SEVENTH FRAMEWORK PROGRAMME

Networked Media

Specific Targeted Research Project

SMART

(FP7-287583)

Search engine for MultimediA environment generated contenT

D2.3 Multimedia Search Framework Open Architecture and Technical Specifications

Due date of deliverable: 31-08-2012

Actual submission date: 08-10-2011

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

Summary of the document

Code:	**SMART D2.3-v1.0**
Last modification:	21/09/2012
State:	Final
Participant Partner(s):	ATOS,GLA, AIT, S3LOG, TELESTO,
Author(s):	John Soldatos (AIT), Aristodemos Pnevmatikakis (AIT), Jose Miguel Garrido (ATOS), Antonis Litke (TELESTO), Iadh Ounis (GLA), Craig Macdonald (GLA), Dyaa Albakour (GLA), Regis Riveret (IMPERIAL), Massimiliano Tarquini (S3LOG), Karim Moumene (PRISA), Tomás García Fresno (SDR)
Fragment:	No
Audience:	☒ public ☐ restricted ☐ internal
Abstract:	N/A
Keywords:	N/A
References:	See within document for references

Table of Contents

Table of Figures

1 Executive Summary

1.1 Scope

This deliverable describes the technical architecture of the SMART system, including its main technical specifications. The architecture of the SMART system defines the main components of the target multimedia search engine for environment generated contents, along with a set of structuring principles specifying the integration of these components into operational search systems, applications and solutions. A key characteristic of the SMART architecture is its openness, which is reflected in the ability to enhance the SMART systems with new sensors, perceptual components (such as multimedia signal processing components), as well as with new data streams stemming from social networks. This openness of the SMART architecture is fully in-line with its on-going implementation as an open source software system, with a view to enabling the open source community to contribute to its evolution, while enabling third parties (outside) to use it for building innovative multimedia sensor search applications.

As part of the deliverable, we provide a short introduction to a range of (background) systems and technologies which have influenced the design of the SMART architecture. This introduction is not intended as an in depth review of state-of-the-art on sensor and search systems architectures. It is rather a brief reference to a set of main state-of-the-art architecture concepts that have boosted the development of the SMART architecture.

The SMART architecture and related technical specifications are presented in three parts, which correspond to three distinct layers of the architecture, namely the edge node layer dealing with information acquisition and processing from sensors and social networks, the search engine layer providing the mean for scalable efficient and real-time searching of sensor information repositories and finally the applications layer providing some utilities for application development and integration. As already outlined, special emphasis is paid not only on the introduction and presentation of these layers, but also in the definition of the main interactions between them. At the level of this deliverable the relevant interactions are presented at a high level, which detailed interactions and semantics will be presented in other deliverables of the SMART technical workpackages. The same holds for the details (e.g., algorithms) of the various modules that comprise the architecture: they will be researched and detailed in other technological workpackages of the project (notably WP3, WP4 and WP5), while this deliverable provides a main introduction in their scope and role within the overall SMART systems.

Overall, the finalization of the SMART architecture and technical specifications is an important step in the SMART project implementation roadmap, since it specifies/clarifies the modules to be developed, along with their interactions in the scope of the SMART systems. In this way it boosts the modularity of the SMART system, while it will be easing integration at later stages. Furthermore, it facilitates the management and governances of open source processes, as described in D7.4 where the responsibilities for the planning of different modules of the SMART open source software has been based on the SMART architecture. In this respect, the specifications listed in the present deliverable are an important outcome of the project.

1.2 Audience

The SMART architecture and technical specifications is of interest to a wide range of individuals within the following groups:

- **SMART software developers**: The developers of the SMART project, notably those dealing with the open source software implementation are expected to leverage the SMART architecture as a means to organize their work and interactions with other developers (working on other complementary modules of the SMART architecture).
- **The SMART project management team**: The project management team of the SMART project (at both the technical and administrative levels) will exploit the outcomes of the SMART architecture specification in order to organize, track and audit the development and the evolution of different parts of the project. It will also use the architecture as a means to regulate the relevant collaboration between partners/teams working on complementary parts.
- **Open source community**: The open source community, notably the community that will be

built around the SMART results, will use the present deliverable as a guide towards understanding the structure of the technical development of the SMART project, as well as the various components that comprise a SMART-compliant search system.

- **The Internet-of-Things community**: SMART has made use of concepts from the Internet-of-Things (IoT) discipline in specifying its architecture. We expect that the SMART architecture could also provide valuable insights to researchers/engineers working in the IoT domain.

1.3 Summary

This deliverable introduces the architecture of the SMART search systems, focusing on the main layers and components comprising a SMART compliant system, as well as on the main structuring principles that drive the integration of these components in the scope of SMART applications. The SMART architecture specification has been influenced by several trends and state-of-the-art developments in different areas including search engines, reasoning engines and the internet-of-things. The consortium has taken into account these trends in order to ensure that SMART is developed according to known principles and best practices, while at the same time advancing the relevant state-of-the-art. At the same time the introduction of the SMART engine has taken into account the set of stakeholders' requirements, which have been documented in deliverable D2.1 of the project. Note that the SMART architecture facilitates to a larger (and sometime lesser) extend the implementation of all key requirements expressed by stakeholders. The present deliverable illustrates the project's influences from the state-of-the-art, but also the impact of the requirements process in several key architecture decisions.

In terms of the actual presentation of the architecture, the deliverable introduces the main modules of the architecture, including the edge node, the social networks manager, the search engine (based on Terrier.org) and the SMART application modules. Each of these modules is presented in the scope of the overall SMART architecture, and later detailed in terms of its main functionalities and interfaces to other components. As part of the presentation of the anatomy of these modules, the deliverable introduces a number of important submodules that play an instrumental role in the operation of SMART compliant systems. For example, the edge node comprises the reasoning engine (allowing the extraction of composite environment-derived context on the basis of the combination of individual streams), the media data manager (enabling the storage and management of multimedia streams at the level of the edge node), as well as the manager of linked data (which enables access to rich metadata available at the Linked Data cloud). Similarly, the search layer of the system includes accumulation and merging components, as well as search brokers. In the scope of the deliverables we briefly outline the purpose and main specification of these components. The specification, design and implementation of their detailed functionalities is beyond the scope of this deliverable and will be conducted within the individual workpackages where these components are developed.

Along with the presentation of the SMART architecture, the present deliverable illustrates some concrete examples associated with the use of the main components of the architecture. These examples provide further insights towards understanding the role of the various modules in the scope of the SMART architecture, as well as their interrelationships and interlinking in the scope of practical applications (such as the SMART live news and security/surveillance use cases).

Note that the SMART architecture is an open architecture, since it allows the incorporation of new sensors and data feeds, while providing open interfaces for accessing the capabilities of the search engine. Furthermore, an open source implementation of the architecture is in progress.

Overall, the SMART architecture boosts openness and modularity, which are expected to facilitate the integration and the extensibility of the SMART systems. At the same time the architecture will facilitate the organization and management of the development work, including the work foreseen as part of the SMART open source project.

1.4 Structure

The deliverable is structured as follows:

- Section 2 illustrates the background projects that influenced the design of the SMART architecture, while it also illustrates how the SMART architecture has taken into account requirements expressed in the scope of earlier WP2 deliverables.
- Section 3 introduces the SMART architecture and its main layers.
- Section 4 is devoted to the presentation of the edge node layer of the SMART architecture, including a high-level presentation of all the modules that comprise this layer.
- Section 5 is devoted to the presentation of the Social Network Manager, a dedicated module of the SMART architecture, which permits the interfacing of the SMART system with social networks for the purpose of consuming information stemming from social networking platforms.
- Section 6 focuses on a high level overview of the search layer of the SMART system, including its interfaces to other layer.
- Section 7 illustrates the role and positioning of SMART applications in the SMART architectures.
- Section 8 concludes the deliverable.

2 Drivers of the SMART Architecture and Technical Specifications

The SMART architecture is driven by a number of factors including:

- **Stakeholder's requirements as illustrated in deliverable D2.1**: The SMART architecture attempts to respond to most of the stakeholders requirements expressed as part of deliverable D2.1. Several of these requirements (including requirements for openness, modularity and extensibility) are addressed by the SMART architecture. Note however that some of the expressed requirements will be also addressed at a component-level (i.e. that as part of the design/implementation of specific components)
- **State-of-the-art developments**: The SMART architecture has been influenced by state-of-the-art development in several areas including internet-of-things platforms, search engines and reasoning engines. In some case SMART will be (re)using readily available components and architecture designs in these areas, as also reflected in the SMART architecture.

Following paragraphs illustrate how the SMART architecture has been influenced by the above factors, including a mapping of key requirements to architecture considerations and architecture development guidelines.

2.1 Mapping of Requirements to Architectural Considerations and Technical Specifications

The following table illustrates how some of the SMART stakeholders' requirements (listed in deliverable D2.1) have driven the development of the SMART architecture. As already outlined, several of the requirements listed in D2.1 are addressed by individual components rather than by the SMART architecture as a whole. These requirements are not listed on the table.

Code	Requirements Description	Impact on Architecture Development
R1.1.1	SMART should reuse existing open source software and tools, where it is appropriate and possible according the license.	The SMART architecture should be based on the modular composition of the existing (i.e. legacy) components and new ones (i.e. to be developed in the project). In terms of background open source components the SMART architecture leverages: (a) The Terrier Search Engine for Indexing and Retrieval Functionalities, (b) Reasoning engines (such as the Alchemy engine) for rule based reasoning and statistical learning.
R1.1.2	The architecture of SMART must be layered, providing separation of concerns	The SMART architecture is layered. The following main layers are defined: (a) Edge Node acting a proxy to physical and virtual sensors, (b) Search Engine, implementing multimedia indexing and retrieval over multiple geographically and administratively distributed edge nodes and (c) Applications layers including SMART applications that execute queries over the search engine. In addition to these layers, the SMART architecture boosts the separation of concerns

		through modularizing the way the search engine interfaces to social networks (through a Social Networks Manager module), as well as the way reasoning over multiple multimedia sensor streams are performed.
R1.1.4	The components should be developed using proven and trusted languages. The open components should use a language with good open source community support.	The Java language has been selected as the main implementation language for SMART. Java has strong support from the open source community. The SMART architecture boosts the use of this language, on the basis of the reuse of Java-based background components (such as Terrier) and the prescription of interfaces that could be implemented based on the Java language.
R1.2.3	Video processing algorithms in SMART should be real-time (there is no storing of data, unless a specific event is detected). The minimum frame rate for significant inter-frame correlation is 10 FPS, hence the algorithms need to process frames this fast.	The perceptual components in the edge node are designed to receive input per request (bring the next available frame, now that processing is done) or by streaming (in which case a frame is dropped if processing is not already finished). The SMART architecture can accommodate real-time data feeds stemming for video processing algorithms.
R1.4.5	SMART system should support a wide set of non-AV sensors.	Edge node as designed should support multiple concurrent data feeds from different type of sensors (temperature, humidity, etc).
R1.4.6	SMART sensors should adhere to the SensorML framework.	Data streams representation formats should adhere to SensorML as well as other similar standards.
R1.4.7	SMART should support for multiple social networks and for filters over social networks.	At least the various large social netowrks such as Facebook and Twitter should have flexible filters for acquiring the relevant information.
R1.5.1	SMART access widgets should be designed using languages usually adopted by the open-source community. To extend the use of the platform to the mobile handsets, HTML5 could be a good approach to SMART widgets programming.	The SMART mash-ups and widgets will be developed in order to allow easily created and customized widgets that will increase the usability of the overall system and the search results.
R1.5.4	Data captured by the SMART sensors should be able to be contributed to the Linked Data framework.	The data captured in SMART will be able to be contributed to LinkedData framework and similar RDF representations.
R1.6.1	The SMART search engine should produce rankings of events, possibly in response to a user's query.	SMART should define a search layer, with an appropriate API that allows applications to issue queries and retrieve a ranked list of events as a response.
R1.6.2	The search engine gathers new input from sensors via edge nodes, and must be robust to communication failures with edge nodes.	The SMART architecture should define a unified API between the search layer and the edge nodes. The search layer should tolerate data communication errors so that it does not corrupt the index.
R1.6.3	Input from sensors via the edge nodes should be used to model the importance of these events. The current data from a sensor may be compared to the previous background history of that sensor.	The perceptual components output continuous metadata and low-level events. These are fused by the reasoning module to get the important high-level events. Low-level data from sensors via the edge nodes

		should be represented in the index of the search layer so that the background information about the sensors can be easily obtained when ranking events for a search query.
R1.6.4	As recent events are likely to be more important, the search engine should regularly index new data arriving from edge nodes.	The search layer subscribes to streams of events from the different edge nodes. The architecture of the SMART search layer should make it possible to index in real-time updates from the various edge nodes and also retrieve events in real-time. As soon as an update arrives from an edge node it will be indexed and made available for search.
R1.7.1	The Media Data component will be able to present multimedia information to the users.	The Media Data Manager (MDM) will present multi-media information to the users about relevant events.
R1.7.2	The video information will be adapted to the most common types of terminals.	The MDM generates different video files adapted to main types of terminals (client devices)
R1.7.3	The Media Data component will be able to store video for a predefined time.	The MDM stores all data for a short space and relevant contents for an indefinite amount of time.
R1.7.4	Media Data component should respect the legal and ethical aspects.	The MDM only store data from approved sources.

Table 1: Addressing the SMART Requirements within the SMART Architecture

2.2 Influences from the state-of-the-art

In addition to taking into account the requirements illustrated in the previous paragraphs, the SMART architecture has been influenced from the state-of-the-art in various scientific areas including search engines, information fusion architectures and the internet-of-things (IoT). In particular, recent IoT developments have influenced SMART in the way it interacts with sensors and other internet-connected devices, given that the SMART search systems need to interface, filter and process physical and virtual streams. At the same time, the state-of-the-art in search engines (including indexing and retrieval) provided a basis for designing the architecture of the SMART search components (i.e. components searching the sensor repositories). Finally, the SMART architecture has taken into account on-going developments in information fusion, with a view to capitalizing and extending the vast amounts of existing works in this area. Following paragraphs elaborate on these influences.

2.2.1 Internet-of-Things Platforms

SMART bears several similarities to IoT systems, mainly due to the fact that the SMART search engine searches information derived from physical sensors (such as microphones and cameras). Note that SMART focuses (mainly) on the consumption of sensors information and does not include the notion of actuation (e.g., the ability to control sensors via the SMART system). Furthermore, it does not address sensor-to-sensor interactions in the scope of SMART applications, which is a radical difference from most IoT applications. However, SMART's need to access, filter, process and combine sensor-derived information makes it very pertinent to IoT system. Therefore, SMART has carefully reviewed IoT systems and platforms, especially in terms of their ability to acquire and process information from heterogeneous sensors.

SMART is aligned to the general structure of IoT systems, in terms of sensor information processing and acquisition. In particular, SMART aligns to the lower layers of the Architecture Reference (ARM) model of the FP7 IOT-A[1]project, which is developed and tested in the scope of the Cluster of European Projects on the In-

[1]Internet-of-Things Architecture Project (www.iot-a.eu)

ternet-of-Things (IERC)[2]. Note that SMART takes into account only the sensor information acquisition and processing of the ARM, since it is not a fully fledged IoT system (i.e. it does not deal with several QoS and management aspects, as well as IoT service composition aspects).

As a result of its open nature, SMART has also considered the architectures and operations of popular online platform for sensor data feeds processing and management. Prominent examples of such platforms include the Things Speak[3] and Cosm[4] platforms. These platforms provide flexibility and extensibility in the definition and use of data feeds that stem from physical sensors (i.e. physical devices), as well as virtual sensors (such as social networks).This flexibility is very relevant to SMART's openness, since it can serve the project's ambition to facilitate the addition of new data feeds in order to make them searchable by the SMART search engine. Therefore, SMART has reviewed the techniques used for the definition and management of such streams, in order to endow the SMART architecture with the relevant flexibility in adding, managing and exploiting data stream (including data streams stemming from multimedia processing notably video and audio processing algorithms). Hence, SMART has adopted similar techniques for feeds addition and management, on the basis of a common specification (for «virtualized» data feeds), which can be used to describe and annotate data feeds regardless of their originating (physical or virtual) sensor. Note however, that SMART has substantially augmented the semantics of its streams, comparing to the relatively simple (name/value pairs) semantics of these systems [Albakour12].

IoT platforms have also influenced the data modeling choices in SMART. Specifically, multimedia oriented modeling techniques (such as MPEG-7), have been sacrificed for the adoption of a virtualized solution that could be applicable to a wide range of sensor data feeds including virtual sensors (such as data feeds stemming from social networks). Overall, the lower part of the SMART system architecture as presented in following paragraphs (i.e. the Edge Node components) have been substantially influenced by information acquisition and filtering, as well as virtual sensor techniques that are widely used in leading edge IoT platforms.

2.2.2 Information Fusion and Reasoning Architectures

The use of Information Fusion techniques (also known as Information Integration) regards the integration of multiple heterogeneous data from sensors and knowledge into meaningful information for the end users (humans as well as applications).

Information Fusion is meant to take place in the SMART edge nodes which is the interface of the SMART system to the physical world of sensor and social networks, as well as the conceptual world of the linked data cloud. Accordingly to the uses cases and the overall goals of the project, the SMART edge nodes have to process continuously flowing data from a large number of geographically distributed and heterogeneous sources at unpredictable rate to timely produce new data streams to answer complex queries.

Instead of traditional Data Based-Management Systems (DBMS) designed to work on persistent data, or Data Stream Management Systems (DSMS) executing standing queries based on common SQL operators where little semantics is attached to the data being processed, the kind of Information Fusion systems requested for SMART rather belong to Complex Event Processing (CEP) class of systems where, given a precise semantics to the information items being processed, events or situations of the real world are detected and combined into understandable composite information or high-level information using complex patterns. Since these high-level information are meant to be notified to users, CEP systems can be seen as an extension to traditional publish-subscribe, which allow subscribers to express their interest in composite information. For a review on Complex Event Processing, see [Cugola11].

In SMART, the detection of high-level information shall be performed by the so-called "Intelligent Fusion-Manager" (IFM). The Intelligent Fusion Manager is meant to (i) recognize high-level information from sensors data via some recognition patterns (ii) learn those meaningful patterns. To do so, different techniques can be used. Possible paradigms include symbolic approaches such as logic-based systems as well as connectivist approaches such as Neural Networks, but since explicit and symbolic linked data from the knowledge base

[2]http://www.internet-of-things-research.eu/

[3]https://www.thingspeak.com/

[4]https://cosm.com/

are intended to be used to recognise high-level information, we shall only consider symbolic techniques to build our Intelligent Fusion Manager. Furthermore, potential use cases involving some sort of regulation also favour the use of a symbolic approach.

Accordingly, the recognition of high-level information from sensors data via some patterns will be performed through inferences using some sets of symbolic rules. These rules shall be either given by a human user (for example an administrator or an expert) or learned machine learning techniques. Many techniques and tools exist to perform rule-based inference and rule learning, and new ones are regularly invented. For this reason, instead of committing to a particular engine, we shall consider the Intelligent Fusion Manager as a container in which different engines can be settled. Each engine shall then be invoked using a common set of APIs.

2.2.3 Search Engines

In general, the SMART search engine layer is inspired by much research in information retrieval and search engines in the last decade. In particular, we are building upon a state-of-the-art information retrieval and search engine platform, namely Terrier, which implements many of the currently accepted indexing and ranking approaches for large numbers of documents, as well as ubiquitous data structures, such as an inverted index [BaezaYates2011]. This ensures the efficient processing of the representation of the events that the search engine will be ranking.

In particular, the ranking of events builds upon recent developments in information-theoretic models for information retrieval, such as the Divergence from Randomness framework [Amati03], of which Terrier implements several models [Ounis06]. Such models define the notion of informative content, which measure the amount of useful content that matches a given query using statistics from the corpus. These models are being adapted to identify useful interesting events by comparison with previous observations for that type of sensor at and across edge nodes over time. In addition, the start and end of events are modelled using state-of-the-art change-point detection algorithms [Killick2011].

Queries to search engines (e.g. web search engines) are increasingly related to current events, and hence the information presented to the users should be adapted with such freshness in mind [Hansell2008]. Similarly in SMART, the search engine will ensure that the most recent interesting events are presented where appropriate, depending on the query, the user and the application.

Parallels between the SMART search layer and web search engines can also be drawn with respect to the continuous collection of data from edge nodes. In particular, the crawler component of a web search engine [BaezaYates2011] assumes that web sites can be unreliable, and hence are collected in an *offline* mode, before being added to the search engine's index. Many websites are crawled concurrently by a crawler. Then, when a query arrives, the search engine can still provide results that address a website that is currently unavailable. The SMART search engine is also robust to transient network conditions when communicating with edge nodes, such that an inability to contact an edge node should not prevent search results from being retrieved, nor updates from other edge nodes from being indexed.

User behaviour and interaction data form an important source of evidence for modern web search engines. For instance, implicit signals of relevance feedback can be obtained by monitoring the click patterns of users, while common query reformulations can also be gathered. Taking into account such signals can improve the effectiveness of the search engines [Joachims02], [Santos11].

Finally, similar to the internal architecture of web search engines [Dean09], the SMART search engine should be based on a distributed architecture, such that it is efficient by parallelising the ranking of search results across many *query servers* before appropriately merging into a final ranking list by a *broker* node.

3 Overview of the SMART Architecture

3.1 High-Level Architecture

The SMART system was design as layered system consisting of three main layers, namely:

- **A layer of sensor edge servers**, which ensures the interfacing of the SMART with data stemming from the real world. The sensor edge server undertakes the tasks of acquiring physical world data via sensors (notably cameras and microphone) and accordingly structuring data based on standard formats (such as XML, RDF/Linked Data) to allow the upper layers to consume these data.
- **A search layer** comprising a customized version of the Terrier search engine, adapted to the SMART project's needs. This layer indexes and retrieves data from the layer of edge servers. At the same time it retrieves, parses, analyzes queries and uses them for the selection (or prioritization) of appropriate edge servers where data of interest can be found. It can also support event-driven queries («pull» mode), in which case the search engine subscribes to events and updates the upper layer upon the occurrence of these events (i.e. recurring queries).
- **A layer of end-user applications** (including text-based interactive queries), which submit queries to the search engine and visualize the results. In addition to interactive-text based queries, this layer supports the Web2.0 compliant visualization of information stemming from the physical world (according to a mash-up concept).

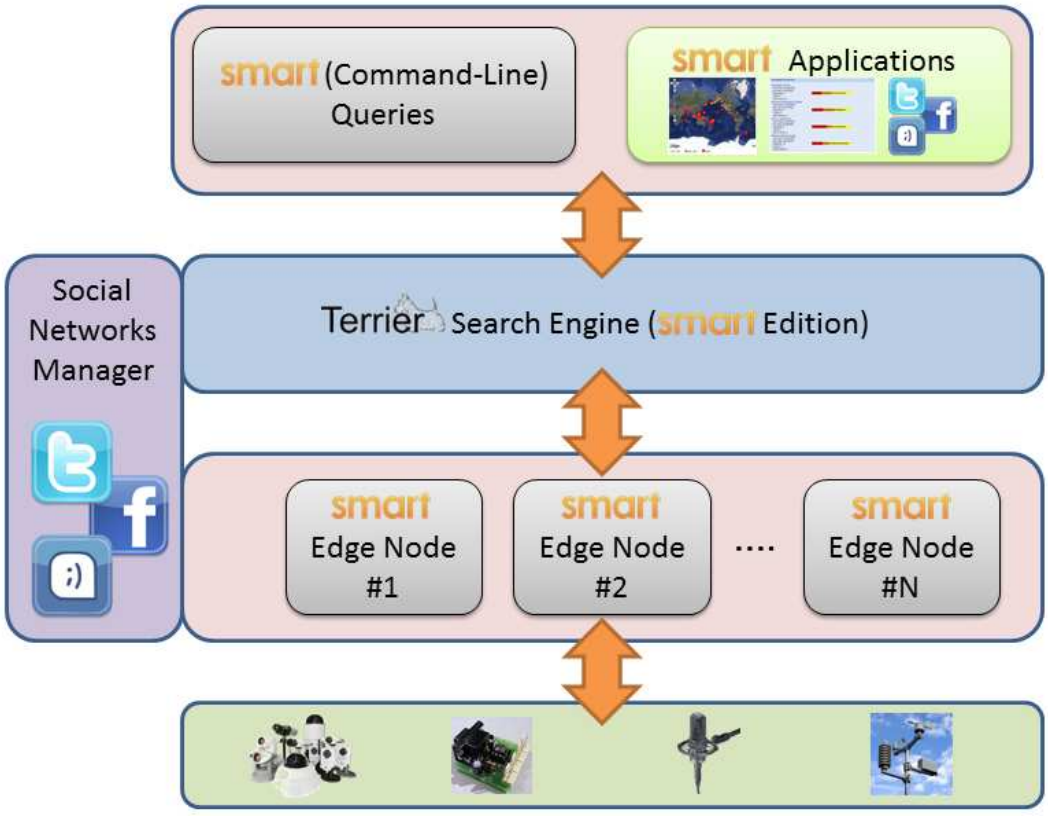

Figure 1: High-Level Overview of the SMART Architecture

The three main layers are depicted inFigure 1. SMART promises also to augment query results on the basis of information stemming from the social networks, while also using social networks information (such as information stemming from facebook.com, twitter.com, etc.) in order to personalize the query results.To this end, we envisage a **Social Networks Manager** (SNM) component, which will provide to the SMART engine

the means for interacting with social networks. Note that the SNM interfaces/interacts with both the search layer and the edge node layer. The interfacing with the edge node layer ensures that the SMART system can process and consume data stemming for social networks, thereby allowing sensor networks and sensor networks processing algorithms (e.g., sentiment analysis [Agarwal11], event detection [Becker09], gender analysis) to be included in the SMART system. At the same time, the interfacing with the search layers allows the search engine to acquire personalized information for specific users on the basis of their personal social networks' accounts and associated social graphs. Following paragraphs provide more information about the various components of the SMART system / platform, while later sections elaborate on their architecture.

3.2 Overview of the SMART Edge Node

The edge node is the interface of the SMART system with the physical world. To do so it offers the following functionalities:

- Interfaces to sensors via the sensor drivers: The edge node is fed with information stemming from physical sensors such as temperature sensors and parking sensors. The integration of these sensors in the edge node hinges on the availability of one driver for each sensor, which undertakes to convert the sensor data feeds into formats that enable their processing by the edge node.
- Processing of metadata streams that stem from audio-visual signal processing algorithms, as well as from algorithms processing social network streams: In addition to physical sensor streams, the edge node will process streams of metadata that are derived from perceptual processing algorithms (i.e. algorithms processing multimedia streams), as well as metadata streams stemming from social networks processing algorithms. Note that the algorithms that enable the processing of information from social networks, can be characterized as virtual sensors (which are also called social sensors in the context of the SMART project).In order to manage social sensors, the SMART architecture introduces the Social Network Manager component which is described in the scope of a subsequent section.
- Intelligent fusion of metadata in an intelligent way, notably in a way that reasons for events: Upon the reception of data feeds from physical sensors, virtual sensors and social networks. The intelligent fusion of the various streams results in the identification of events and other contextual information about the environment where the edge node is deployed. The results of the fusion of two or more streams of the edge node can be modeled on the basis of additional streams.
- Delivers information to higher layers on the basis of standard file formats: The full range of information produced in the edge node (including elementary/atomic streams of sensor data and metadata, as well as composite information (streams, events) produced through the information fusion process) are made available to the higher layers of the system, notably the search engine which can index, retrieve and process information across many different heterogeneous nodes. To this end, the edge node produces (periodically) files that comprise metadata pertaining to the various elementary and composite streams. The files follow standardized formats (XML/RDF), which facilitates their parsing/processing from the search layer of the SMART system.
- Provides a range of configuration capabilities concerning individual sensors (physical, virtual and social), as well as the reasoning rules that drive their combination into composite sensors. The latter rules can be either explicitly specified (i.e. by the entity deploying the edge node) or automatically identified by the edge node on the basis of statistical learning processes implemented at the reasoning component of the edge node. Note that the configuration capabilities of the platform target also the frequency of the production of files.

Figure 2 illustrates the components of the edge node that provide the above functionalities. In particular it shows the sensor drivers that enable the adaptation of physical (and virtual) sensors, as well as the perceptual components which process audio-visual data thereby being a critical element of the multimedia search engine. Furthermore, the intelligent fusion manager is depicted, as the software entity that receives and combines the various data feeds from the underlying sensors and perceptual components. The information that is produced by the Intelligent Fusion Manager serves as a basis for producing file collections, which are (at a later stage) indexed and processes by the Terrier search engine (which belongs at the search layer of the SMART system). Finally, the figure illustrates the edge node configuration manager component, which is in charge of configuring and managing the main parameters of the edge node, including parameters regulat-

ing the fusion of information and the collection of data from the underlying sensors.

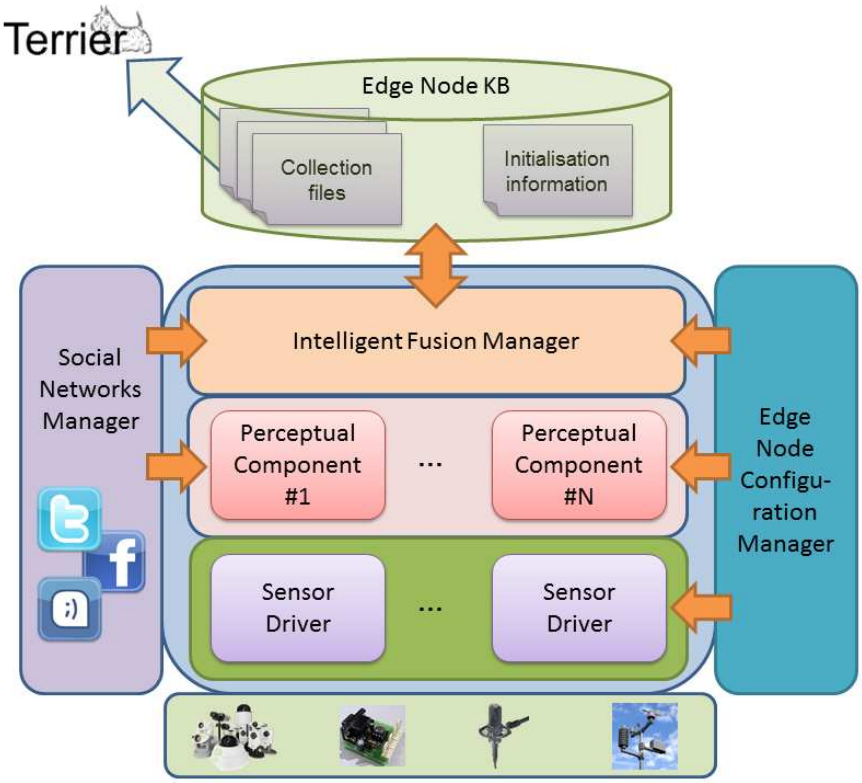

Figure 2: Anatomy of the SMART Edge Node

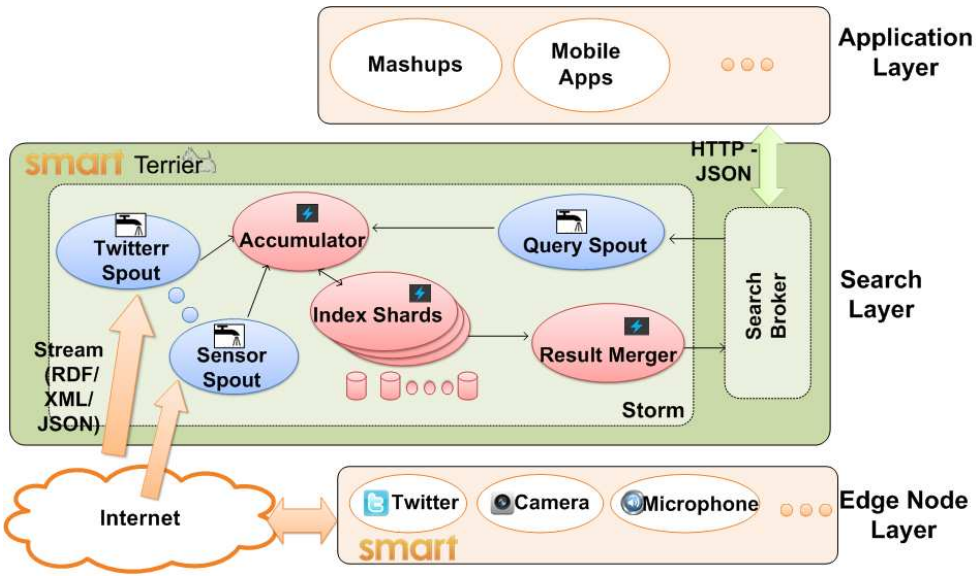

Figure 3: Anatomy of the SMART Search Layer

3.3 Overview of the Search Layer

While the edge node deals with context acquisition (typically context relating to events occurring within the vicinity of the (physical) sensors deployed at the edge node), the search engine is a higher level entity in charge of indexing, access and analyzing content from multiple heterogeneous and geographically distributed edge nodes.Figure 3 illustrates some of the components of the search layer of the systems including its interfaces to the applications and edge node layers. These interfaces are prescribed based on web services, which will enable the transferring of XML based files from the edge node to the search layer, but also from the search layer to the applications.

Inside the search layer, several functionalities are specified, including the accumulation of information stemming from the various edge nodes, and the merging of results towards answering given queries. The process involves mapping the query terms to indexed results stemming from the edge node, in a way similar to the popular Map-Reduce techniques that will lead to the generation of the target results set. The relevant SMART mechanism that will be designed in the scope of the SMART system is conveniently called SmartReduce.

3.4 Overview of the Applications

The SMART applications layer (Figure 4) includes applications that comprise one or more SMART queries (i.e. queries to sensor information through the search layer). A SMART application is therefore a conventional web-based application, which incorporates one or more result sets stemming from the SMART search engine. An application can also embed mash-up components that comprise SMART queries. SMART plans to implement mash-up libraries (comprising commonly used components such as maps, charts and graphs).Such libraries will facilitate the visualization of SMART queries and their incorporation/integration within SMART applications. Similar to all popular search engines, the SMART search engine will also support command-line queries.

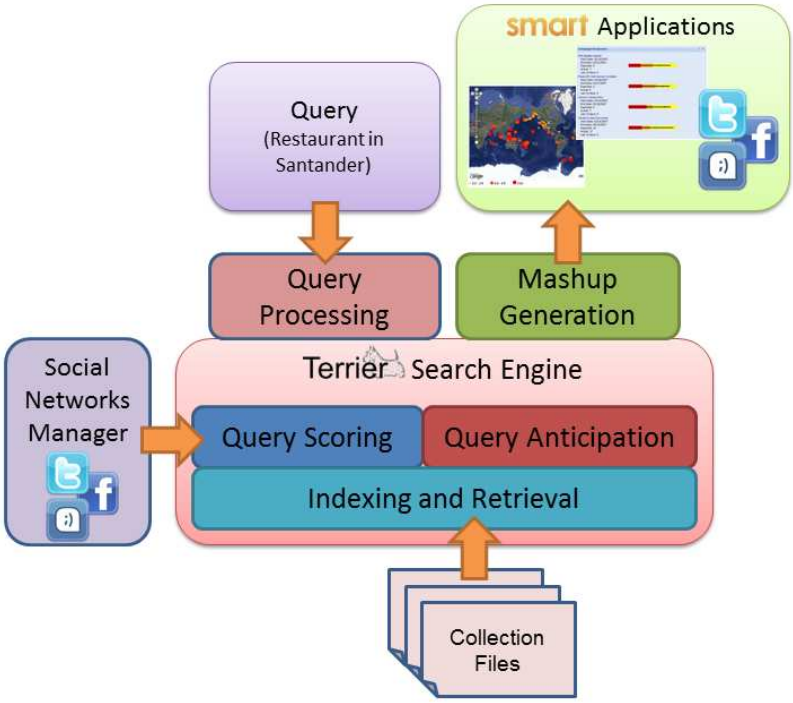

Figure 4: SMART Application Layer

4 Edge Node Architecture

The edge node is the interface of the SMART system to the physical world of sensor and social networks, as well as the conceptual world of the linked data cloud.

The edge node has three stages shown in Figure 5:

- Processing stage: Algorithms operating on sensors, the social network manager operating on social network data and the linked data manager operating on linked data.
- Reporting stage: Formatting the metadata and forwarding them to the database for storage and streaming towards the search engine, and,
- Reasoning stage: Metadata from different streams are fused together to infer high level events.

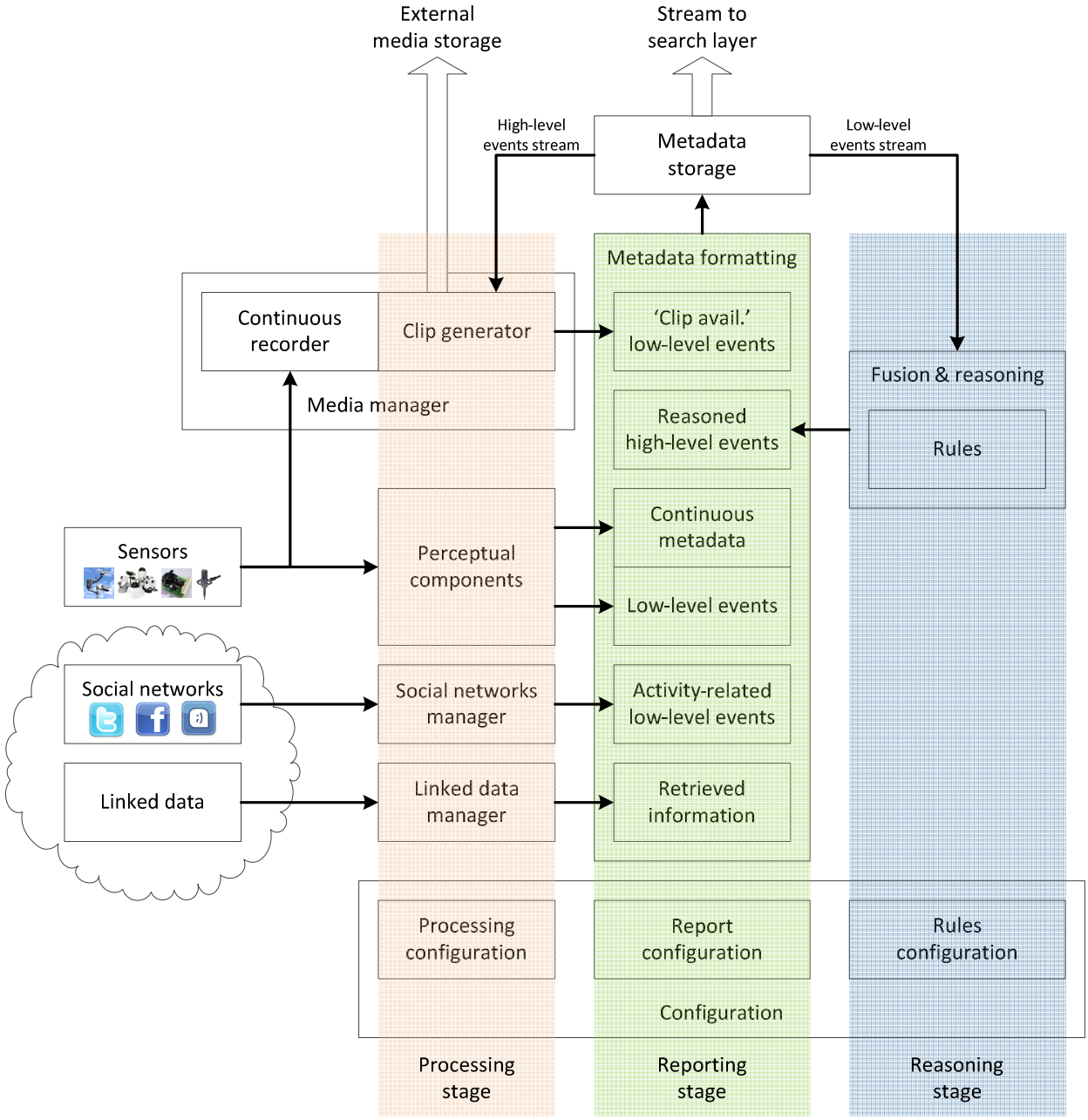

Figure 5: Detailed view of the edge node

The edge node brings to the SMART system information:

- perceived from its physical surroundings as they are sensed by a sensor network,
- filtered from social networks,
- retrieved from the linked data cloud, and,
- inferred by combining all the above from diverse sources.

These types of metadata are detailed in D4.1, “SMART Distributed Knowledge Base and Open Linked Data Mechanisms,” where examples of their content and usage are given.

In order to obtain the perceived information, sensor feeds need to be managed and subsequently processed. While processing is the aim of several of the WP3 deliverables, the architecture for managing the feeds is described in Section 4.1.

Filtered information from social media is obtained by the Social Network Manager (SNM), described in Chapter 5.Note that from an architecture point of view, the Social Networks Manager can serve a single edge node (in which case it is considered another edge node component), or multiple nodes. In the former case, an SNM instance could be bundled within the edge node thereby offering tight integration and better control over the configuration of the SNM instance (i.e. given that the instance will be «dedicated to the edge node). In the latter case, an SNM instance could be hosted within a cloud computing infrastructure and used to serve multiple edge nodes (much in the same way social networking platforms serve multiple users/users). Both of the above options are possible and in general in-line with the SMART architecture specification. The final decision lies in the realm of deployment issues/considerations and will also depend on the deployment choices of the SMART services integrator. The SMART project intends to share experiences from the actual deployment of the SMART systems (as part of the live news and security/surveillance use case), which should be taken into account from potential deployers.

Retrieved information stems from the liked data cloud. The component for managing linked data is not yet mature; only some initial information is given in Section 4.2.

Reasoning is done by fusing perceived, filtered and retrieved information within the Intelligent Fusion Manager, described in Section 4.3. Reasoning, as well as interfacing with the higher search layer, is facilitated by storing the information in the edge node database, summarised in Section 4.4.

Finally, apart from metadata, the edge node will be able to provide images and/or short video clips depicting the inferred events of interest. This is conditional to receiving permission from the Data Protection Agencies. The means for providing media is the Media Data Manager, discussed in Section 4.5.

4.1 Sensor Data Feeds Management

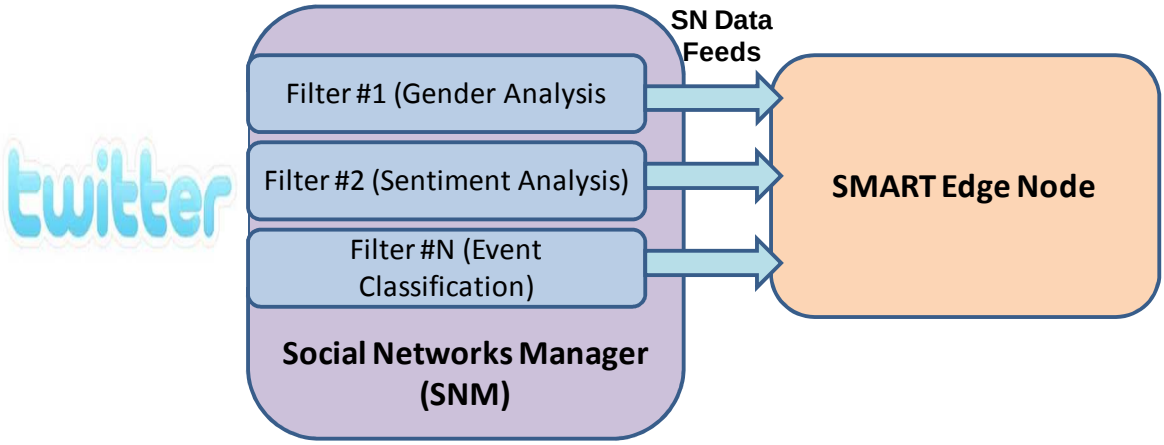

Figure 6: Twitter Filters Examples

The Smart Edge Node will provide the infrastructure needed for collecting data related to the physical world, through a variety of sources. These sources may include both hardware and software feeds that contribute raw data to the system, or software filters responsible for detecting events or activities in both the physical world and Social Media (see for example Figure 6 for an example of handling different streams associated with Twitter processing algorithms). Thus the Smart Edge Node handles any type of data source in an abstract manner, in order to hide heterogeneities and enable data fusion. At this point it must be stressed that a Feed may have more than one type of outputs. In order to add a new feed, one should describe its properties through an XML based description that contains information about:

- The name of the feed.
- The output types of the feed.
- The physical location of the feed (if any).

After the addition of a new feed, the Edge Node assigns it a unique ID number. In order to add measurements to the Edge Node, data feeds should send the measurement data encoded in a simple key-value pair format, through HTTP Put requests. To support the needs such as the addition or data retrieval of feeds as well as the measurements they produce, the SMART Edge Node provides a collection of API Calls to handle the following topics:

- Adding Feeds
  The API should include a call in order to handle the addition of new data feeds regardless of their type. To do so, this call should provide the means to transmit and store XML based descriptions of the feed to the Edge Node. After the processing of the addition request, the new feed should be assigned a unique ID, which users could retrieve in order to extract data from it in the future.

- Management of the Attached Feed
  The API should also include a set of operations related to the management of the attached feeds. Feed management operations supported by the edge node should allow users or software clients, retrieve a list of the feeds that are currently attached to the system, modify their properties, or permanently delete them. In addition to the previous functionalities the edge node must also incorporate ways for querying data feeds by utilizing their properties such as location, observed features etc. Data feed outputs will be described through unique URLs that will allow changing between several encodings of measurement data, depending the HTTP request that accompanies them (e.g. possible encodings will include JSON, XML, RDF or html widgets used to display values)

- Collecting Measurements
  For the purpose of measurement collection, Data Feeds must send data directly to the Edge Node by utilizing an appropriate API call. In order to avoid problems related to the heterogeneity of the data sources, measurement semantics should be explicitly described through the definition of the output properties of each feed. To relieve developers from the burden of using heavyweight xml encodings, the current implementation prototype of the Edge Node demands measurement data to be encoded in a very simple key-value pair format. However in the future JSON as well as XML based solutions will possibly be considered as well.

- Retrieving/Querying Measurements
  The most important functionalities of the edge node will focus on retrieving/querying measurements. Clients should be able to request the latest measurement from a specific feed, or lists of measurements from a feed, that were added to the edge node during certain time periods. Additionally the Edge Node must also provide functions to allow measurement queries that are based on their type, the location they were created as well as the time window that they were inserted to the system.

4.2 Linked Data Manager

Linked data are retrieved by the SMART system in a distributed manner: each edge node collects linked data related to it. The geographical information of the node and its different names/aliases are used to find relevant data.

The linked data manager will request information from the different linked data repositories, and will subsequently collect it.

4.3 Intelligent Fusion Manager - Sensors Fusion and Reasoning

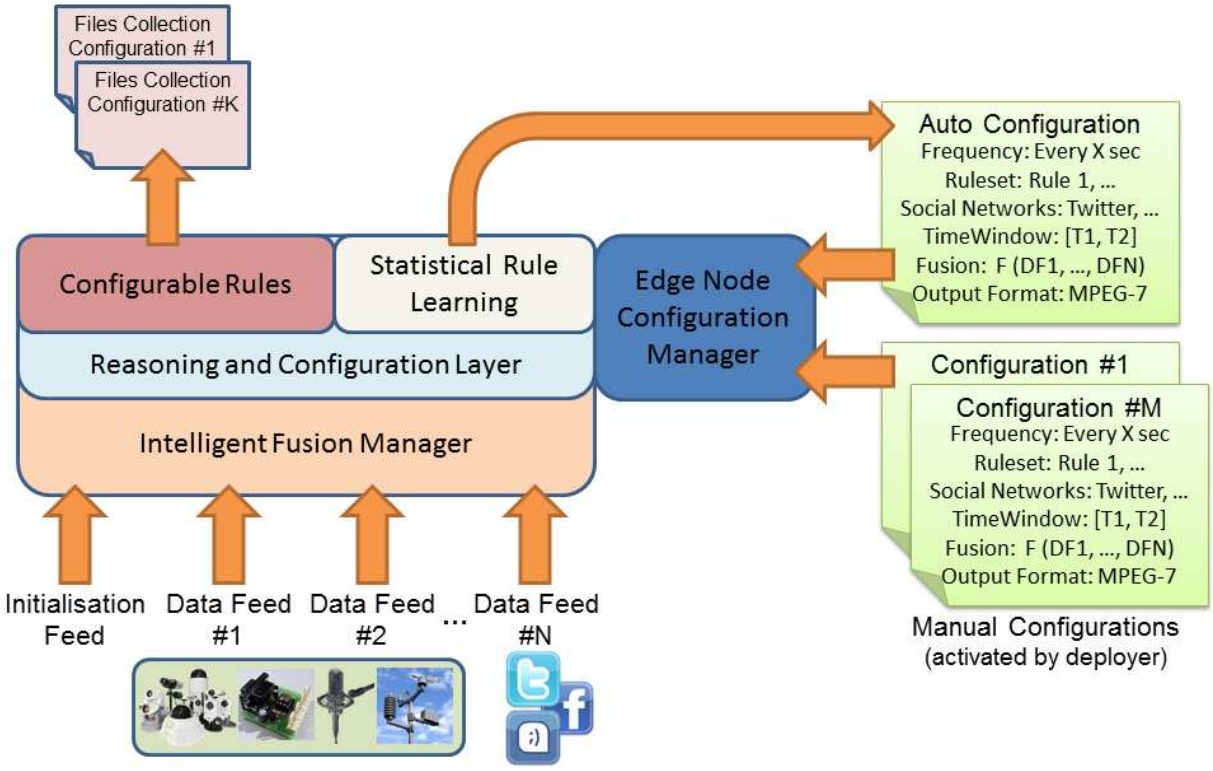

Figure 7: Anatomy and configuration of the SMART Intelligent Fusion Manager

The Intelligent Fusion Manager is meant to (i) recognize high-level information from sensors data via some recognition rule-based patterns (ii) learn those meaningful patterns.

Accordingly, the recognition of high-level information from sensors data via some patterns will be performed through inferences using some sets of rules. These rules shall be either given by a human user (for example an administrator or an expert) or learned machine learning techniques. Many techniques and tools exist to perform rule-based inference and rule learning, and new ones are regularly invented. For this reason and bearing in mind the requirement of openness, instead of committing to a particular technology, we shall consider the Intelligent Fusion Manager as a container in which different engines can be settled. Each engine shall then be invoked using a common set of APIs. Nevertheless, we will propose some engines in particular. In what follows, we quickly comment the APIs, and inference and machine learning techniques used in the Intelligent Fusion Manager.

APIs. With regard to rule-based inference, we shall respect as much as possible the specification Java Rule

Engine API (JSR 94)5. With regard to learning, no API has been committed yet.

The representation format of the rules shall depend on the Intelligent Fusion Information instance, and no common representation format such as RuleML6 is foreseen so far, though such common format shall be investigated in the future. With respect to data formatting, external and internal linked data will be passed to the indexing module to ensure their proper format before being sent to the knowledge base.

Inference. With regard to the inference of high-level information from sensor data, so many rule-based engines exist that we can only propose some possible approaches including:

- Pure rule-based engine oriented for complex event processing such as Drools Fusion[7]: Though Drools does not have a proper defined logic (it is nevertheless based on the Rete algorithm); it is well spread in the business domain due to its performances and its ease of integration with object-oriented programs. This is the reason we consider it.
- Semantic Web reasoning engines such as Jena[8] or SweetRules[9]. Semantic Web engines implements subsets of first-order logic (usually the so-called Description Logic [Krötzsch12]). Pure Semantic Web reasoning engines such OWL engines allow limited use cases such as instance classification, while their association with rule-based reasoning allow more complex reasoning use cases such as inferring properties of some instances from other properties of other instances. Some engines such as SweetRules combine Semantic Web reasoning and rule-based reasoning.
- Event Calculus reasoning engines. Event Calculus engines are based on a logical language (the Event Calculus [Kowalski86]) for representing and reasoning about actions and their temporal effects. A common implementation is based on the reification of this Event Calculus into a logic programming language such as Prolog. However, some other technical variants are possible, and for example, we shall take advantage of some investigation from the FP7 EU project PRONTO (FP7-ICT 231738), namely, the reification of Event Calculus in Markov Logic Networks [Skarlatidis12].

The diversity of approaches and logic languages and thus the multitude of engines raise the issue of interoperability or the combination of these approaches. The Intelligent Fusion Manager may deal with such issue to some extent, but, by default, any engine shall be considered in isolation within the Manager.

Pattern Learning. Like inference engines, many techniques exist with respect to rule-based pattern learning. For our purposes, we shall use learning techniques used in Markov Logic Networks as well as Inductive Logic Programming (ILP).

4.4 Edge Node data storage

The rationale behind the use of a database for storing the metadata is discussed in D4.1, “SMART Distributed Knowledge Base and Open Linked Data Mechanisms.” Since the sources of the metadata are diverse, and we also expect third-party users to be able to connect their sensor/processing pairs to an open SMART system, it is unreasonable to expect that a common structure will exist. For this reason the edge node uses CouchDB, a relatively new no-SQL database.

After adding the feed (see section 4.1) the server receiving the XML feed description generates the database fields. This is the feed registration phase with the database. Following this, information is written to the database by posting a request to send data, either in JSON or XML format. Since the data are stored in JSON, the server possibly reformats timestamps and writes them in the database.

[5] http://www.jcp.org/en/jsr/detail?id=94

[6] http://ruleml.org/

[7] https://www.jboss.org/drools/drools-fusion.html

[8] http://jena.apache.org/

[9] http://sweetrules.projects.semwebcentral.org/

The edge node database is queried for metadata by the use of web services. These allow access in two modes: request for particular temporal windows, or continuous streaming. The implementation of these web services is detailed in D4.1, “SMART Distributed Knowledge Base and Open Linked Data Mechanisms.”

4.5 Media Data Management

4.5.1 Overview

The main mission of the Media Data Management unit is to be able to provide multimedia (audio, video) contents to the users of SMART. The MDM must store the multimedia contents from the multimedia sensors, and adapt the contents to the most common terminals.

One of the main concerns regarding multimedia data storing are the legal limitations and the privacy issues. Storing images for cameras is legally restricted. Personal data or any data that can identify the persons cannot be stored. These restrictions limit the multimedia video sources that we can use for the Media Data Manager:

- Interior sensors showing persons that accept the use of their image.
- Exterior sensors which position prevents the individual identification.
- Multimedia data previously processed in order to avoid identification (for instance, hiding faces or number plates)

In turn, this means that the Smart system can only offer multimedia information for an event if the multimedia information available is according to the legal terms.

4.5.2 The problem of storing multimedia data

A common approach for multimedia search engines is storing everything from every sensor (camera). The main advantage of this approach is that there is no problem at all about losing something important. As everything is stored, we can access to every moment for any camera. If we decide that a moment 3 days ago is important, we can present it to the users even if that moment in time was not consider relevant when it happens.

This approach is taken usually in surveillance applications. All the video is stored, some years ago using equipment as videotapes and video recorders and nowadays using hard disk. There are many commercial products that implement this function.

Unfortunately, the multimedia data take a lot of storage space, specially storing video from a High Definition camera. With 720p or 1024i images, the minimum bitrate necessary is around 6000 bps, even using modern video codecs like h.264. This means that it is necessary around 3 gigabytes of storing per hour per camera, or 75 gigabytes per day and camera.

For storing only a month, a whole modern hard disk of 2 terabytes is needed for each camera. Apart from the direct costs of buying the storage, adding huge storage capacity has a lot of derived management problems, and it is problematic using cheap, common available PC hardware.

This means that storing all the multimedia data would limit the scope of the multimedia search capacity to a narrow margin of time. We'd be able to search anything in the last few days, but we could not find the multimedia information about a very important event that happened a year ago.

The solution is to extract the relevant parts of the multimedia information, and store it indefinitely. The rest of the multimedia contents will be removed from storage space after a reasonable length of time. This way, the storage requirements can be contained in reasonable amount of space.

4.5.3 Mission and design of Media Data Manager (MDM)

As we commented in the section above, the Media Data Manager will store all the multimedia data from the suitable sensors, but only for a sensible length of time. This data is used as an intermediate "buffer" for creating and storing "event clips" relevant events. Using this approach, it can be possible extract relevant data about events from a point before they actually happened (for instance, the seconds before a car accident), and at the same time, the storage requirements can be contained.

We can summarize the operations from the MDM in a serial of steps:

- Storing and converting the multimedia data from the cameras. The data from the cameras will be stored in several data files.
- Creating video clips for relevant events. We need to cut and edit the data files corresponding with the time frame of the event.
- Store and converting the created "event videos". The event videos must be converted to one or more formats suitable for different terminals. An image from the video will be extracted, so we can present a visual clue to the user.
- Creating one or more URL for the stored contents that can be accessed by an application. This URL can be kept with the rest of the searchable document, so the URL will be obtained using the Search Layer. In turn, the application will present this multimedia information to the user.

For a given event, the MDM can generate "event videos" in different formats, for instance, in a format suitable for iPhone, Android or PC. Apart from this, MDM can also generate still images or sound clips, if necessary. This means that MDM will generate several URLs pointing to the different multimedia formats for the same event.

The modules in the MDM will take care of the above steps. The application layer will be responsible of the final presentation to the user. This includes choosing the right video/audio format and in some cases, transmitting it to the terminal. For instance, an application fort the iPhone will choose the URL In most cases, a streaming engine will not be necessary, so the application could take the contents directly from the "event videos" repository.

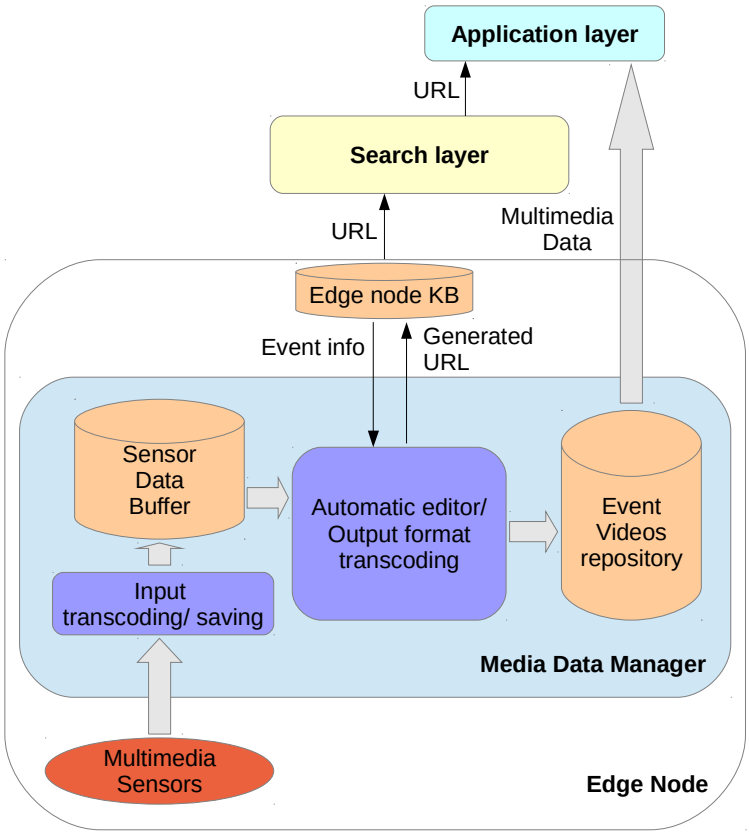

Figure 8 Overview of Media Data Manager

4.5.4 CPU power, bandwidth and data storage

We need to take into account some practical limitations for the deployment. We have talked in the last section about adapting the contents to different formats. This operation usually implies transcoding, and transcoding high quality video content is a process very demanding in computing power. This means that some modules of MDM could need a dedicated CPU. In any case, every transcoding operation weakens the video quality, so only unavoidable transcoding operations must be done.

We told in the section 4.5.2 about the high requirements that multimedia data imposes in the storage. Another related problem is the problem of bandwidth for media broadcasting. In a practical deployment, some Edge Nodes can be connected using only a copper telephone line connection, like an ADSL, so we cannot expect a bandwidth good enough for broadcasting multimedia contents to many users.

The solution to this possible problem is to deploy the MDM storage for "event videos" outside the Edge Serve. This server will actuate as "proxy" for multimedia contents, and it should feature a high bandwidth connection. Storing the elaborated multimedia contents in the "proxy", several users will be able to watch the multimedia contents, without saturating the connection with the Edge Serve. If necessary, the multimedia contents can be stored in a CDN, It is important to take this into account for generating the URL for the contents.

Another practical issue is the storing of MDM application data and multimedia contents. The data will be stored in the NoSQL database engine, the same used for other modules in the Edge Node. It is possible to store the multimedia contents as documents in the database, instead of as files in the local computer. Usually, multimedia applications store the contents using files, but using the database for storing the contents has many advantages, at least in theory. The decision must be taken after some practical evaluation of aspects as the performance.

4.5.5 Interfacing with other functional units

The inputs of the MDM will be:

- Multimedia data from the camera (the video and audio)
- Information about the relevant events (high level events). This information will be generated from the rest of modules of the Edge Node (likely from Intelligent Fusion Manager), according to the rules defined. For the MDM, we need 3 basic parameters.
    - Sensor (camera)
    - Time of begin
    - Length of the event

As outputs:

- Some URLs that will allow accessing to the event videos and generated contents.
- The multimedia contents for the event.

4.6 Summary of interfaces

The interfaces between the different edge node components and the higher layer of SMART, discussed throughout section 4, are summarized as follows:

- **Sensors to processing stage components**: Processing components post a new frame of multimedia/measurements request, or streaming of multimedia/measurements towards the components. The multimedia/measurements request posts are the most frequent, occurring up to 30 times per second.
- **Cloud information (social networks/linked data) to processing stage components**: Processing components post a new data request. These posts are not very frequent, especially for the linked data.
- **Processing stage components to metadata storage**: For initialization the components post a request to register the feed. During metadata collection they post a request to send data. The data send posts are very frequent, possibly occurring multiple times per second.
- **Metadata storage to clip generator** (media data manager): Continuous streaming of high-level events.
- **Metadata storage to reasoning engine**: Continuous streaming of low-level events.
- **Metadata storage to search engine**: Continuous streaming of the multiple metadata feeds.

5. Social Network Manager (SNM) Architecture

5.1 Overview

As stated above the Smart project, aims to combine information stemming from sensor as well as social networks, in order to provide more accurate and useful answers to queries that are related to the physical world. For the extraction and fusion of data from multiple social networks, SMART Edge Nodes incorporate a component known as the 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.

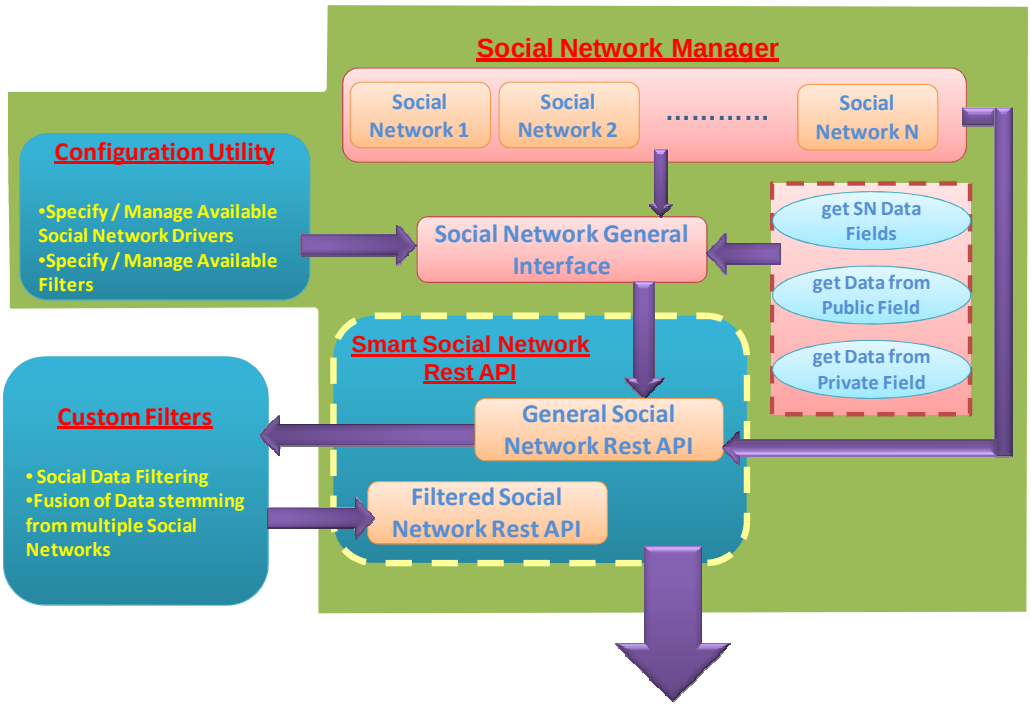

Figure 9: Overview of the SMART Social Network Manager

5.2 Interfacing to 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, may 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 searching 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:

- Extend the system with non-standard functionalities that have not been predicted.
- Extend the system without having to mess 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.

5.3 Configurable Social Networks 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. Data produced by the filters, is accessible though the Filtered Social Network API and is encoded in XML or RDF.

5.4 Design of the Social Network Manager API Calls

General Social Network API:

The General Social Network API focuses on allowing clients of the Social Network Manager perform queries. The rest of this section explains the usage of the calls included in this API:

- Querying all available social networks with one call:

The General Social Network API should allow 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 have to be encoded in both RDF and XML formats. The contents of the results will be limited to posts.

- Querying a certain social network with one call:

Additionally the General Social Network API should allow clients, query a specific social networks 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 have to be encoded in both RDF or XML formats. The contents of the results will be limited to posts.

- Browsing through pages of Results

Since the results from social network searches may be vast in number, a paging mechanism is used to handle load. Therefore the Social Network Manager API will utilize calls that will allow users browse through the retrieved data.

- Using Custom Search Functionalities

Since the functionalities of Social Networks may vary, the Social Network Manager Drivers may be extended using custom search functionalities as explained in section 5.2. The API will provide access to these functionalities through a call that will be used to send xml descriptions of them. Users developing custom search functionalities must specifically encode their results in XML and RDF

Filtered Social Network API:

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

6. Search Layer Architecture

The search layer of the SMART framework is composed of the required software components that provide services and end-users with easy real-time access to information stemming from both the social and sensor media streams. The search layer deals with the representation, storage, organization and access to the information provided by the SMART edge nodes and social media networks.

In particular with references to the requirements identified in Section 2.2, the search layer aims to achieve the following objectives:

- *Real-time* indexing of social and sensors streams (R1.6.2, R1.6.3 & R1.6.4),
- Retrieval and ranking of *interesting* events in response to a user query with associated relevant posts from the social networks (R1.6.1 & R1.6.4),
- Anticipation of user queries depending on their context, e.g. their location or time of the day (R1.6.3),
- Offering of APIs that allow services in the application layer to issue queries and to obtain results and updates in real-time (R1.6.1).

In this section, we describe the architecture of the search layer and identify its main components, the functionalities they provide, their interactions and the information flow between them. We also describe the interfaces of the search layer with the other SMART components.

6.1 Anatomy of the Search Layer

Figure 5 illustrates the logical structure of the search layer where the main software components are identified and the information flows between these components is also illustrated. Most of these components rely on an enhanced and extended version of the open source Terrier platform to deal with the nature of information in SMART.

- **Indexing Component:** The indexing component is responsible for the representation, storage and organization of the information provided by the SMART edge nodes and the social network updates so that they can be later retrieved. It processes the streams of social and sensor updates and performs a real-time indexing of those streams in appropriate data structures allowing efficient retrieval. The real-time indexing ensures that as soon as an update is received it becomes available for search – this is an architectural requirement R1.6.4 as identified in Section 2.2. The index should be *distributed across* multiple index shards (machines) so as to cope with potentially a high number and volume of sensor and social streams.
- **Query Processing Component (QP):** The query processing component identifies the user information needs as specified explicitly by the user or implicitly from the user context e.g. the application, the location of the user, his or her profile or the time of the day. Queries can be anticipated or expanded by observing the patterns in past events (R1.6.3).
- **Matching, Retrieval and Ranking Component (MR&R):** This component is responsible for matching a user query against the index to identify and rank events according to how they satisfy the user information needs. This component will rely on newly developed retrieval models that can identify interesting “unusual” events across different locations covered by the SMART edge nodes from the sensor metadata streams (R1.6.1).
- **Filtering Component:** The filtering component identifies in real-time events (or social network posts) as they happen that match a user’s “running” query. This permits a user to be notified of new events that they will find interesting. This component handles queries after they have been submitted to the search layer so that updates are streamed back to the application layer in real-time as they happen (R1.6.1 & R1.6.4).

- **Data Feed Connectors:** The data feed connectors handle the streams received from the edge nodes and social networks. They parse and extract the raw data received so that it can be properly represented to the indexing component. The data feed connectors are also responsible for smoothing out irregularly timed or missing updates from edge nodes (R1.6.2).
- **Search Logs Component:** This component maintains the search behavior of the user population as indicated by the corresponding application. The search behavior includes the user interactions with the application such as the queries that have been issue in a user search session, the results that have been displayed and whether the user has browsed for further information about the results (e.g. a request to play a media file). The implicit feedback obtained by monitoring users' search behavior can be fed back to improve the effectiveness of the search engine's results.
- **Search API Component:** This component provides an interface to the search layer where the main end-user functionalities are defined and made available to the application layer (R1.6.1).
- **Configuration Component**: The configuration component offers administrative functionalities, as appropriate, to set up the various other components and tune their parameters.

We give an example of the high-level interactions between the different components of the search layer in Figure 9 to give the reader a concrete example of how they communicate in a real scenario.

The architecture of the search layer attempts to achieve the objectives described above and supports the overall architectural aims of the SMART framework by addressing the following key objectives:

- *Openness*: the search layer is developed upon the open source Terrier platform, where the specification of the representation of the index structures is available to the open source community. The search layer is also open in the sense that it can be transparently customized to new social and sensor data feeds. Prospective information providers (including sensor infrastructure providers) are able to connect and contribute new feeds.
- *Modularity and Extendibility*: the search layer is extendible in terms of new ranking models, filtering methods, and query anticipation techniques. This is supported by adhering to the modularity principles in the open source Terrier platform. This objective is based on the architecture requirement R1.1.1 in Section 2.2.
- *Interoperability*: the search layer interfaces with other SMART components using appropriate middle-ware platforms (RESTful services) and data representation formats (e.g. JSON) which makes it easy to deploy with other components of the SMART framework. This objective corresponds to the architecture requirement R1.1.1 in Section 2.2.
- *Fault tolerance*: the search layer processes a large amount and volume of streams in real-time. The architecture should support a fault-tolerant operational mode where the distributed indexing should ensure that the index can be recovered at any time if for some reason and indexing node becomes unavailable. A fault-tolerant operation ensures that we meet the requirement R1.6.2 in Section 2.2.

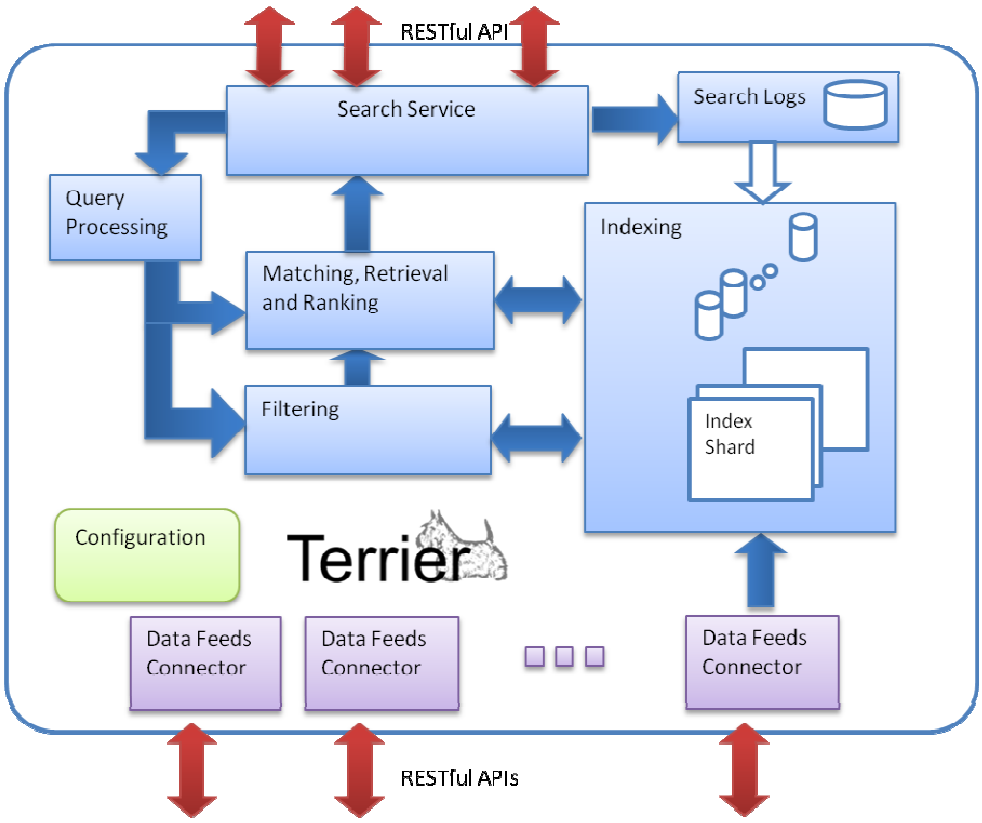

Figure 10: The search layer architecture

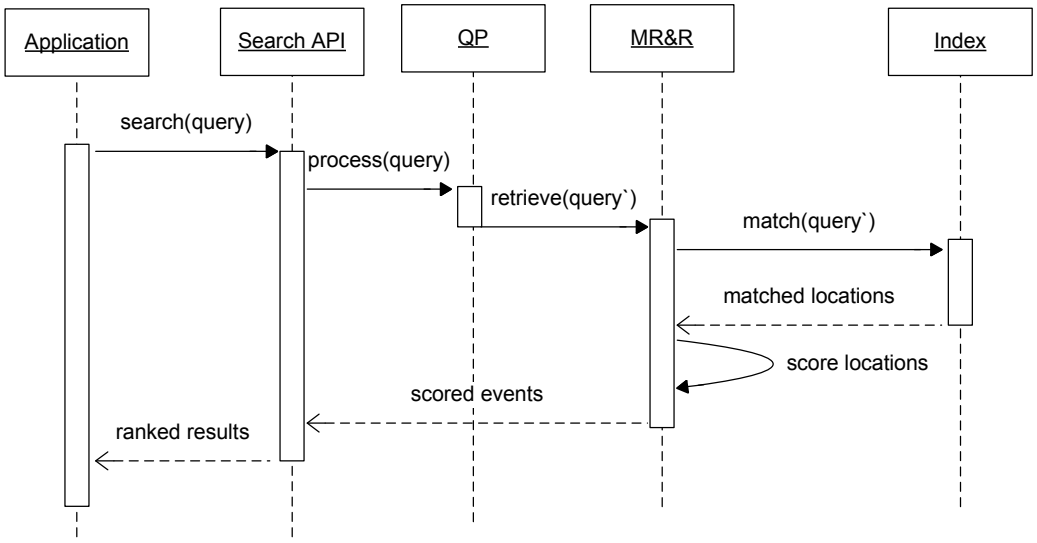

Figure 11: Sequence diagram for issuing a query

6.2 Edge Node to Search Engine Interfacing

As per requirement R1.6.2 in Section 2.2, the search engine interfaces with the SMART edge nodes and social networks to collect in real-time streams of updates, which are then processed by the data feed connectors.

In particular, this interface provides the following functionalities:

- The SMART search engine should receive up-to-date information from sensors at the various locations covered by a given edge node;
- The SMART search engine should receive up-to-date social information from the location area covered by a given edge node, including various social filters;
- The SMART search engine should receive the location of "media snapshots" of sensors at a given point in time from the edge nodes, such that these can be presented within its search results.
- The SMART search engine should be able to connect to edge nodes that have security restrictions.

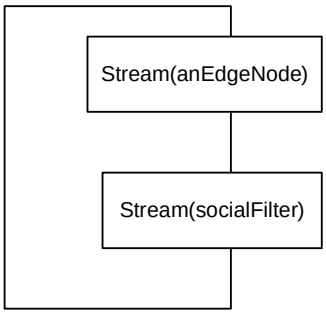

Figure 12: Search layer to edge Node Interface

This interface is realized with a RESTful API such that the search engine can connect to an edge node to receive updates from that node. In particular, the edge node API supports a streaming model, where the search engine receives all updates from the edge node in real-time (see Figure 6). The designed API meets the architectural requirement R1.6.2 as identified in Section 2.2.

In Figure 11, we illustrate the interactions between the search layer and other SMART components when indexing streams originating from the edge nodes or the social networks. As illustrated, the search layer connects to these streams in a pushing manner where new updates continuously pushed to the indexing component.

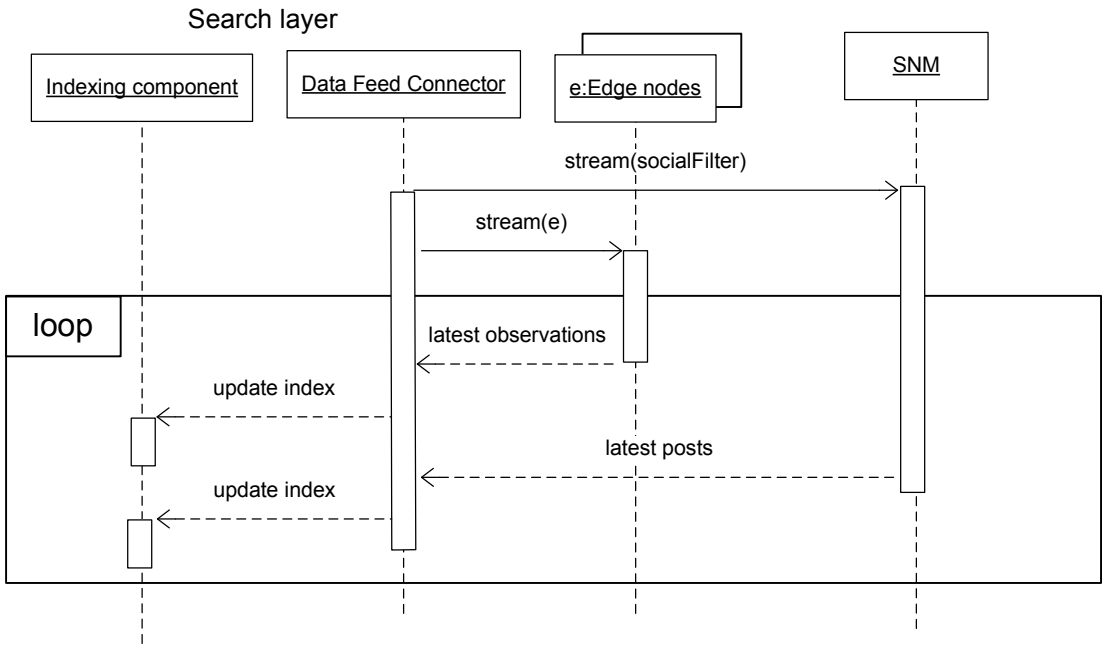

Figure 13: Sequence diagram for indexing streams

6.3 Search Engine to Applications and Mash-ups Interfacing API

As per requirement R1.6.1, the search engine offers an API that allows applications and mash-up libraries to offer services using the search layer's capabilities in providing information access. In particular, there are four primary information flows within the search engine API:

1. Flow of queries from use case/visualisation to the search engine,
2. Flow of results from the search engine to the use case/visualisation.
3. Flow of sensor observations through the search engine, from the edge node location to the use case/visualisation,
4. Flow of user interaction behaviour from the use case/visualisation to the search engine, such that the search results can be improved.

The typical information flow is as follows: a query request is issued by the use case/visualisation (1). This is followed very quickly by results returned from the search engine (2) – in this case, results are pulled by the use case application. On the other hand, a visualisations/use case may also want to be updated when new results match a query. In this case, a *running query* may be registered by the use case application, which the search engine only responds to once updated results have been received. (3) Next, some visualisation methods may require the latest sensor observations for a given location. (4) Finally, the search engine may improve its search results, through further training, based on user interaction behaviour provided by the use case/visualisation to the search engine.

There are three distinct part of the SMART layered architecture that should be considered by this API, as listed below:

- The search engine, which provides results for queries.
- The use case (HTTP) server application. This may, for example, be a PHP or JSP application running on a web server that renders HTML pages to the user.

- The client application accessed by the user. For instance, this may be a JavaScript Web2.0 mash-up based on AJAX/JQuery/HTML5, which accesses the search engine directly for updated results.

In this case, we propose that the requirements of the search engine API be as follows:

- The search engine API will be accessible over HTTP, by browsers or other application user interfaces (e.g. mobile phone apps), as well as directly by the server of the application.
- The search engine will permit rich queries to be specified, either by the user or by the application.
- The search engine API will provide results in a format that can be easily parsed by a JavaScript mash-up application running within a client browser.
- The search engine API will be accessible by applications in such a manner that they can register for update results to be pushed to them over an HTTP connection as soon after they are identified.
- The search engine API will permit the latest sensor observations for a given edge node location to be obtained.
- The search engine API will permit user behaviour data from an application to be transferred to the search engine, such that additional training can be conducted to improve the search results.
- The search engine may be restrictive in who can access the API, or how many calls to the API can be made.

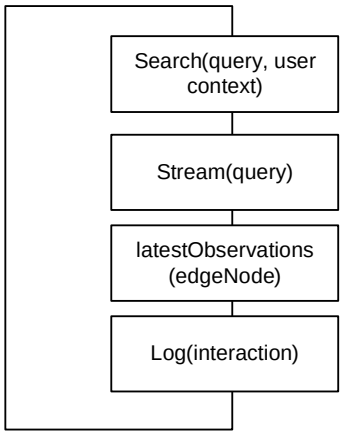

Figure 14: Search layer to applications interface

The Search Engine API aims to achieve the architectural requirement R 1.6.1 as identified in Section 2.2. The API will be realized using a RESTful API, which supports different operations (see Figure 7), namely:

- Submitting a query to the search layer and retrieving back a ranked list of results.
- The streaming of search results for a running query.
- Permitting the latest observation of the each edge node for a given location to be accessed.
- Logging of the user's searching behavior.

7. SMART Applications

7.1 Command-Line Search

All functionalities of the search engine are accessible over a RESTful API. However, there are several objectives that can be serviced through an open source command line client for the search engine:

- Debugging of the search engine: permit the easy viewing of search results without development of complex browser-based applications.
- A trivial example search: permits the use of the search engine API to be seen in its most straightforward form.
- Scripting of SMART-based applications: Having access to the search engine allows unforeseen yet simple server-side applications to be quickly developed, e.g. email alerts based on interesting events occurring.
- Evaluating the search engine in a batch-mode fashion: This permits the effectiveness and efficiency of the search engine to be measured.

The requirements of the command-line search application mirror the functionalities provided by the RESTful search Engine API detailed in Section 6.3 above.

7.2 Mash-up Libraries

Mash-ups are unusual or innovative compositions of content (often from unrelated data sources), made for human (rather than computerized) consumption. Most mash-ups are usually comprised of two different participants that are logically and physically disjoint (they are likely separated by both network and organizational boundaries. These participants include:

- API/content providers
- To facilitate data retrieval, providers often expose their content through Web-protocols such as REST, Web Services, and RSS/Atom (described below).
- Mash-up Logic

Mash-up logic can be implemented using server-side dynamic content generation technologies like Java servlets, PHP or ASP. However mash-up logic may also be implemented using client side techniques and technologies such as AJAX or JavaScript.

Within the SMART project several needs have been identified. Such needs include:

- The presentation of Raw Data from Feed Measurements
- The presentation of Feed Details
- The presentation of Social Network Related Data
- The presentation of Detected events
- The presentation of Reasoning Results
- Fusing Data from Multiple feeds
- Customization

To provide solutions for the aforementioned issues, the SMART platform will include a tool that will enable the Development Customizable Mash-ups, known as Mash-up Builder. Mash-ups will be built, using customizable widgets that will display data from multiple Smart Components such as Feeds, the Social Network Manager, or the Reasoner attached to each edge node. However these mash-ups could also possibly act as information sources that will utilize user controlled data filters in order to produce new useful data, that could be stored back in the system in an recursive manner. The tool will take advantage of a web based GUI, that will let users edit and manage data sources represented as nodes of a graph thus offering an easy way to customize user experience.

7.3 SMART Queries Based Applications

7.3.1 Security Proof of Concept

The security proof of concept is intended to represent a simple realization of security use cases demonstrating how is possible to use SMART data to get information about possible anomalies caused by security problems at public events or in the presence of the crowd.

Moreover, the application is intended to demonstrate how to create query to be submitted to Terrier in a graphical way using simple tools, and provide a small set of API that could be extended to create general purpose smart applications.

Following the list of high level functionalities provided by SMART Common Operational Picture (COP) Application:

- Graphical representation and query building.
- Representation of geolocated sensor networks.
- Representation of geolocated alarms generated by multiple connected systems.
- Representation and data mining on historical data.
- Data integration and Data mining.

SMART COP connects to the SMART search API in order to get different kind of information:

- Status of the sensors (working, not working);
- Type of sensors (audio, camera, gas, etc.);
- Detailed info about sensors (lens, aperture, audio level, etc. etc.);
- Results from posted queries;
- Events: special events occurring when something happens based upon rules.

SMART COP communicates with the server via connectors that uses terrier API. Connectors are intended to hide the SMART API specification offering a high level interface for the SMART COP API. This interface is an abstraction useful to decouple HMI (Human Machine Interface) layer from the communication layer.

In the following we illustrate how an HMI component connects to the server:

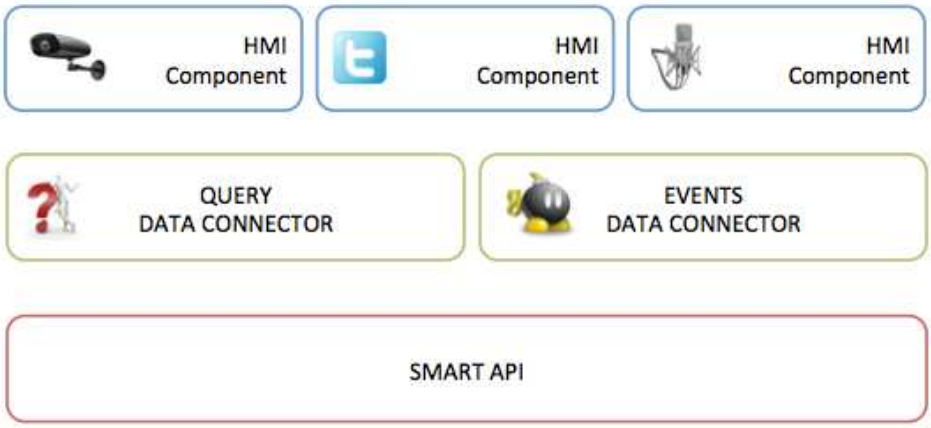

Figure 15: Interfaces of HMI components to the SMART Search Engine

- HMI Component is a simple component that represents a sensor or any kind of data source for the SMART search engine. Each HMI component can be extended to create more complex ones;

- DATA CONNECTOR is an abstraction of the concept of query or event;
- SMART API represents the SMART APIs.

Data Connectors are used to communicate with the server to transfer two kinds of data: query and events. Each component is responsible of:

- Receiving a request from the HMI components
- Packing the request in a SMART API compatible way;
- Sending the request;
- Waiting for response;
- Furnishing to the components a response.

Following sequence diagrams showing how the components communicate each others:

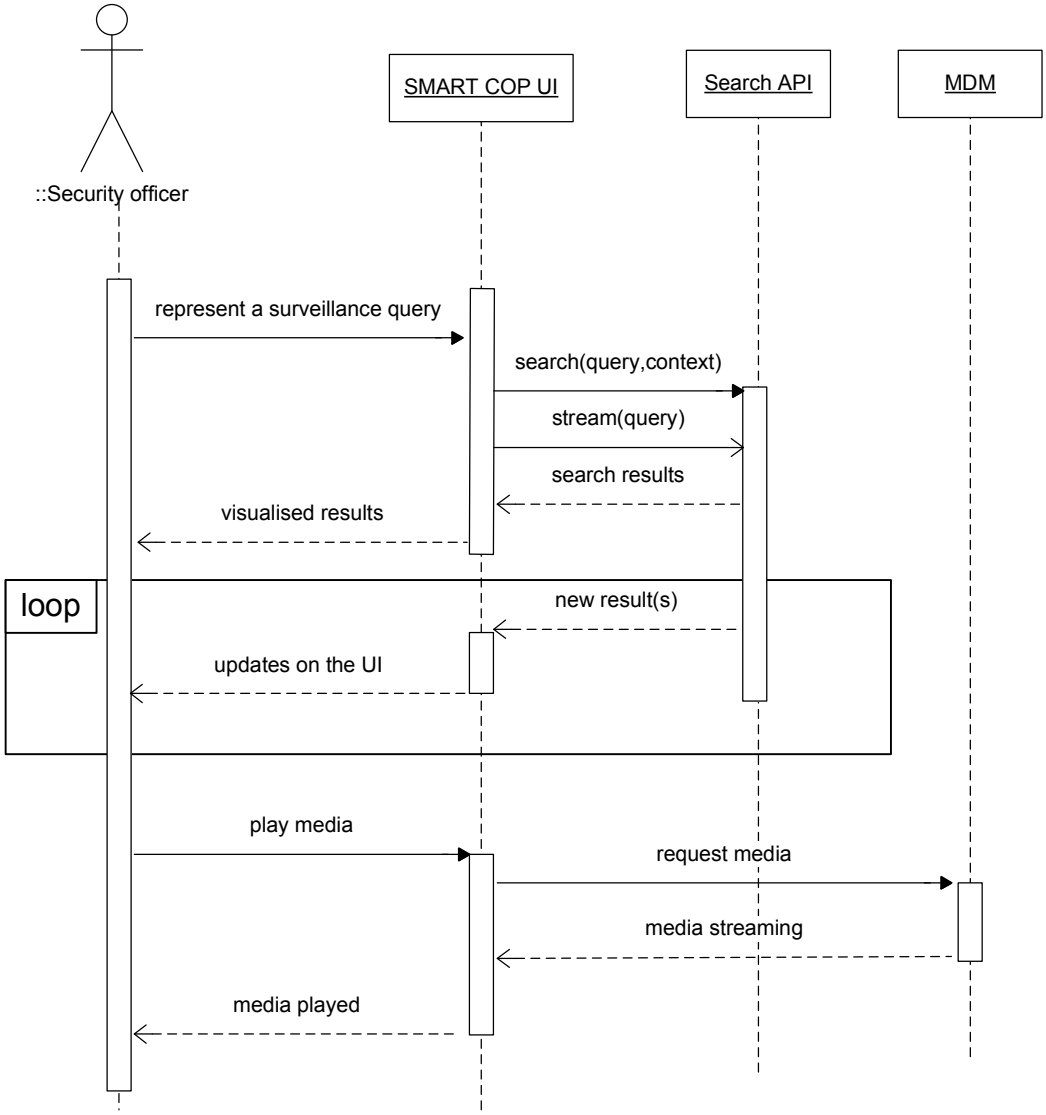

Figure 16: SMART COP applications interactions with the SMART Search Engine

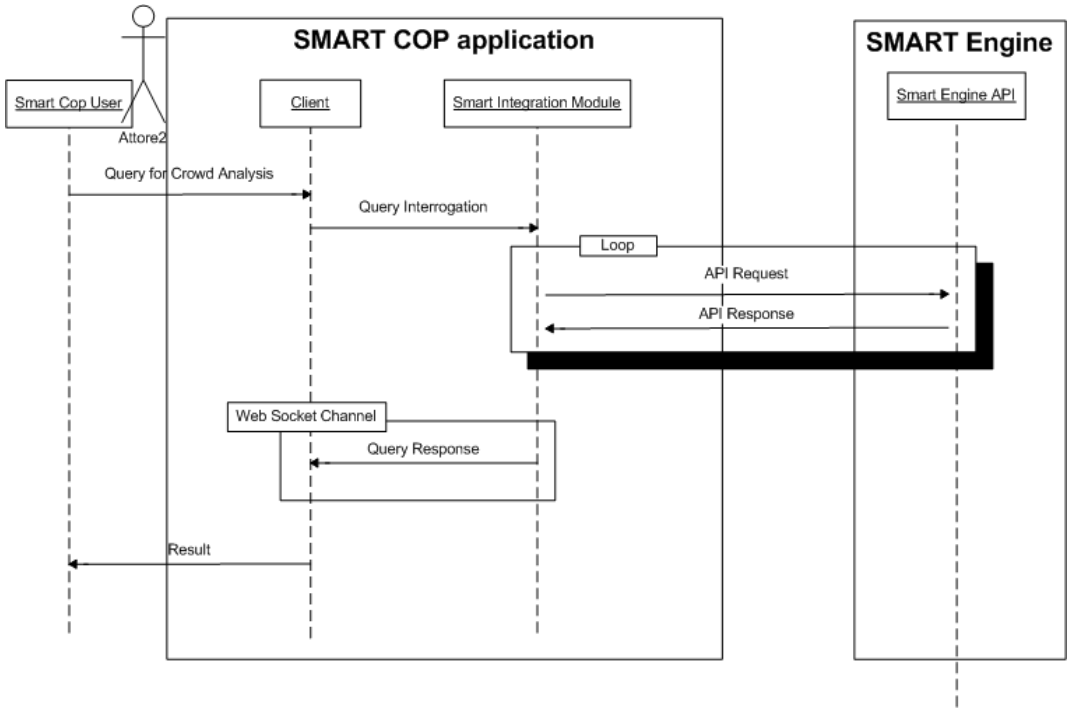

Figure 17: SMART COP applications interactions with the SMART Search Engine

7.3.2 Live News proof of concept

The live news proof of concept is intended to represent a simple version of the live news use case demonstrating how a final user can get information about his city using SMART data.

The high level functionalities provided by the Live news proof of concept are:

- Graphical representation of some event depending on the location.
- Representation of geolocated social network conversation.
- Access to PRISA's Eskup social network. Note that Eskup, allows readers to post comments and keep them on file. This allows them to participate in open chat channels on certain issues.

SMART Live news receives from SMART the following data:

- Events: special events occurring when something happens based upon rules.
- Social networks conversation density and trending topics.
- Geolocalized colour tendency.

SMART Live News connects to the SMART system to get data via Terrier search API. On the proof of concept predefined queries are going to be used depending on the user's location.

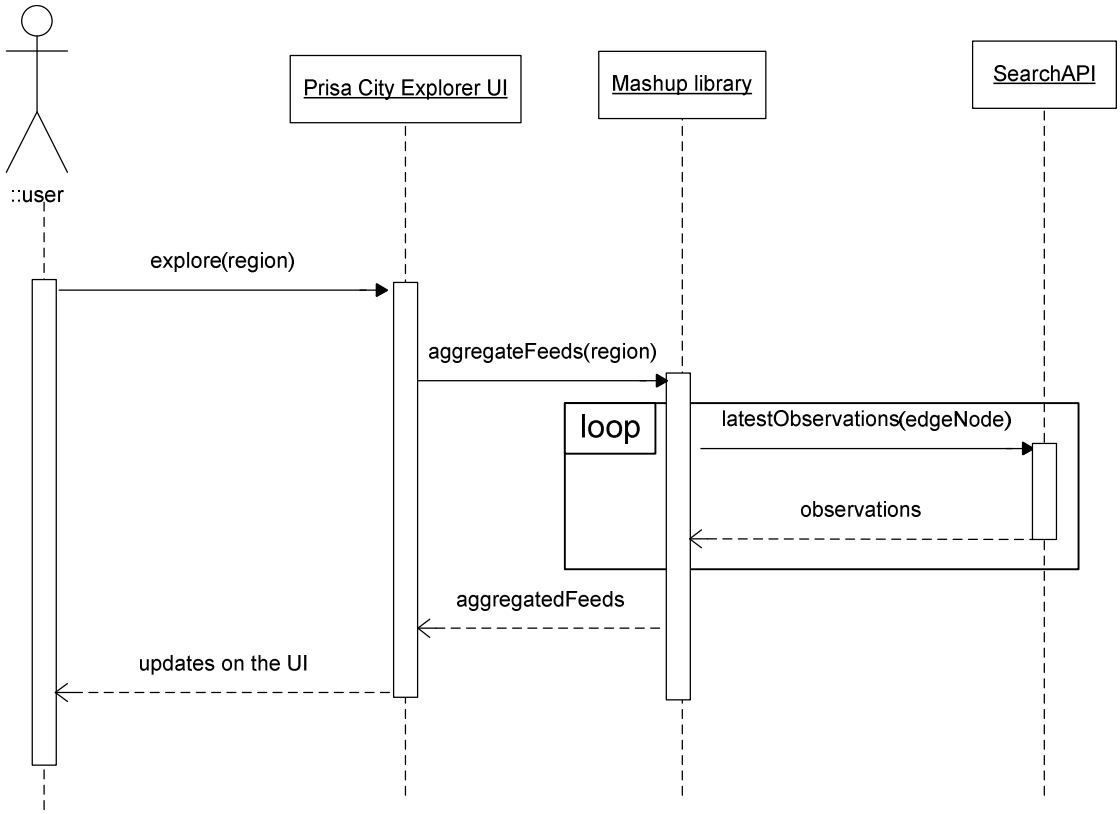

Figure 18: Live News application interactions with the SMART Search Engine

8. Conclusions

This deliverable has introduced the architecture of the SMART system. The architecture comprises a wide range of modules that can deliver all the main functionalities associated with the SMART systems including handling of metadata from multimedia signal processing algorithms, integration of data feeds from both sensors and social networks, support for advanced reasoning algorithms towards identifying events and patterns, the storage and management of media (audio/video) streams, as well as support for large scale searches over multiple geographically and administratively dispersed sensor repositories.

From a technical perspective the SMART architecture is structured in three distinct layers (edge node, search engine, applications), each one comprising specific components. Furthermore, it also includes the social network manager component, which transcends all the three layers outlined above. The edge node component acts as the proxy of the SMART system to the physical world. In particular, edge nodes are in charge of acquiring and combining metadata streams for related physical and virtual sensors that are attached to them. At the search layer lies the search engine (an extended/enhanced version of the Terrier search engine), which can index and retrieve data from numerous edge nodes. Finally, the applications layer interface to the search engine in order to execute queries and deliver application layer functionalities. Application development in SMART will be facilitated by a mash-up library comprising commonly used mash-up components (such as maps and charts).

The architecture boosts the modularity of the SMART compliant systems, thereby facilitating integration. At the same time it renders the system extensible in terms of new physical and virtual sensors, thereby ensuring openness. Furthermore, the architecture facilitates the management and organization of the project's open source development process through: (a) facilitating allocation of responsibilities for the different modules and subprojects of the open source software project, (b) ensuring the proper integration of the various components, (c) minimizing reliance on specific components thereby reducing risks, and (d) facilitating the planning of the development process (including relevant project management activities).

In a nutshell the SMART architecture boost the structuring and organization of several project tasks, which are currently in progress in the scope of other workpackages of the project. In this way, WP2 has accomplished its main objectives associated with the specification and delivery of the SMART architecture as means to modularize facilitate the main technical results of other workpackages. In particular, WP4 of the project is devoted to the edge node implementation; WP5 is devoted to the search engine implementation, while WP6 is devoted to the SMART applications. The SMART architecture (as described in this deliverable) will give a significant boost to these workpackages, while also facilitating their interactions (and the related collaboration of the partners).

9. References

[Albakour12] M.-D. Albakour, C. Macdonald, I. Ounis, A. Pnevmatikakis, J. Soldatos, «SMART: An Open Source Framework for Searching the Physical World», Proceedings of the ACM SIGIR 2012 Workshop on Open Source Information.

[Amati03] G. Amati. Probabilistic models for information retrieval based on divergence from randomness. PhD thesis, Department of Computing Science, University of Glasgow. 2003.

[Agarwal11] Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambowod, Rebecca Passonneau, «Sentiment analysis of Twitter data», in the Proceedings of the Workshop on Languages in Social Media (LSM '11), pp. 30-38

[BaezaYates2011] R. Baeza-Yates and B. Ribeiro-Neto. Modern Information Retrieval. Addison-Wesley, 2ed. 2011.

[Becker09] Becker H., Naaman M., Gravano L., “Event Identification in Social Media”, In WebDB, 2009.

[Cugola11] G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. ACM Computing Surveys, 2011

[Dean09] J. Dean. Challenges in building large-scale information retrieval systems: invited talk. In Proceedings of ACM Conference on Web Search and Data Mining (WSDM). 2009.

[Hansell2008] S. Hansell. Google Keeps Tweaking Its Search Engine. New York Times, 3rd June 2007. http://www.nytimes.com/2007/06/03/business/yourmoney/03google.html. 2007.

[Joachims02] T. Joachims: Optimizing search engines using clickthrough data. In Proceedings of 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 133-142.

[Killick2011] R. Killick, P. Fearnhead, and I.A. Eckley. Optimal detection of changepoints with a linear computational cost. arXiv. 2011.

[Kowalski86] R. Kowalski and M. Sergot (1986) "A Logic-Based Calculus of Events," New Generation Computing 4: 67–95.

[Krötzsch12] Markus Krötzsch, František Simančík, Ian Horrocks: A Description Logic Primer. CoRR abs/1201.4089. 2012

[Ounis06] I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. Terrier: A high per-

formance and scalable information retrieval platform. In Proceedings of the 2nd workshop on Open Source Information Retrieval at SIGIR 2006. pp18–25.2006.

[Santos11] R. L. T. Santos, C. Macdonald, I. Ounis. Exploiting query reformulations for web search result diversification. Proceedings of the 19th International Conference on World Wide Web (WWW). pp 881-890.

[Skarlatidis12] Anastasios Skarlatidis, Georgios Paliouras, Alexander Artikis, George A. Vouros: Probabilistic Event Calculus for Event Recognition CoRR abs/1207.3270: (2012)