither does. | | $\neg\mathtt{Fr}(x,y) \vee \neg\mathtt{Sm}(x) \vee \mathtt{Sm}(y)$ | 1.1 |

Figure 7 shows the graph of the grounded Markov network defined by the last two formulas of the table above and the constants Anna (A) and Bob (B).

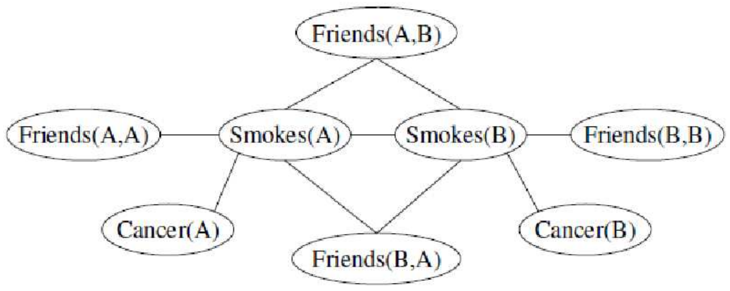


**Figure 7: Graph of the grounded Markov network defined by the last two formulas of Table 3.**

With this graph, we will be able to compute the probability that Anna and Bob are friends given their smoking habits, the probability that Anna has cancer given she is friend with Bob, etc. In fact, a ground MLN specifies a joint probability distribution over possible worlds (i.e. truth value assignments to all ground atoms).

The probability of a world x is given by an exponential model:

$$P(X=x) = \frac{1}{Z} \exp\left(\sum_i w_i n_i(x)\right)$$

where the sum ranges over the formulas of the knowledge base, wi is the weight of formula i, and ni(x) is the number of true grounding of the formula i in the world x. Z is a normalisation factor over all the possible worlds (Z is also called as the partition function). So, for instance, in the world x, where the ground atoms Fr(A,B), Fr(B,A) and Smokes(B) are true, then no grounding of any template formulas is true, thus, the probability of this world is: 1/ Z. In another world y where the ground atoms Fr(A,B), Fr(B,A), Smokes(B) and Cancer(B) are true then the grounding of one formula is true ¬Sm(B) v Ca(B). Thus this world has probability: exp(1.5)/Z. Hence this world y is exp(1.5) = 4.8 times more probable than the world x.

## 5.3 An example of reasoning with Social Network and Physical World Data

In this section, we present an example illustrating the fusion of with social network and physical data.

Consider the following example. Suppose that you collect for a period of time the density of the crowd in a venue by the predicate `CrowdDensityMeasurementValue`. Also, consider the frequency of a certain expression (e.g. “Live music”) in Twitter in the previous one hour represented by the predicate `OnehourFrequencyValue`. We define `Notification` as the predicate that represents the type of an event.

For each time instant, we shall collect the data in the forms of a number of triples, for instance:

```
ApplauseScoreMeasurementValue(Plaza, High)
OnehourFrequencyMeasurementValue(Livemusic, High)
CrowdDensityMeasurementValue(Plaza, High)
Notification(Event,Jubilation)
```

The above example is a scenario in which the applause score is high, the frequency of the expression “live music” in the tweets of the last hour has been high and finally the crowd density in the city plaza is high. This then should result in the notification of an event of the type “Jubilation” (e.g. a concert or festival) in the plaza.

When we have the data, we want to learn some rule-based patterns that account for such data. To do so, we declare some patterns to weight. For example, we have the event type that depends on the crowd density, the frequency of the tweets relevant to the event and the applause factor:

```
// Patterns to weight

!ApplauseScoreMeasurementValue(r, +value)  v
!OnehourFrequencyMeasurementValue(+featureOfInterestvar, +value) v
!CrowdDensityMeasurementValue(r, +value) v

Notification(Event, +eventType)
```

The result of the learning process is a file that contains the different weighted MLN patterns for each combination of the applause factor, tweet frequency, and crowd density. We present few of the patterns here:

```
-1.24084

!ApplauseScoreMeasurementValue(Plaza,Low)    v
!OnehourFrequencyMeasurementValue(Livemusic,Low)   v
!CrowdDensityMeasurementValue(Plaza,Low) v

Notification(Event,Jubilation)

...
```

```
6.51578
!ApplauseScoreMeasurementValue(Plaza,Low)     v
!OnehourFrequencyMeasurementValue(Livemusic,High)  v
!CrowdDensityMeasurementValue(Plaza,High) v
Notification(Event,Jubilation)
...
14.0996
!ApplauseScoreMeasurementValue(Plaza,High)   v
!OnehourFrequencyMeasurementValue(Livemusic,High)  v
!CrowdDensityMeasurementValue(Plaza,High) v
Notification(Event,Jubilation)
```

Informally, these rules tell us that there is a high (low) probability of a notification of type "Jubilation" when there is a high (low) crowd density, high (low) applause factor, and high (low) frequency of the words "Live music" in tweets. When we collect some new data on these variables, we can reapply these patterns to derive the probability of having an event of type "Jubilation". For example, if we obtain the following information,

```
ApplauseScoreMeasurementValue(Plaza,Low)  v
OnehourFrequencyMeasurementValue(Livemusic,Low)    v
!CrowdDensityMeasurementValue(Plaza,Low) v
```

Then we can derive the following triple with associated probability,

```
Notification(Event,Jubilation) 4.9995e-05
```

In other words, there is a "jubilation" event with a probability close to 0.

# 6 Conclusion

In this deliverable we have addressed the topic of the integration of social network and sensor network data. We detailed the architecture used for accessing social network information in SMART along with the rationale behind it. We then gave a detailed overview of the Social Network Manager component and how it handles metadata.

Finally, we described how the data from social networks can be combined with the data from sensor networks to infer high level events at the reasoning layer along with few use cases namely live news, security scenarios, and a specific example of an event recognition using reasoning by a Markov Logic Network reasoning engine.

Note that the combination of information from sensors and social networks will provide accurate and useful answers to queries and therefore is a major goal of SMART. Because some of this data comes from different social networks, Social Network Manager has been incorporated within Edge Node to enable the edge node and external users or software clients to query, filter and retrieve social network data in a uniform manner. Furthermore, other SMART components can request data and use SNM for more complicated computations.

## 7 BIBLIOGRAPHY AND REFERENCES

1. Matthew Richardson and Pedro Domingos (2006). "Markov Logic Networks" in *Machine Learning* **62** (1-2): 107–136. http://www.cs.washington.edu/homes/pedrod/papers/mlj05.pdf
2. Stanley Kok, Marc Sumner, Matthew Richardson, Parag Singla, Hoifung Poon, Daniel Lowd, Jue Wang, Aniruddh Nath and Pedro Domingos (2010). "The Alchemy System for Statistical Relational AI", Technical Report, Department of Computer Science and Engineering, University of Washington, Seattle, WA. http://alchemy.cs.washington.edu
3. SMART project, D.4.1, "SMART Distributed Knowledge Base and Open Linked Data Mechanisms"
4. SMART project, D.3.1, "Sensors and multimedia data knowledge representation"
5. SMART project, D.5.1.a, "SmartReduce Engine"
6. The Engineering Behind Twitter's New Search Experience http://engineering.twitter.com/2011/05/engineering-behind-twitters-new-search.html
7. Twitter4J, an unofficial Java library for the Twitter API. http://twitter4j.org/en/index.html
8. RestFB, a simple and flexible Facebook Graph API client written in Java. http://restfb.com/
9. Foursquare V2 API for Java, an open source easy library implementing the use of the foursquare API in Java. https://code.google.com/p/foursquare-api-java/
10. SMART project, D3.4, "Tools and Techniques for processing non-AV sensor streams"
11. S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 2007. http://alchemy.cs.washington.edu/
12. SMART project, D.4.3, "Sensors and multimedia data knowledge representation"
13. Jena™, a Java framework for building Semantic Web applications.

    http://jena.apache.org/
14. CouchDB A Database for the Web, http://couchdb.apache.org/
15. Domingos, Pedro, et al. "Unifying logical and statistical AI." *Proceedings of the Twenty-First National Conference on Artificial Intelligence*. 2006.