SEVENTH FRAMEWORK PROGRAMME

Networked Media

Specific Targeted Research Project

SMART

(FP7-287583)

Search engine for MultimediA environment generated contenT

D4.1.1 SMART Distributed Knowledge Base and Open Linked Data Mechanisms

Due date of deliverable: 31-01-2014

Actual submission date: 05-02-2014

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

Summary of the document

Code:	**D4.1.1 SMART Distributed Knowledge Base and Open Linked Data Mechanisms**
Latest modification:	05/02/2014
State:	Final
Participant Partner(s):	AIT, Imperial College, ATOS, TELESTO
Author(s):	Aristodemos Pnevmatikakis, Nikos Katsarakis, Arta Babae, Josemi Garrido, Thanos Alexiou
Fragment:	NO
Audience:	☒ public ☐ restricted ☐ internal
Abstract:	This document contains the second version of the D4.1 SMART Distributed Knowledge Base and Open Linked Data Mechanisms [5].
Keywords:	• Metadata, Linked Data, RDF, XML, JSON • Knowledge-base, non-SQL database
References:	N/A

Table of Contents

1 Executive Summary

1.1 Scope

The SMART system is about offering users information fused from the physical world of sensors and the virtual world of social networks and Linked Data. The local entities that collect and fuse the information from these sources for a given location are the edge nodes. This document was originally designed as a companion to the edge node software release. Instead, we now see a twofold purpose: on the one hand we introduce the edge node architecture, data and capabilities, while on the other hand we provide an installation and usage manual for the edge node software components.

This second version of the D4.1 deliverable [5] is considered the functional manual for the edge nodes. It does not require the earlier version, although the SMART documentation on Trac:

http://opensoftware.smartfp7.eu/projects/smart

and the D4.2 deliverable [2] complement it in some aspects, as discussed in the text.

1.2 Audience

The handling of metadata in a SMART edge node is of great interest to a number of stakeholders including:

- **Developers of the SMART open source project**: This document will serve as a manual for the functionality provided by an edge node.
- **The Open Source Community**: Pointers to the SMART software releases, as well as the installation documentation in Trac are given in this document.
- **SMART project members (notably members working in WP5 and WP6 of the project)**: The present deliverable will provide valuable insights to all SMART project members as to how to employ the available metadata in the search and application layers.

1.3 Summary

From the functionality perspective the SMART edge nodes serve as the local source of information to the SMART system. The architecture supports metadata collection from diverse sources, fusion and reasoning upon them. The metadata, either raw or processed are stored in the edge node database, which is part of the distributed SMART knowledge base. The metadata are streamed, either within the different edge node modules, or towards the search layer from the database. All this process is demonstrated by an early proof-of-concept.

From a software implementation perspective, the SMART edge node comprises (i) components that process the data streams from the physical world of sensors and the virtual world of social networks and Linked Data, (ii) components that format the metadata feeds and ensure the communication to and from the database and finally (iii) components about reasoning.

All the functionality of the edge node is discussed in this document, since the functionality is now mature and some information in earlier WP4 deliverables is outdated. Some aspects are detailed, while either are summarised since the complete information can be found in the SMART documentation on Trac, and in the D4.2 deliverable [2].

1.4 Structure

The chapters in this document serve either as the manual for the software implementation, or as the edge node functionality presentation.

The document is structured as follows:

- Chapter 2 provides an introduction to the document by discussing the latest edge node architecture, and puts the deliverable in the scope of the rest of SMART documentation.
- Chapter 3 discussed the media clip generation at the Media Data Manager.
- Chapter 4 summarises how the social feeds from Twitter and Foursquare are generated at the

Social Network Manager. The complete information can be found in D4.2 [2].
- Chapter 0 details the Linked Data Manager that brings into the edge node local information of interest from the cloud.
- Chapter 6 updates the Intelligent Fusion Manager that reasons for higher-level context using the local feeds. The initial (now outdated) version has been discussed in D4.2 [2].
- Chapter 7 summarises the metadata handling module of the edge node. It serves as an update of the types of feeds discussed in D4.1 [5] and their description discussed in D3.1 [6]. It complements the documentation found in Trac, and provides pointers to it.
- Chapter 8 discusses the configuration layer across all the edge node modules.
- Finally, chapter 9 concludes the deliverable.

Only a minimum of information from other deliverables is repeated here, and this in a summary form. For information on the exact sections that this happens, see section 2.3.

2 Introduction

Traditional search engines offer information from documents in the Web. SMART is about real-time information gathered from the neighbourhoods of the cities we live in. This information is obtained by processing feeds from the physical world of sensors and the virtual world of social networks and Linked Data, turning those into metadata streams and fusing them together into interesting high-level events.

To do so the SMART system needs to be able to sense the neighbourhoods of the cities, and to collect information about them from the cloud for the Linked Data and the social networks.

2.1 The SMART edge node

The SMART edge node provides exactly this local functionality: It senses the pulse of the neighbourhoods by collecting and fusing the information from diverse sensors and networks for the given location. The edge node has to process continuously flowing data from a large number of geographically distributed and heterogeneous sources at unpredictable rate, to populate metadata feeds with the local information to be used in answering complex queries.

After year 2 of the project, the partners have installed five edge nodes:

- Two in the city of Santander: These are outdoors edge nodes populating visual, audio, twitter, Foursquare venues, sensor, city agenda, reasoning and clip information feeds. All these are discussed in Chapter 6.
- Athens Information Technology: Indoors edge node offering visual and social feeds, but also serving as a sandbox for the project partners and the community to test new feed deployment.
- Glasgow University: Indoors edge node offering visual and social feeds.
- Telesto (Athens): Outdoors edge node offering sensor feeds.

Such a small number of nodes can only serve to demonstrate the potential of SMART. For the system to be functional we need multiple edge nodes, contributed by the community. For this it is important to have a take-up of the SMART open source project. This document serves exactly this purpose: By demonstrating the capabilities of the SMART edge node and together with the online documentation in Trac providing for a software installation and usage manual, it is one part of the SMART documentation that will allow people outside of the consortium to setup their functional edge nodes in the way their metadata will be accessible by the SMART system.

2.2 Edge node architecture

The edge node is built in three modules (depicted vertically in Figure 2.1):

- **Metadata generation**: Here metadata are generated, either directly from physical sensors, or from virtual sensors. The latter are algorithms running on the signals from physical sensors, or on textual information from the cloud (social networks or linked data), in both cases extracting context. The algorithms on physical sensors are described in the WP3 deliverables, while those operating on textual information are described in chapters 4 and 0 of this document.
- **Metadata handling**: Here metadata are organized into feeds and are stored in the edge node database. To do so the feed metadata are first validated against the feed XML description (discussed in D3.1 and updated in section 7.2 of this document. The stored metadata can be streamed to the reasoning layer, or outside the edge node to the search engine. Metadata handling is discussed in section 0, while the search engine is discussed in WP5 deliverables.
- **Reasoning**: Here the edge node feeds are used to extract local higher-level context. As a result, the reasoning feed is populated, as described in chapter 6.

A horizontal **configuration** layer (also depicted in Figure 2.1) handles the configuration of the modules, as discussed in chapter 8.

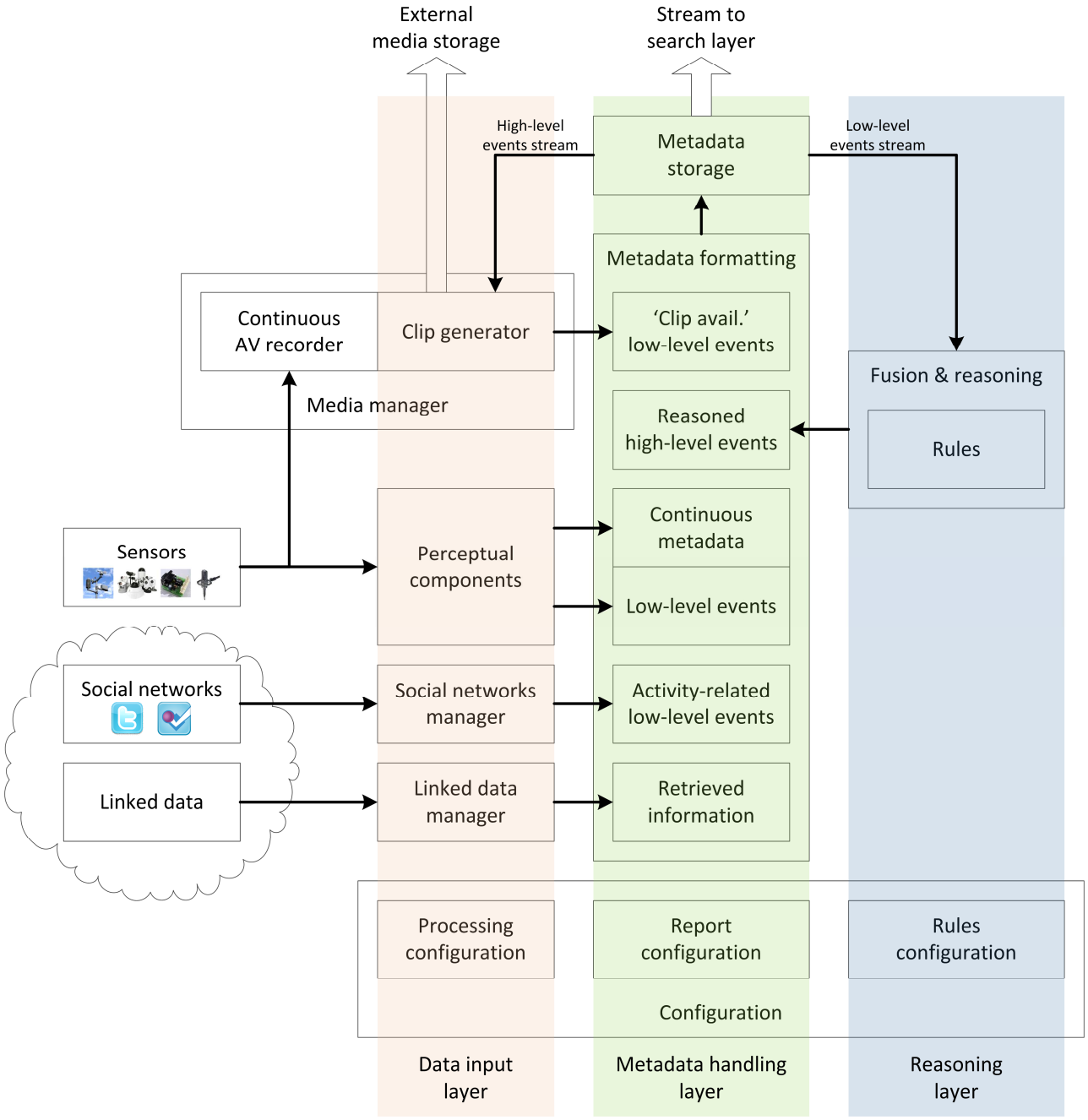

Figure 2.1: The edge node architecture.

2.3 Relation to the rest of the documentation

This document is the second version of D4.1 [5], but is meant to be used as independent of its predecessor. The information in this document together with the edge node documentation on Track is what users need to handle metadata in their nodes.

Information about the Social Network Manager can be found in detail in D4.2 [2]. Here only a summary is given.

Information about the Multimedia Data Manager requirements and design can be found in D2.3. Linked Data is presented in D5.2.

In chapter 7 of this deliverable we provide updates on the metadata feed description, originally discussed in D3.1 [6]. Metadata generation is only partially covered in this deliverable. Generation from perceptual components operating on physical sensors is handled in the WP3 deliverables. Only section 0 of this chapter has a summary of information that has already appeared in D4.1 [5] and is documented in Trac (as pointed by the numerous links therein).

D4.3 [4] discusses the installation of the software components presented in this deliverable. The second version of D4.3 [4] will contain any updates on the instructions, but also possible updates on the components described here.

3 Media Data Manager

The Media Data Manager (MDM) is the component of the Edge Node able to store multimedia information about the relevant events. The MDM stores and adapts the content, so the applications using SMART can easily present the video/audio files from the cameras to the users.

The challenges and requirements for the multimedia module of SMART, and the intended design of the MDM was outlined in the deliverable D2.3, “Multimedia search framework open architecture and Open Linked Data Mechanisms”, section 4.5.

The functionality is quite similar to a type of programs or devices called PVR (Personal Video Recorders). The MDM is able to create a video clip for a determined period of time, and is able to do so for a time in the past (for instance, yesterday between 3:03 and 3:04). This is mandatory because the important timespan of an event usually does not start at the moment it is detected, but sometime before. For instance, in car accident, the important interval does not start when there is the loud crash sound (that facilitates the event detection), but before, since it is important to know the circumstances leading to the accident.

In the months between the design outlined in D2.3 and this document, several implementations of MDM has been implemented and tested. The main change is that two different types of MDM exist:

- The “complete” MDM: A version similar to the one described in D2.3. This version stored permanently video clips only for relevant events.
- The “simple” MDM: A simplified version that stored all the contents, but temporally, as it is typical in surveillance applications. The main advantage of this version is that is independent from other modules in the Edge Mode, and therefore a lot easier to integrate.

3.1 Complete MDM

As it was stated in D2.3, multimedia data needs a lot of storage space, especially if we want to store video in High Definition. The capacity of hard disk is increasing each year, but a full disk of 2 Terabytes is needed for storing a month of HD video (for each camera). It is not feasible for the type of infrastructure that we can realistically use in SMART to buy and rack dozens of disks.

The solution is to store all video information only the time needed for selecting the relevant events, and store permanently the video data only for these relevant events.

3.1.1 Technical challenges

The main technical challenge for the MDM is **creating a video clip without visible jumps or interruptions**, and, at the same time, without making the program excessively complex, or entering directly in low level operations of video transcoding.

In order to achieve the maximum quality, the number of total encoding/decoding operations must be reduced to a minimum. Nearly all the codification algorithms for audio and video are “lossy”, and it is assumed that some quality is loss. Each codification operation implies a loss of quality.

The first step is to cut the stream of the camera in clips of the same length that can be stored. Several tools (as ffmpeg[1]) allow cutting operations in a video. But usually the precision of the operation is not enough, especially if we do not want video recoding. That is because in modern video formats, the frames are grouped in GOPs (Group Of Pictures), and the block begin and length is not related to second limits. Sound is grouped in blocks too, and block of video and audio are not correlated. It is possible to adjust the block length, but even using that way the results are not perfect, a very small jump in the images or sound is perceptible in the created clips for the event. Other solutions were tried, as the use of MJPEG or working frame by frame. In this case, the video is fixed but the audio continues to be a problem.

At the end, the solution was to use a Transport Stream from the MPEG standards. Transport Stream is a type of container that it is used for TV broadcasting. It has many features, but the main feature for

[1] http://www.ffmpeg.org/

MDM requirements is that TS stream is divided in packets of 188 bytes. It is possible to cut the stream by the packet limits, and store and recombine the packets later.

So the input stream is cut in 188 bytes packets and stored, along with metadata about timing. Doing this cut is not a trans-codification, so there is not loss of quality.

When it is necessary to construct a video for a given time interval, it is possible to select the packets corresponding to the time interval, and concatenate it again (in the original order, of course), and obtain a fragment of the actual stream. This operation is a simple concatenation, not a recodification, so the quality is the same as the original stream.

For the deployed system, we convert the input from any type of camera to a MPEG 2 TS using ffmpeg. The parameters are adjusted so all the frames are i-frames (key frames), so the stream is similar to a MJPEG, minimizing the transcoding losses in this step.

In any case, many client devices or video player are unable to play a MPEG 2 TS, so an additional final transcoding for adapting the contents to one or more devices and bandwidths is unavoidable. The MDM is able to use the MPEG 2 TS video as “master copy” to generate some other videos in other formats and bitrates (sending commands to ffmpeg)

In practice, another challenge related to the last one is **to store the video packets**. A multimedia file can be really long, and dividing it in millions of blocks of 188 bytes with metadata is not feasible in practice. The first step is to group 1024 packets in 18 kB chunks. Using our high quality parameters for MPEG 2, there are about 5 chunks per second, so the time resolution is good enough for SMART.

The initial solution was to save chunks as files, with timing information in a database, but modern databases are able to store binary data. Specifically, a feature of the MongoDB no-SQL database, called “capped collections”, is really suitable for storing these chunks,.

Capped collections are ordered cyclic buffers that delete older data automatically. The use of a database allows storing and retrieving the data and timing information together, and thousands of chunks can be stored without the limitations of file systems.

3.1.2 Implementation

The overview schema for the MDM does not change from D2.3 (repeated in Figure 3.1).

The MDM is an integration of several open source tools:

- CouchDB: the feed repository of the Edge Node, the Knowledge Base
- MongoDB: this no-SQL database store all contents in high definition temporally (the sensor data buffer in the diagram)
- Ffmpeg: the tool for converting between different video/sound formats.
- A web server (as Apache), that works as the permanent video repository and server
- cURL, the tool for copying video files from the Edge node to an external web server

All this low-level tools are orchestrated by two programs (blue in the diagram), that provide the management and integration layer.

- The “chopper” or “slicer” is a program that takes a MPEG 2 TS (Transport Stream) as input. The input can be directly a MPEG2 TS if the sensor produces it, or we can use a tool as ffmpeg to convert to this format. The “chopper” cuts the stream in blocks of 18 kB, and stores it with timing information in the database.
- The “feed reader” monitors the Knowledge Base, specifically the feed from the IMF (described later), and creates and stores the video clips for the specific relevant events.

Both programs are scripts in Python 2.7. In order to take advantage of modern processors those scripts spawn new processes for making the different operations. The python code runs without changes in MS Windows and Linux. The other tools are also ported to many systems which helps the MDM be portable.

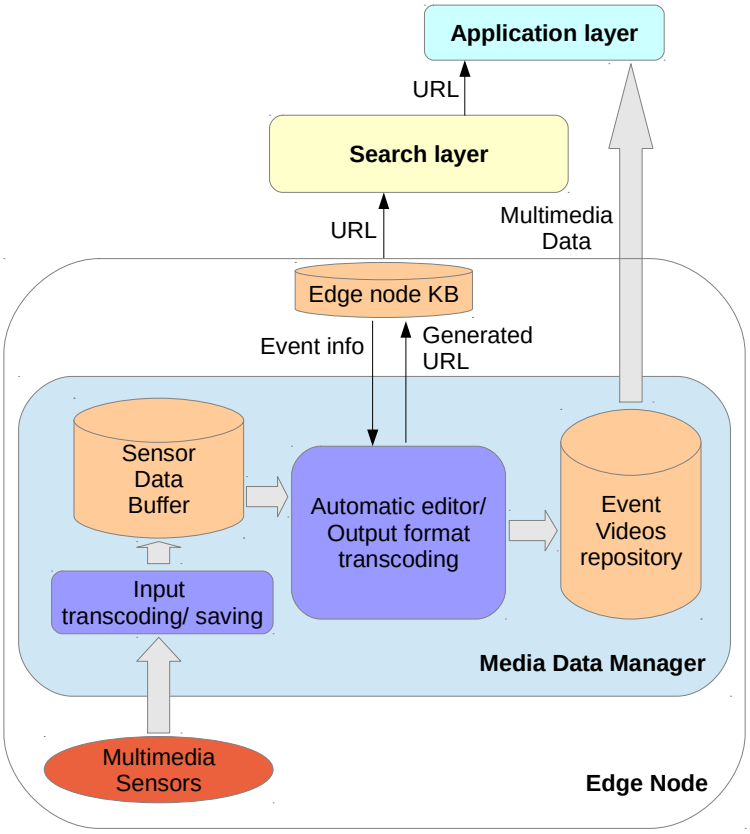

Figure 3.1: Overview of the Media Data Manager.

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

- Transcoding and storing the audio-visual data from the cameras. The data from the cameras will be transcoded to MPEG 2 TS, divided in 18kb chunks, and stored in the MongoDB database.
- Creating video clips for relevant events. We need to select the data corresponding with the time frame of the event, and concatenate the blocks to create a “master” video file for the event.
- Adapting 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 (the thumbnail).
- Storing the different event videos in the video repository/server. The video is uploaded using cURL.
- Creating one or more URL for the stored contents that can be accessed by an application. This URL is stored in a JSON document, and in turn it is kept with the rest of the searchable document in the Knowledge Base (couchDB feeds). So the URL can be obtained using the Search Layer, and in turn, the application can present this multimedia information to the user

3.2 Simple MDM

The version of the MDM described above is able to satisfy the requirements, but there is a problem: it depends completely of other unit that determine what events are relevant (according to the current architecture, the IMF). Without this, no relevant event video is store. So in order to facilitate the integration, a different version of the MDM was developed.

In this version, 60 second clips are taken and stored for some time (currently, a week). Everything is stored and can be searched. But the video is not centred on the relevant events. There is also a 1-2 second gap between the videos.

The Simple MDM generates the final video file directly in MPEG4 format. It is not able to provide

several formats or generating a thumbnail. Fortunately, the output video format is very popular nowadays. For example: it can be used by the Liferay[2] portal, by some Adobe Flash Players and by the iPhone, iPad and many Android devices.

In any case, this version is invaluable providing some multimedia contents to the development and integration of other modules and applications

3.2.1 Implementation

Similar to the complete version described above, the simple MDM uses some open source tools:

- CouchDB, the knowledge base
- FFmpeg, to capture a convert the video
- cURL to transmit the video

The Simple MDM is a Python 2.7 program. In this case, there is only a module, the chopper that performs all the functions.

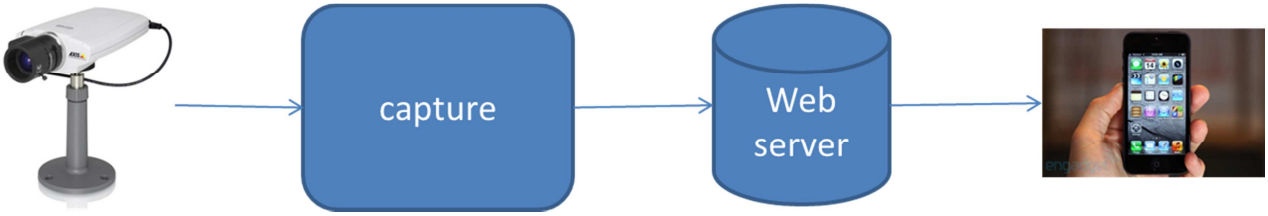

Figure 3.2: Functional diagram of the Simple form of the Media Data Manager.

The Simple MDM repeats a loop with the following steps (also shown in Figure 3.2):

- The simple MDM connects to the camera using ffmpeg and creates 60 seconds video clips, stored as a file.
- The video is transmitted to the video repository server using cURL.
- A JSON containing the URL for the video is created, and stored in the Knowledge Base. The format is the same as used in “regular” MDM, so for the rest of the system and client application is transparent.

[2] http://www.liferay.com/

4 Social Network Manager

The social network manager is currently using data from Twitter[3] and Foursquare[4], as discussed in the following sections. The social network manager also supports Facebook, but its usage in the SMART edge nodes has been put in hiatus due to privacy concerns.

4.1 Twitter

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

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

Social Network Manager uses Twitter4J[5], an unofficial Java library to easily integrate with the Twitter service and communicate with the API in order to gather tweets containing the requested words/phrases or tweets containing geolocation data located within the requested boundaries.

4.2 Foursquare

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

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

Social Network Manager uses Foursquare V2 API for Java[6], an open source easy library implementing the use of the foursquare API using the Java programming language, to query the Foursquare API and access the information requested. Such information are currently consisted mainly of total check-ins or current check-ins for a selected venue or all the venues in a specified area in an attempt to provide useful information to the upper layers of SMART for event recognition such as a “crowded” event.

4.3 Temporal hints

Social Network Manager comes with a filter named “TwitterDateTimeExtractor” which is able to filter recent tweets for the ones containing the requested word/activity, for example “concert”. Upon filtering the tweets, the filter works on the set of tweets containing the actual word/activity by searching them for any temporal information that come within the tweet text or in the supporting tweet metadata such as dates and/or times that will probably be associated with the actual activity referred. Results are provided as a whole for the requested activity and not per tweet basis and they are formatted in XML or RDF. The information currently extracted is summarized in Table 4.1.

[3] https://twitter.com/

[4] https://foursquare.com/

[5] http://twitter4j.org/en/index.html

[6] http://code.google.com/p/foursquare-api-java/

Table 4.1: Example observations and associated outcome.

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

The filter is considered to be a basic example of the possibilities the SNM has through the filter system, it is possible for developers to create new filters based on the filter interface [D4.2] or further develop the existing filter extending its' filtering capabilities. For example a possible update could be the classification of the dates and/or times that come with the activity into "starting" or "ending" dates, times.

5 Linked Data Manager

Linked Data has a double role in SMART, since it is used both in the Edge Node and in the Search Engine Layer (WP4 and WP5). As the requirements are similar, both functionalities can be addressed by the same module, the Linked Data Manager (LDM).

- In the Edge node, the Linked Data Manager is a "virtual sensor", as for instance, the Social Network Manager described in chapter 4. The relevant results from the Linked Data repositories are stored as a feed in the Knowledge Base (couchDB).
- At the Search Engine level, the LDM is used to automatically add descriptions to some keywords and locations (see D5.2).

5.1 Feature description

The Linked Data Manager offers an API as REST interface. The functionality of the LDM is divided in two parts, textual search and structured search.

5.1.1 Textual search

In this type of search the LDM searches in the textual fields of entities: comment, abstract, name, label. The aim is is to discover linked data resources on the web that match with the search criteria (keywords specified by a user).

There are two different types of textual search:

- Search Activities. Find activities whose text fields match against a regular expression pattern (user keywords). For instance, we can search for activities as Formula 1 if we search for "Grand Prix"
  An example can be the web service invocation:
  /SMART_LinkedDataManagerREST/rest/txtSearch/activities?label=%22Grand%20Prix%22
- Search Venues. Find venues whose text fields match against a regular expression pattern (user keywords). Using this text search, we can find "venues" related to Santander (as name).
  /SMART_LinkedDataManagerREST/rest/txtSearch/venues?label=%22santander%22

This functionality is used mainly by the Edge Node, where the LDM is considered yet another virtual sensor.

5.1.2 Structured search

In this type of search the LDM performs a structured query. In the case of SMART, it is focused in Geospatial queries for locations and activities.

It is possible to search for venues and activities using the geographical range of interest as a square or as a circle around a point of interest. Therefore, there are four different REST calls available:

- Search Venues place in a rectangle.
  /SMART_LinkedDataManagerREST/rest/structuredSearch/locRec?lat1=40.157623&long1=-74.855347&lat2=41.077281&long2=-73.586426
- Search Venues place in a circle.
  /SMART_LinkedDataManagerREST/rest/structuredSearch/locCirc?lat1=40.440497&long1=-3.692093&radius=100
- Search Activities place in rectangle
  /SMART_LinkedDataManagerREST/rest/structuredSearch/actRec?lat1=40.157623&long1=-74.855347&lat2=41.077281&long2=-73.586426
- Search Activities place in a circle
  /SMART_LinkedDataManagerREST/rest/structuredSearch/actCirc?lat1=40.440497&long1=-3.692093&radius=100

This functionality is used mainly by the Search Engine layer.

5.2 Output

All the REST calls return a JSON with the main data and references to the original pages in the Linked Data repositories. An example of output can be:

```
{
  "data": {
    "time": "2014-02-04T11:03Z",
    "locations": {
      "http:\/\/dbpedia.org\/resource\/Estadio_El_Sardinero_(1913)": {
        "location": [
          "http:\/\/dbpedia.org\/resource\/Estadio_El_Sardinero_(1913)",
          "Santander, Spain@en"
        ],
        "isPrimaryTopicOf": [
          "http:\/\/en.wikipedia.org\/wiki\/Estadio_El_Sardinero_(1913)"
        ],
        "txt": [
          "Estadio El Sardinero (1913)",
          "Estadio El Sardinero was a multi-use stadium in Santander…        ]
      }
    }
  }
}
```

The number of results can be very high, so only a sample is presented. The exact fields of the returned JSON are a bit different for the textual or structured search, or if we search for venues or activities.

5.3 Implementation and integration with Knowledge Base

The main component of the LDM is implemented as Java program. The final product of the compilation is a WAR file that can be installed in any Java Application Server like Apache Tomcat.

Each web service in the API works executes the following steps (see Figure 5.1):

- The input parameters in the API are converted to a suitable semantic query.
- The query is sent to the right set of Linked Data repositories. We can use dbPedia, FactForge and GeoNames. The search is made in real-time directly in the repository, not in a local copy. This way we always have an actualized response. The drawback is that the query can take several seconds, especially if the number of results is high.
- The different formats of the results are translated to the JSON format discussed in section 5.2 and returned.

Figure 5.1: Functional diagram of the Linked Data Manager.

The “web services” model is different than the model of “Virtual Sensor” that we use in SMART Edge Nodes. In order to integrate the web services with the Knowledge Base, an additional component is required. This component queries the main module at regular intervals, using terms related to the location of the specific Edge Node, and the response is stored as a document in the edge node database in order to populate the linked data feed to be used by the rest of the system.

6 Intelligent Fusion Manager

Each SMART edge node has an Intelligent Fusion Manager (IFM) where the reasoning about incoming data takes place. IFM uses the Alchemy reasoning engine which is an implementation of Markov Logic Networks (MLN) [1]. MLN is an approach based on combining first-order logic (FOL) and graphical models into a single representation. We gave a summary of MLN in D4.2 [2]. A complete survey on MLN can be found in [3].

During the learning phase, the engine receives annotated observations and some rule-based patterns accounting for this data. In the inference mode, given some sensors' data and some rule-based patterns, the engine derives new information with some attached probabilistic weight. The result will then be populated in the database CouchDB. We give an example in the next section to clarify these steps further.

In [3] we showed how to combine data from audio or video with data from sensors like thermometers to enrich the reasoning about a situation. The example in the next section shows how social network data along with sensor data can be utilised to infer about an event.

6.1 An example of inference about the type of an event

In this section, we present an example which can clarify the fusion of social network and physical data. Consider the scenario in which the target is to decide that an outside event is accident or a concert/celebration (Accident versus Jubilation). We first need to collect for a period of time the density of the crowd in a venue by the predicate CrowdDensity. Also, consider the frequency of a certain words corresponding to a celebratory event (e.g. "Live music" and "celebration") from the tweets around the area in which the edge node is located. We call this MusicFreqValue. We can also get the applause factor from audio input, Applause. We define Notification as the predicate that represents the type of an event.

First we let the engine learn the rules from annotated observations, like those shown in Table 6.1.

Table 6.1: Example observations and associated outcome.

Observation	Outcome
Applause(High), CrowdDensity(High), MusicFreq(High)	Jubilation
Applause(Low), CrowdDensity(High), MusicFreq(Low)	Accident
Applause(High), CrowdDensity(High), MusicFreq(High)	Jubilation
Applause(High), CrowdDensity(High), MusicFreq(High)	Jubilation
Applause(High), CrowdDensity(High), MusicFreq(High)	Accident
…	

The result of the learning phase would be attaching a corresponding weight to the type of the event based on different permutations of the input variables. The higher weight in this context corresponds to a higher probability. Note that until annotated data is used these rules can be hard-coded by setting weights to the desired values. An example of this is given in Table 6.2.

Table 6.2: Example of weights associated with observations. These are either learnt, or manually configured in the absence of annotated training data.

Observation	Weights w.r.t. Accident
Applause(High), CrowdDensity(High), MusicFreq(High)	-3
Applause(Low), CrowdDensity(High), MusicFreq(High)	2
Applause(Low), CrowdDensity(High), MusicFreq(Low)	10
Applause(High), CrowdDensity(High), MusicFreq(Low)	-1
Applause(High), CrowdDensity(Low), MusicFreq(High)	-4
…	

After learning, a new observation can be fed to the reasoning engine and then the learned (or configured) rules are utilised to assign a probability to the observation. Regarding our example, the probability of the event being an accident or celebration is returned as shown in Table 6.3.

Table 6.3: Using the trained reasoning engine to get probabilities of events.

Observation	Probability of the event being an accident
Applause(High), CrowdDensity(High), MusicFreq(High)	0.001

This means that there is a very low probability of a notification of type “Accident” when there is a high crowd density, high applause factor, and high frequency of the words related to celebrations in tweets.

6.2 Latest version of the Reasoning component

Currently, a simple reasoning feed has been integrated within edge node that can process the visual metadata by parsing the video feed and decide whether there is a noteworthy event is happening in the edge node area. It then puts back the result into the reasoning feed which is described in section 7.2.9. The time stamp of the result is the same as the time stamp of the video which has been parsed.

7 Edge node data repository

7.1 CouchDB databases

In every edge node, the CouchDB hosts a number of databases:

- One database per feed. Records hold metadata at different time instances.
- A database summarizing the feeds. Records hold metadata describing the different feeds

7.2 Metadata feeds

7.2.1 Grouping information sources into feeds

By the end of year 2, the metadata have been organised in feeds based on the rule one feed per sensor. This results to a single feed regarding audio, video, city agenda and twitter. The list is augmented by a reasoning feed after processing the rest of the feeds of the node and a clip information feed describing what is recorded.

On top of those we have the 180 sensor feeds and many feeds from Foursquare, one per venue of interest. In total, this is a large number of feeds.

Since every feed requires an HTTP connection at the search engine, this situation is not scalable. Instead, the sensor and venue feeds are grouped geographically into feeds with multiple components. See how this is done in sections 7.2.5 and 7.2.7.

7.2.2 Feed description generation tool

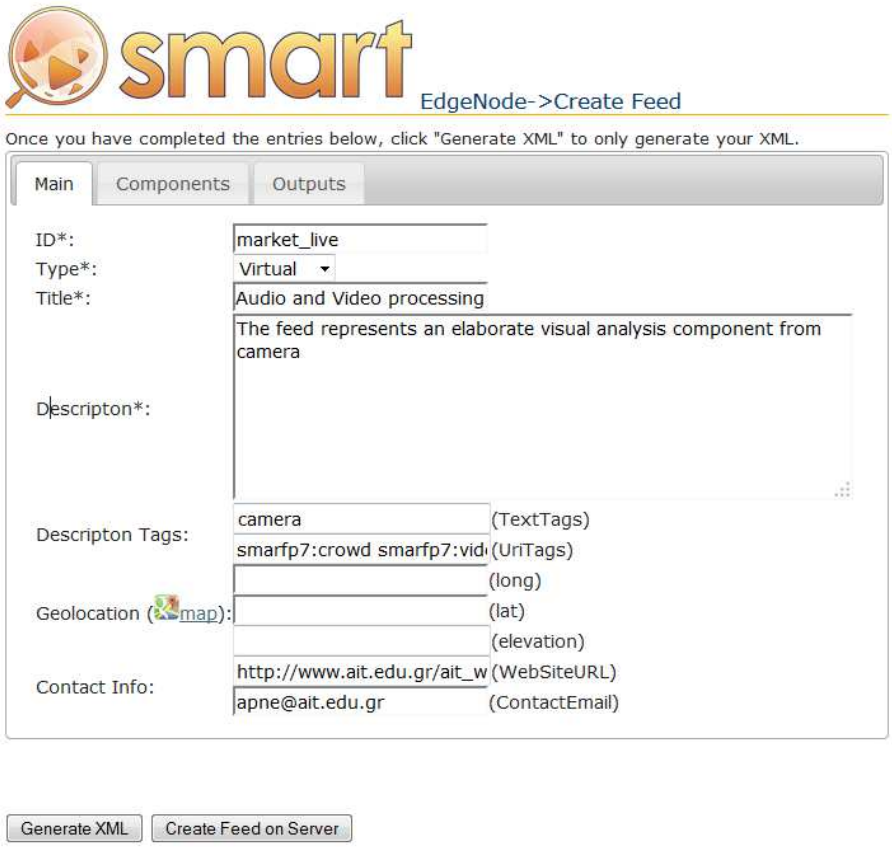

Figure 7.1: Feed description generation tool – The Main tab.

The feed description generation tool simplifies the process of generating the XML description with

which the feed metadata need to comply. At the sandbox edge node, the tool is accessible at http://dusk.ait.gr/SMART-FP7/CreateFeed/

At the Main tab (see Figure 7.1), the feed identifier, title, textual description and contact information are added. The most important sections are:

- The feed type selector, allowing the choice between physical and virtual (see D3.1 [6]).
- The description tags, both textual and URI, which allow the search engine to know what the metadata are about (see D5.2 [8]).
- The geolocation information. By clicking on the map, a popup window is opened, powered by the Google Maps API (see Figure 7.2). There one can search for a location and then refine the marker manually (either by moving the market or by changing the latitude/longitude and updating the position on the map). When finalized, the information is updated in the main tab.

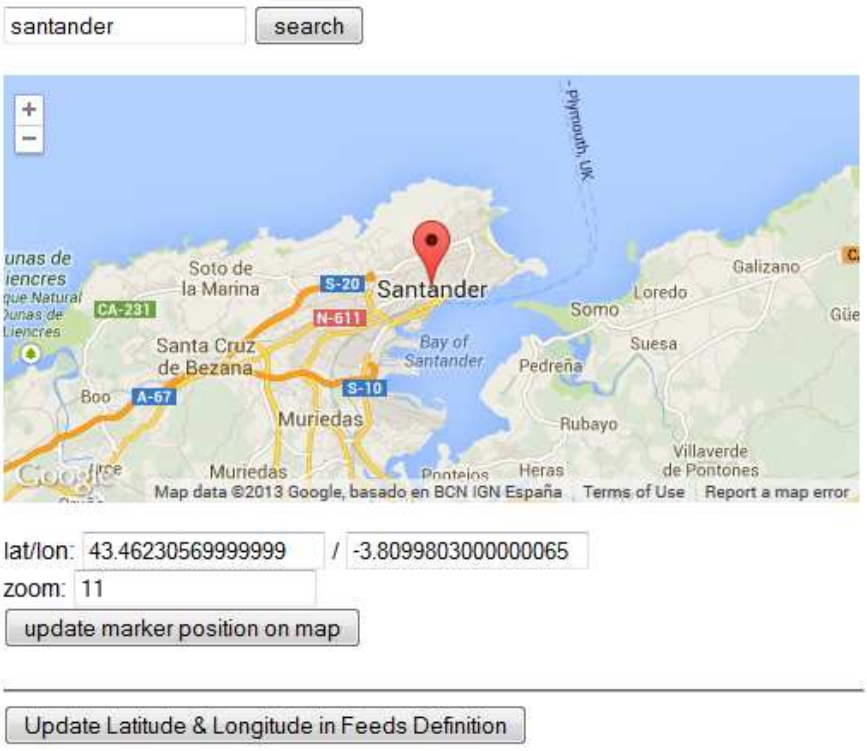

Figure 7.2: Feed description generation tool – Geolocation.

At the Components tab the components are described. The user can add, expand, collapse and delete either physical or virtual components.

The components come with their own description tags and geolocation.

Their type refers to a categorisation into different processing modules used. At year 2, we use the following types in our edge node components:

- visual_density: Components measuring the amount of foreground activity in the region of interest in the camera frames. The tags in its description specify if this density is primarily due to vehicles or crowds.
- objcounter: Components counting the objects passing through gates of interests in the camera frames.
- audio: Components carrying the results from audio processing of the microphone signals.
- social: Components conveying information from social networks (Twitter and Foursquare) as well as any agenda yielding events' schedule.
- gateway: Components describing a collection of sensors connected at the same gateway.

- reasoning: Components describing the result of local reasoning at the edge node.
- clip_info: Components describing the media clips recorded by the Media Data Manager.
- ldm_result: Components describing the search results from the Linked Data Manager.

Their efficiency is quantifying the accuracy of the component's outputs, assuming that it does not change over time. If it does, then we set the "Has Confidence" property of the individual output to true and utilise the <output_name>_confidence field in the JSON.

As an example five virtual components are added (see Figure 7.3):

- Three for visual analysis in three different zones of interest on the frame, namely the roads (where the measured foreground is a mixture of vehicles and pedestrians), the square and the sidewalks (where the measured foreground is due to crowds)
- Two for object counter at the entrance and exit of the underground parking (see Figure 7.4).

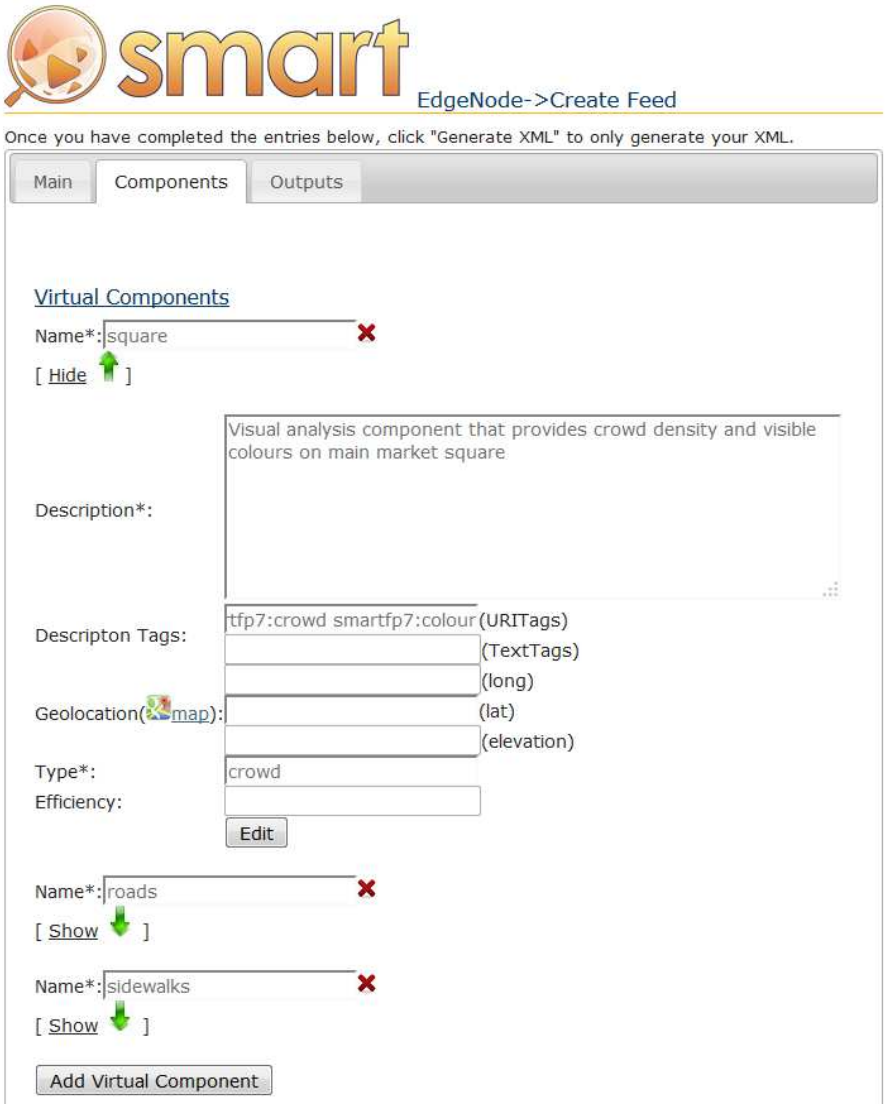

Figure 7.3: Feed description generation tool – The Components tab.

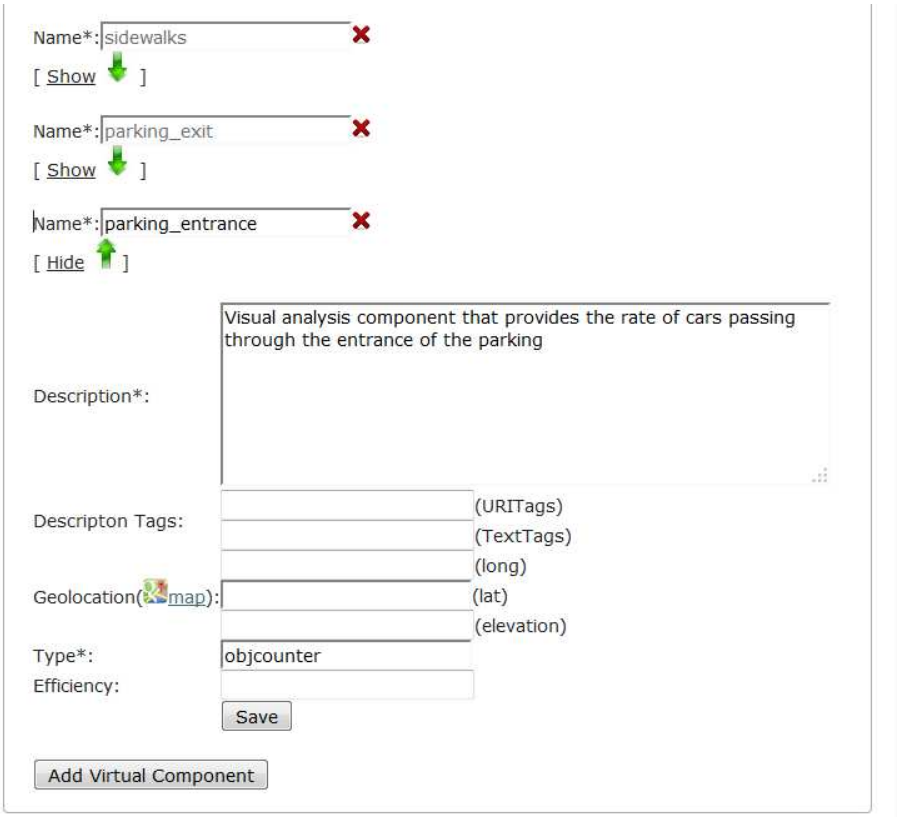

Figure 7.4: Feed description generation tool – Details of the object counter component in the Components tab.

Finally the outputs are associated to the components at the Outputs tab. The outputs have their own description tags, but not independent localisation. They have a type (integer, double, strings, arrays of such) and possibly a confidence.

Continuing the example above, 11 component outputs are added (see Figure 7.5):

- Three outputs are associated to each visual analysis component. The “density” is of type “double”, “colournames” is of type “array(string)” and “colourpercentages” is of type “array(double)”.
- The output “obj_per_min” of type “double” is associated to each object counter component (see Figure 7.6).

None of the outputs has confidence associated with it.

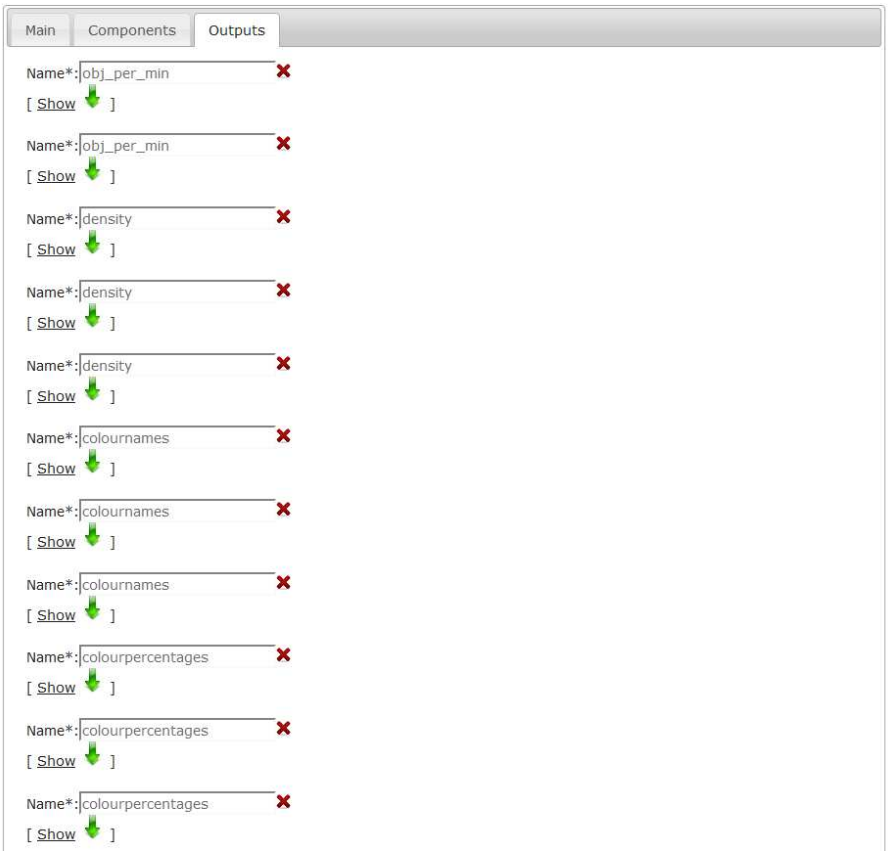

Figure 7.5: Feed description generation tool – The Outputs tab.

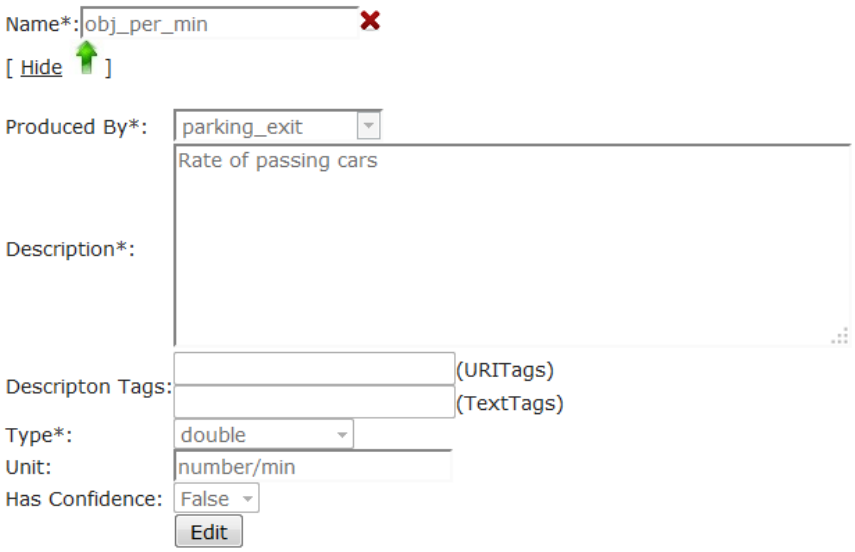

Figure 7.6: Feed description generation tool – Details of the object counter output in the Outputs tab.

Upon finalising the description, clicking “Generate XML” generates the document.

7.2.3 Visual processing feed

Four out of the five edge nodes operational by the end of year 2 of SMART have a visual processing

feed, namely the two Santander, the AIT and the Glasgow edge nodes. The Year 2 version of this feed includes visual density and colour analysis in different processing zones of interest.

The XML description of the visual processing feed is given below:

```
<?xml version="1.0" encoding="UTF-8"?>
<Feed xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://dusk.ait.gr/SMARTEdgeNode/SMART_Datafeed_Schema_v0.3.xsd"
>
    <Id>market_live</Id>
    <Type>Mixed</Type>
    <Title>Video processing Feed</Title>
    <Description>Visual processing at Mercado de la Esperanza</Description>
    <DescriptionTags>
        <TextTags>camera</TextTags>
        <URITags>smarfp7:video</URITags>
    </DescriptionTags>
    <Geolocation>
        <Longitude>-3.8106562166717595</Longitude>
        <Latitude>43.462796296630465</Latitude>
    </Geolocation>
    <ContactInfo>
        <Website>http://www.ait.edu.gr/ait_web_site/faculty/apne/pnevmatikakis.html</Website>
        <ContactEmail>apne@ait.edu.gr</ContactEmail>
    </ContactInfo>
    <Components>
        <Virtual>
            <Name>square</Name>
            <Description>Crowd density and colour analysis on main market square</Description>
            <DescriptionTags>
                <URITags>smartfp7:visual_density smartfp7:crowd smartfp7:colour</URITags>
            </DescriptionTags>
            <Type>visual_density</Type>
        </Virtual>
        <Virtual>
            <Name>roads</Name>
            <Description>Visual density and colour analysis on the two roads</Description>
            <DescriptionTags>
                <URITags>smartfp7:visual_density smartfp7:vehicle smartfp7:colour</URITags>
            </DescriptionTags>
            <Type>visual_density</Type>
        </Virtual>
        <Virtual>
            <Name>sidewalks</Name>
            <Description>Visual density and colour analysis on sidewalks</Description>
            <DescriptionTags>
                <URITags>smartfp7:visual_density smartfp7:crowd smartfp7:colour</URITags>
            </DescriptionTags>
            <Type>visual_density</Type>
        </Virtual>
        <Virtual>
            <Name>parking_exit</Name>
            <Description>Rate of cars exiting the parking</Description>
            <Type>objcounter</Type>
        </Virtual>
        <Virtual>
            <Name>parking_entrance</Name>
            <Description>Rate of cars entering the parking</Description>
            <Type>objcounter</Type>
        </Virtual>
    </Components>
    <Outputs>
        <Output>
            <Name>obj_per_min</Name>
            <ProducedBy>parking_exit</ProducedBy>
            <Description>Rate of passing cars</Description>
            <Type>double</Type>
            <Unit>number/min</Unit>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
```

```
            <Name>obj_per_min</Name>
            <ProducedBy>parking_entrance</ProducedBy>
            <Description>Rate of passing cars</Description>
            <Type>double</Type>
            <Unit>number/min</Unit>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>density</Name>
            <ProducedBy>roads</ProducedBy>
            <Description>Vehicle density</Description>
            <Type>double</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>density</Name>
            <ProducedBy>sidewalks</ProducedBy>
            <Description>Crowd density</Description>
            <Type>double</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>density</Name>
            <ProducedBy>square</ProducedBy>
            <Description>Crowd density</Description>
            <Type>double</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>colournames</Name>
            <ProducedBy>roads</ProducedBy>
            <Description>Names of visible colours</Description>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>colournames</Name>
            <ProducedBy>sidewalks</ProducedBy>
            <Description>Names of visible colours</Description>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>colournames</Name>
            <ProducedBy>square</ProducedBy>
            <Description>Names of visible colours</Description>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>colourpercentages</Name>
            <ProducedBy>roads</ProducedBy>
            <Description>Weights of visible colours</Description>
            <Type>array(double)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>colourpercentages</Name>
            <ProducedBy>sidewalks</ProducedBy>
            <Description>Weights of visible colours</Description>
            <Type>array(double)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>colourpercentages</Name>
            <ProducedBy>square</ProducedBy>
            <Description>Weights of visible colours</Description>
            <Type>array(double)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
    </Outputs>
</Feed>
```

7.2.4 Audio processing feed

Only the two Santander edge nodes out of the total five operational by the end of year 2 of SMART have an audio processing feed. The Year 2 version of this feed offers audio event classification. There is a single audio_events component, with eight outputs yielding information about the time in seconds, six possible audio events scores (crowd, traffic, applause, music, speech and siren) and the sound intensity level in dB.

```
<?xml version="1.0" encoding="UTF-8"?>
<Feed xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SMART_Datafeed_Schema_v0.3.xsd">
    <Id>audio_feed_market</Id>
    <Type>Virtual</Type>
    <Title>Audio feed</Title>
    <Description>Audio analysis using Smart DBN audio classifier</Description>
    <DescriptionTags>
        <TextTags>audio</TextTags>
        <URITags>smartfp7:audio</URITags>
    </DescriptionTags>
    <ContactInfo>
        <ContactEmail>zvi@il.ibm.com</ContactEmail>
    </ContactInfo>
    <Components>
        <Virtual>
            <Name>audio_events</Name>
            <Description>audio events classification (Mercado de la Esperanza)</Description>
            <DescriptionTags>
                <TextTags>audio crowd traffic music applause street square</TextTags>
            </DescriptionTags>
            <Type>audio</Type>
        </Virtual>
    </Components>
    <Outputs>
        <Output>
            <Name>position</Name>
            <ProducedBy>audio_events</ProducedBy>
            <Description>Time within the sample</Description>
            <Type>double</Type>
            <Unit>seconds</Unit>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>crowd_score</Name>
            <ProducedBy>audio_events</ProducedBy>
            <Description>Score for crowd audio</Description>
            <Type>double</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>traffic_score</Name>
            <ProducedBy>audio_events</ProducedBy>
            <Description>Score for traffic audio</Description>
            <Type>double</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>applause_score</Name>
            <ProducedBy>audio_events</ProducedBy>
            <Description>Score for applause audio</Description>
            <Type>double</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>music_score</Name>
            <ProducedBy>audio_events</ProducedBy>
            <Description>Score for music audio</Description>
            <Type>double</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
```

```
            <Name>speaker_score</Name>
            <ProducedBy>audio_events</ProducedBy>
            <Description>Score for speaker audio</Description>
            <Type>double</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>siren_score</Name>
            <ProducedBy>audio_events</ProducedBy>
            <Description>Score for siren audio</Description>
            <Type>double</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>level</Name>
            <ProducedBy>audio_events</ProducedBy>
            <Description>Audio level in dB</Description>
            <Type>double</Type>
            <Unit>dB</Unit>
            <HasConfidence>false</HasConfidence>
        </Output>
    </Outputs>
</Feed>
```

7.2.5 Venues feed from Foursquare

Only the two Santander edge nodes out of the total five operational by the end of year 2 of SMART have foursquare venue feeds. The Year 2 version of this feed offers the number of people at a venue at a given time and the total check in at a venue, organised in two outputs of a virtual component related to the venue. Multiple venues can be supported simply by adding more components. For this reason it is best to have the geolocation per component and not at the main feed description.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Feed  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SMART_Datafeed_Schema_v0.3.xsd">
    <Id>fsq_venue_4b952850f964a5200a9234e3</Id>
    <Type>Virtual</Type>
    <Title>Mercadona, calle Castilla</Title>
    <Description>{"venueDescription":{"location":{"lat":43.45421,"lng":-
3.8203614,"country":"Spain","cc":"ES"},"categories":[{"id":"4bf58dd8d48988d1ff941735","name":"M
iscellaneous Shop","pluralName":"Miscellaneous
Shops","shortName":"Shop","icon":{"prefix":"https://foursquare.com/img/categories_v2/shops/defa
ult_","suffix":".png"},"primary":true}]}}</Description>
    <DescriptionTags>
        <TextTags>social foursquare here_now miscellaneous shop</TextTags>
        <URITags>smartfp7:here_now smartfp7:checkins_count smartfp7:social
smartfp7:foursquare</URITags>
    </DescriptionTags>
    <ContactInfo>
        <ContactEmail>thanos@telesto.gr</ContactEmail>
    </ContactInfo>
    <Components>
        <Virtual>
            <Name>foursquare_venue</Name>
            <Description>Venue: Mercadona, calle Castilla Foursquare</Description>
            <DescriptionTags>
                <TextTags>mercadona, foursquare</TextTags>
                <URITags>smartfp7:foursquare</URITags>
            </DescriptionTags>
            <Geolocation>
                <Longitude>-3.8203614</Longitude>
                <Latitude>43.45421</Latitude>
            </Geolocation>
            <Type>social</Type>
        </Virtual>
    </Components>
    <Outputs>
        <Output>
            <Name>here_now</Name>
            <ProducedBy>foursquare_venue</ProducedBy>
```

```
            <Description>current here_now</Description>
            <DescriptionTags>
                <TextTags>here_now</TextTags>
                <URITags>smartfp7:here_now</URITags>
            </DescriptionTags>
            <Type>integer</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>checkins_count</Name>
            <ProducedBy>foursquare_venue</ProducedBy>
            <Description>current checkins_count</Description>
            <DescriptionTags>
                <TextTags>checkins_count</TextTags>
                <URITags>smartfp7:checkins_count</URITags>
            </DescriptionTags>
            <Type>integer</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
    </Outputs>
</Feed>
```

7.2.6 Twitter feed

The Santander Twitter feed contains the geolocated tweets of the Santander area.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Feed  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SMART_Datafeed_Schema_v0.3.xsd">
    <Id>santander_twitter</Id>
    <Type>Virtual</Type>
    <Title>Santander Twitter</Title>
    <Description>This feed contains tweets from the Santander city</Description>
    <DescriptionTags>
        <TextTags>twitter tweet santander</TextTags>
        <URITags>smartfp7:twitter smartfp7:tweet smartfp7:santander</URITags>
    </DescriptionTags>
    <ContactInfo>
        <ContactEmail>thanos@telesto.gr</ContactEmail>
    </ContactInfo>
    <Components>
        <Virtual>
            <Name>sdr_tweets</Name>
            <Description>Tweets of Santander Area</Description>
            <Type>social</Type>
        </Virtual>
    </Components>
    <Outputs>
        <Output>
            <Name>tweet</Name>
            <ProducedBy>sdr_tweets</ProducedBy>
            <Description>Text of the SDR-related tweets</Description>
            <Type>string</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
    </Outputs>
</Feed>
```

7.2.7 Sensor feed

The two Santander and the Telesto edge nodes out of the total five operational by the end of year 2 of SMART have sensor feeds. The Year 2 version of this feed varies a lot due to the many different sensor combinations found connected to the same board. As an example, suppose a light and temperature sensor combined into the same gateway. They are organised in a single physical component, which has three associated outputs: the battery life, the light intensity and the temperature readings. Multiple sensor combinations that are somehow logically related (e.g. geographically) can be supported in the

same feed simply by adding more components, one per gateway. For this reason it is best to have the geolocation per component and not at the main feed description.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Feed  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SMART_Datafeed_Schema_v0.3.xsd">
    <Id>santander_node200</Id>
    <Type>Physical</Type>
    <Title>Santander Sensor Feed</Title>
    <Description>The feed contains battery light temperature sensors</Description>
    <DescriptionTags>
        <TextTags>battery light temperature</TextTags>
        <URITags>smartfp7:battery smartfp7:light smartfp7:temperature</URITags>
    </DescriptionTags>
    <ContactInfo>
        <ContactEmail>tomasgarcia@ayto-santander.es</ContactEmail>
    </ContactInfo>
    <Components>
        <Physical>
            <Name>gateway_node_200</Name>
            <Description>Node 200 battery, light & temperature sensors</Description>
            <DescriptionTags>
                <TextTags>battery, light, temperature</TextTags>
                <URITags>smartfp7:battery, smartfp7:light, smartfp7:temperature</URITags>
            </DescriptionTags>
            <Geolocation>
                <Longitude>-3.80957</Longitude>
                <Latitude>43.46267</Latitude>
            </Geolocation>
            <Type>gateway</Type>
            <Exposure>Outdoor</Exposure>
            <Disposition>Fixed</Disposition>
        </Physical>
    </Components>
    <Outputs>
        <Output>
            <Name>battery</Name>
            <ProducedBy>gateway_node_200</ProducedBy>
            <Description>battery reading</Description>
            <DescriptionTags>
                <TextTags>battery life</TextTags>
                <URITags>smartfp7:battery_value</URITags>
            </DescriptionTags>
            <Type>double</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>light</Name>
            <ProducedBy>gateway_node_200</ProducedBy>
            <Description>light reading</Description>
            <DescriptionTags>
                <TextTags>light intensity</TextTags>
                <URITags>smartfp7:light_value</URITags>
            </DescriptionTags>
            <Type>double</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>temperature</Name>
            <ProducedBy>gateway_node_200</ProducedBy>
            <Description>temperature reading</Description>
            <DescriptionTags>
                <TextTags>temperature value</TextTags>
                <URITags>smartfp7:temperature_value</URITags>
            </DescriptionTags>
            <Type>double</Type>
            <Unit>Celcius degrees</Unit>
            <HasConfidence>false</HasConfidence>
        </Output>
    </Outputs>
</Feed>
```

7.2.8 Agenda feed

Only the two Santander edge nodes out of the total five operational by the end of year 2 of SMART have agenda feeds from the city council. The Year 2 version of this feed offers the agenda event description using multiple outputs organised in a virtual component.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Feed  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SMART_Datafeed_Schema_v0.3.xsd">
    <Id>santander_agenda</Id>
    <Type>Virtual</Type>
    <Title>Santander Cultural Agenda</Title>
    <Description>This feed contains data for each event on Santander city based on the city's
cultural agenda</Description>
    <DescriptionTags>
        <TextTags>agenda event santander</TextTags>
        <URITags>smartfp7:agenda smartfp7:event smartfp7:santander</URITags>
    </DescriptionTags>
    <ContactInfo>
        <ContactEmail>kmoumene@prisadigital.com</ContactEmail>
    </ContactInfo>
    <Components>
        <Virtual>
            <Name>agenda_event</Name>
            <Description>Event on the Agenda</Description>
            <Type>social</Type>
        </Virtual>
    </Components>
    <Outputs>
        <Output>
            <Name>id</Name>
            <ProducedBy>agenda_event</ProducedBy>
            <Description>Event ID</Description>
            <Type>string</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>name</Name>
            <ProducedBy>agenda_event</ProducedBy>
            <Description>Event Name</Description>
            <Type>string</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>latitude</Name>
            <ProducedBy>agenda_event</ProducedBy>
            <Description>Event GeoLocation - Latitude</Description>
            <Type>double</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>longitude</Name>
            <ProducedBy>agenda_event</ProducedBy>
            <Description>Event GeoLocation - Longitude</Description>
            <Type>double</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>location_name</Name>
            <ProducedBy>agenda_event</ProducedBy>
            <Description>Location Name of the Event</Description>
            <Type>string</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>category</Name>
            <ProducedBy>agenda_event</ProducedBy>
            <Description>Event Category</Description>
            <Type>string</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
```

```
        <Output>
            <Name>starttime</Name>
            <ProducedBy>agenda_event</ProducedBy>
            <Description>Event Start Time</Description>
            <Type>string</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>endtime</Name>
            <ProducedBy>agenda_event</ProducedBy>
            <Description>Event End Time</Description>
            <Type>string</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>modified</Name>
            <ProducedBy>agenda_event</ProducedBy>
            <Description>Event Last Modified Time</Description>
            <Type>string</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>description</Name>
            <ProducedBy>agenda_event</ProducedBy>
            <Description>Event Description</Description>
            <Type>string</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>link</Name>
            <ProducedBy>agenda_event</ProducedBy>
            <Description>Event Link</Description>
            <Type>string</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
    </Outputs>
</Feed>
```

7.2.9 Reasoning feed

Only the two Santander edge nodes out of the total five operational by the end of year 2 of SMART have reasoning feeds, albeit currently simple ones. The Year 2 version of this feed offers the probability of an event of interest as the output of a virtual component.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Feed  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SMART_Datafeed_Schema_v0.3.xsd">
    <Id>reasoning_feed_imperial</Id>
    <Type>Virtual</Type>
    <Title>reasoning_feed_imperial</Title>
    <Description>Notification regarding the type of an event</Description>
    <Components>
        <Virtual>
            <Name>alchemy_reasoner</Name>
            <Description>Reasoning on camera output</Description>
            <Type>reasoning</Type>
        </Virtual>
    </Components>
    <Outputs>
        <Output>
            <Name>notification_probability</Name>
            <ProducedBy>alchemy_reasoner</ProducedBy>
            <Type>double</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
    </Outputs>
</Feed>
```

7.2.10 Clip information feed

Only the two Santander edge nodes out of the total five operational by the end of year 2 of SMART have clip information feeds from the Media Data Manager. The Year 2 version of this feed offers information about all the recorded material as outputs of a virtual component.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Feed  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SMART_Datafeed_Schema_v0.3.xsd">
     <Id>simple_mdm_feed</Id>
     <Type>Virtual</Type>
     <Title>Santander Town Hall media clips</Title>
     <Description>Information about the clips recorded at the Santander Town Hall</Description>
     <Components>
          <Virtual>
               <Name>sdr_town_hall_cam1</Name>
               <Description>Recordings from camera 1</Description>
               <Type>clip_info</Type>
          </Virtual>
     </Components>
     <Outputs>
          <Output>
               <Name>event_id</Name>
               <ProducedBy>sdr_town_hall_cam1</ProducedBy>
               <Type>integer</Type>
               <HasConfidence>false</HasConfidence>
          </Output>
          <Output>
               <Name>camera</Name>
               <ProducedBy>sdr_town_hall_cam1</ProducedBy>
               <Type>string</Type>
               <HasConfidence>false</HasConfidence>
          </Output>
          <Output>
               <Name>length</Name>
               <ProducedBy>sdr_town_hall_cam1</ProducedBy>
               <Type>integer</Type>
               <HasConfidence>false</HasConfidence>
          </Output>
          <Output>
               <Name>filenames</Name>
               <ProducedBy>sdr_town_hall_cam1</ProducedBy>
               <Type>array(string)</Type>
               <HasConfidence>false</HasConfidence>
          </Output>
          <Output>
               <Name>clip_urls</Name>
               <ProducedBy>sdr_town_hall_cam1</ProducedBy>
               <Type>array(string)</Type>
               <HasConfidence>false</HasConfidence>
          </Output>
          <Output>
               <Name>clip_res_x</Name>
               <ProducedBy>sdr_town_hall_cam1</ProducedBy>
               <Type>array(integer)</Type>
               <HasConfidence>false</HasConfidence>
          </Output>
          <Output>
               <Name>clip_res_y</Name>
               <ProducedBy>sdr_town_hall_cam1</ProducedBy>
               <Type>array(integer)</Type>
               <HasConfidence>false</HasConfidence>
          </Output>
          <Output>
               <Name>clip_formats</Name>
               <ProducedBy>sdr_town_hall_cam1</ProducedBy>
               <Type>array(string)</Type>
               <HasConfidence>false</HasConfidence>
          </Output>
          <Output>
               <Name>thumbnail_urls</Name>
```

```
            <ProducedBy>sdr_town_hall_cam1</ProducedBy>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>thumbnail_res_x</Name>
            <ProducedBy>sdr_town_hall_cam1</ProducedBy>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>thumbnail_res_y</Name>
            <ProducedBy>sdr_town_hall_cam1</ProducedBy>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>thumbnail_formats</Name>
            <ProducedBy>sdr_town_hall_cam1</ProducedBy>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
    </Outputs>
</Feed>
```

7.2.11 Linked Data feed

The Linked Data Manager is a new component and no linked data feed is currently hosted in any edge node. The Year 2 version of this feed offers the linked data information as outputs of a single virtual component.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Feed  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="SMART_Datafeed_Schema_v0.3.xsd">
    <Id>ldm_feed</Id>
    <Type>Virtual</Type>
    <Title>Santander Town Hall Linked Data Manager</Title>
    <Description>Linked Data Information about Santander</Description>
    <Components>
        <Virtual>
            <Name>ldm_search1</Name>
            <Description>The result of a search on LDM. Search could be done by keywords
(textual) or geographic coordinates (geolocation) and report actions or venues</Description>
            <Type>ldm_result</Type>
        </Virtual>
    </Components>
    <Outputs>
        <Output>
            <Name>searched_item</Name>
            <ProducedBy>ldm_search1</ProducedBy>
            <Description>Whether the user searched for "venue" or "activity"</Description>
            <Type>string</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>search_type</Name>
            <ProducedBy>ldm_search1</ProducedBy>
            <Description>Whether the search was based on "textual" input or "geo-
search"</Description>
            <Type>string</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>search_coords</Name>
            <ProducedBy>ldm_search1</ProducedBy>
            <Description>Available in "geo-search" queries, coordinates that the user
provided, 4 elements (start_lon, start_lat, end_lon, end_lat) if a rectangle search, 3 elements
(center_lon, center_lat, radius_km) if search on a circle</Description>
            <Type>array(string)</Type>
```

```
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>keywords</Name>
            <ProducedBy>ldm_search1</ProducedBy>
            <Description>Available in "textual" queries, keywords that the user requested on
search</Description>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>key</Name>
            <ProducedBy>ldm_search1</ProducedBy>
            <Description>Object identification in linked data</Description>
            <Type>string</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>location</Name>
            <ProducedBy>ldm_search1</ProducedBy>
            <Description>Location list for an object</Description>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>location_long</Name>
            <ProducedBy>ldm_search1</ProducedBy>
            <Description>Longitude list of the found locations for venues/events</Description>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>location_lat</Name>
            <ProducedBy>ldm_search1</ProducedBy>
            <Description>Latitude list of the found locations for venues/events</Description>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>is_primary_topic_of</Name>
            <ProducedBy>ldm_search1</ProducedBy>
            <Description>List of web links (usually from wikipedia) for the
locations</Description>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>date</Name>
            <ProducedBy>ldm_search1</ProducedBy>
            <Description>Available in "activity" queries, date of the activity</Description>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>txt</Name>
            <ProducedBy>ldm_search1</ProducedBy>
            <Description>Main text for the locations</Description>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
        <Output>
            <Name>label</Name>
            <ProducedBy>ldm_search1</ProducedBy>
            <Description>Labels for the locations</Description>
            <Type>array(string)</Type>
            <HasConfidence>false</HasConfidence>
        </Output>
    </Outputs>
</Feed>
```

7.3 Accessing the repository

Two interfaces have been built to handle metadata and feeds. The direct access interface provides direct access to the CouchDB and uses the API of that specific database, therefore producing information only in JSON format. As it does not have intermediate processing stages, it is the fastest interface but it requires more effort from application developers. It also supports live streaming by keeping the HTTP connection open for a short time while waiting for new data. On the other hand, some operations such as feed creation require administrator privileges and are therefore not accessible via this interface for simple users.

The common interface is not database-specific, effectively hiding the database from the user and thus allowing the use of any database. It is based on Java servlets, allowing both XML and JSON document formats, as well as and human-readable time codes in the queries and data.

The two interfaces are detailed in our Trac documentation:

http://opensoftware.smartfp7.eu/projects/smart/wiki/Usage

Most of the feed manipulation processes can be carried out using both interfaces, but for each one there is an interface that is mostly suitable. There are two rules. Firstly, use the direct interface for retrieving information, since it is the only one that provides streaming:

- Discovering feeds: Retrieving information about the feeds. See: http://opensoftware.smartfp7.eu/projects/smart/wiki/EdgeNodeGetCommands#FeedList
- Retrieving metadata from feeds: Getting the payload. See: http://opensoftware.smartfp7.eu/projects/smart/wiki/EdgeNodeGetCommands#Retrieval

Secondly, use the common interface for sending information to the CouchDB, since it is the only interface allowing for error checking:

- Creating feeds: The common interface is very important for error checking. Managing the databases without needing administrative access and feed description document conversion from XML to JSON. See: http://opensoftware.smartfp7.eu/projects/smart/wiki/EdgeNodePostCommands#CreateFeed
- Appending into feeds: The common interface is very important for conformance checking against the feed description document. See: http://opensoftware.smartfp7.eu/projects/smart/wiki/EdgeNodePostCommands#Append

8 Configuration

8.1 Processing configuration

There are several parameters to configure regarding signal processing. Regarding visual processing, there is a set of things that need configuration at the edge node setup (that have to do with what to process in the frames) and some others that are configured online by the processing algorithms themselves.

8.1.1 Configuration at setup

8.1.1.1 Visual processing zones

The configuration into different visual processing zones is very important since the camera views different entities of the surroundings.

As an example, let's focus on the three zones of interest discussed in the example of section 7.2.2. There the camera views the Mercado de la Esperanza Square, but also the surrounding streets and sidewalks. These three areas are handled differently by the system, since visual density at the different areas has different interpretation:

- On the square: the market (or any other event) is on, crowds are causing the high measurements.
- On the sidewalks: a busy part of the day, crowds are causing the high measurements.
- On the streets: Traffic, vehicles are causing the high measurements.

Whenever there can be such different interpretations of visual density based on the particular part of the frame, a new processing zone should be configured.

Returning to the example of section 7.2.2, the three processing zones are shown in Figure 8.1.

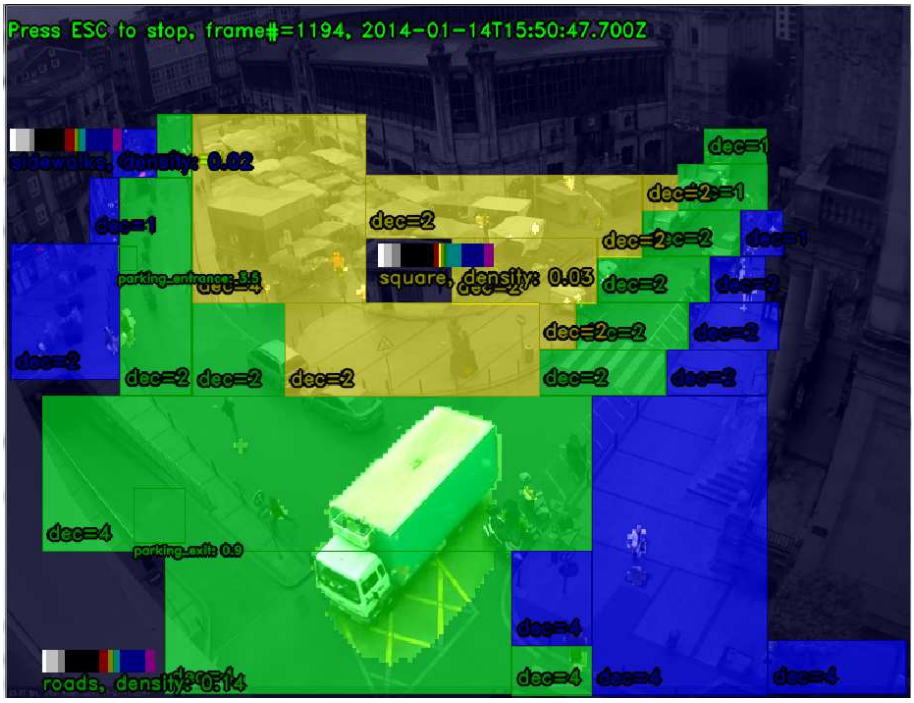

Figure 8.1: Configuration of the visual analysis component in the Mercado de la Esperanza Square (Santander, Spain) edge node. The three processing zones are sidewalks (blue), streets (green) and square (yellow). They comprise of 29 rectangles, each with its own decimation factor. The zone on streets hosts two object counting gates, at the parking entrance and exit (rectangles labelled parking_entrance and parking_exit respectively).

8.1.1.2 Gates for object count

At the processing zone of the streets, we are interested in counting the objects (vehicles) through two gates, the entrance and the exit of an underground parking lot. These are also shown in Figure 8.1.

8.1.1.3 Processing rectangles and decimation factors

The processing zones can be arbitrarily shaped. In the current implementation, the only restriction is that they need to be composed of (arbitrarily small) rectangles. These are called processing rectangles, as each of them can have its own visual processing parameters. Currently the only such parameter employed is the decimation factor. As a rule, this is unity for processing rectangles that are far away, and every pixel counts, and increases (usually as a power of two) as the rectangles approach the camera. This arrangement allows for the full frame to be processed and yet the system to run very fast, since only the necessary amount of pixels is processed by the foreground extractor. The motion and colour analyses on the other hand are performed on the full resolution.

8.1.1.4 Foreground extraction algorithm parameters

The foreground extraction algorithm parameters have been detailed in deliverable D3.2 [7]. All the parameters that are configured at setup are included in a configuration file. For the example of section 7.2.2 this file is listed below. D3.2 [7] and the comments in the configuration file are all that is needed to understand its content.

```
######################
#                    #
# General parameters #
#                    #
######################
#Note: In most of the below parameters the defaults are enclosed in parentheses after the
current value
[General]
# Minimum time in milliseconds between two consecutive posts to CouchDB, disabled when <=0
POST_FREQUENCY_MSEC=60000 (60000)
#Every how many frames to report timing statistics on the console
REPORT_STEP = 500 (500)
#Every how many frames to draw the output window
#negative values save images to disk instead of showing
DRAW_STEP = 2 (50)
# Generate output video
#WRITEVIDEO=1
# Number of threads to use. Rule of thumb: twice the number of threads in the CPU. Use 0 for
single-threaded version. (numThreads should be <= MAX_THREADS, currently 16)
#numThreads=4
# Number of channels to process, 1(grayscale) or 3(color)
#NCHANNELS = 1
# Skip this number of frames before starting processing of video
#START_FRAME=0 (0)
# Number of frames to skip at start (if conditions change due to mean intensity jump or
updateRate2() detection)
SKIP_FRAMES       = 0 (0)
# Number of additional frames for which only Stauffer is running, after conditions change
detection and SKIP_FRAMES have passed
#FRG_ONLY_FRAMES  = 20
# Record individual event videos
#RECORDEVENTS = 1
#temporary position
# How many frames to process to init background model
TRAIN_FRAMES=100 (100)
# Background learning rate used at first TRAIN_FRAMES
TRAIN_BKG_RATE=.08 (.08)
# Base background learning rate
BASE_BKG_RATE=.0004 (.0008)
######################
#                    #
# Processing Regions #
#                    #
######################
```

```
[Regions]
#
# Configure Regions of interest and processing zones
#
# Each region has a name, contains one or more rectangular processing zones and is configured
as follows:
# region_name=Desired name of region
# proc_rect=[Decimation factor] [Defining rectangle top left x] [Top left y] [Width] [Height]
# Optional zone1 description
# proc_rect=[Decimation factor] [Defining rectangle top left x] [Top left y] [Width] [Height]
# Optional zone2 description
#colour_pos after a region defines the position of the colour histogram, current valid values
are
#top-left, top-right, bottom-left, bottom-right, center
#if no value or incorrect value is defined, it defaults to top-left
region_name=roads
color_pos=bottom-left
proc_rect=1 133 94 31 55
proc_rect=1 617 106 56 31
proc_rect=1 594 137 80 40
proc_rect=2 100 149 64 189
proc_rect=2 563 177 87 40
proc_rect=2 523 217 100 40
proc_rect=2 164 257 83 82
proc_rect=2 504 257 100 42
proc_rect=2 471 298 113 40
proc_rect=4 31 338 487 135
proc_rect=4 140 472 307 125
proc_rect=4 447 555 71 43
region_name=sidewalks
color_pos=top-left
proc_rect=1 93 106 40 43
proc_rect=1 73 149 26 57
proc_rect=1 650 177 38 40
proc_rect=2 3 206 97 118
proc_rect=2 623 217 49 40
proc_rect=2 604 257 80 42
proc_rect=2 584 298 90 40
proc_rect=4 518 338 156 260
proc_rect=4 447 472 71 83
proc_rect=4 674 550 123 47
region_name=square
color_pos=center
proc_rect=4 164 94 154 164
proc_rect=2 317 146 246 56
proc_rect=2 563 146 32 32
proc_rect=2 394 201 129 58
proc_rect=2 523 201 40 18
proc_rect=2 246 257 226 82
proc_rect=2 471 258 33 40
# Rectangles for object counting, make sure they are inside a processing rectangle
count_rect=112 418 46 48 0.7 parking_exit
count_rect= 98 208 18 22 0.7 parking_entrance
###################################
#                                 #
# Adaptive Background parameters  #
#                                 #
###################################
[Background]
# How many frames to process to init background model
#TRAIN_FRAMES=100 (100)
# Background learning rate used at first TRAIN_FRAMES
#TRAIN_BKG_RATE=.08 (.08)
# Base background learning rate
#BASE_BKG_RATE=.0004 (.0008)
# Number of Gaussians in pixel model (NOG should be <= MAX_GAUSS, currently 10)
#NOG=5
# Minimum allowed Gaussian variance (MIN_VAR). It is amplified up to 3 times as Gaussian
becomes brighter.
#MIN_VAR=16 (16)
# Initial Gaussian variance (INIT_VAR)
#INIT_VAR=64
```

```
# Maximum distance to accept current pixel value in one of the Gaussians (MAX_DIST_GAUSS). It
is amplified up to 3 times as Gaussian becomes brighter.
#MAX_DIST_GAUSS=4 (4)
# Accumulated weight of Gaussians before the current, for the pixel to be considered foreground
(TAU)
#TAU=0.65
# Threshold for merging Gaussians (MERGE_THR)
#MERGE_THR=150
# Minimum Ratio of weight of the Gaussian immediately before the current over that of the
current for define the pixel to be considered foreground (WEIGHT_RATIO)
#WEIGHT_RATIO=1
###################################
#                                 #
# Colour modelling    parameters #
#                                 #
###################################
[Colour]
#How many milliseconds after video start we should start reporting colours
REPORT_TIME = 20000
#Colour model learning rate, at the beginning it is 1 and is slowly reduced to this value
COLOUR_RATE = 0.005
#Minimum foreground area ratio to consider colours of current frame and avoid stray pixel
errors
COLOUR_FRG_MIN = 0.05
#Maximum foreground area ratio to consider colours of current frame and avoid flickering errors
COLOUR_FRG_MAX = 0.75
```

The processing zone and object count gate definitions are not entered manually in this configuration file. We are in the process of finalising a graphical tool for the definition of the processing rectangles, their association with zones and parameters (decimation factor) and the object count rectangles.

8.1.2 Online configuration

8.1.2.1 Foreground learning rate

The only online configuration parameter used to date is the adaptive foreground learning rate. This has been extensively discussed in D3.2 [7].

8.2 Reasoning rule learning

8.2.1 Learning input: Annotated events and configuring decision factors

Configuring new reasoning rules is a two-step process:

- First the events are identified. This is the process of high-level annotation of the input (sensory and social) data. As a result, certain time instances are labelled as examples of the occurrence of some new event of interest. There should be enough instances of every event of interest (at least the number of observations should be higher than the number of different permutations of decision factors), and these are split into training and testing occurrences (e.g. at a 70/30 split).
- Then the decision factors are selected. These are the parameters that would define the probability of a certain event and are the metadata of our perceptual components (e.g. crowd density, audio level, tweet frequency, etc.). The decision factors are thus a subset of the available feeds in the edge node, selected based on their relevance to the event of interest. They are enumerated at the annotation instances and then the values are passed to the reasoning engine for learning the rules.

8.2.2 Learning mechanism

The learning engine (i.e. Alchemy) uses Markov Logic Networks (MLN) for both learning and inference. As mentioned in Section 6, MLN is based on unifying Logical and Statistical Artificial Intelligence. We have detailed the MLN in D4.2 [2]. For an extensive documentation on this please refer to [1].

Clearly, the accuracy of prediction will improve as the number of observations grows. Note that the learning engine can scale to thousands of observations which is sufficient for local reasoning (by local reasoning we mean the reasoning in the local area of the edge node as opposed to global reasoning

which happens at the search engine and uses the input from multiple or all the edge nodes).

8.2.3 Learning output: Weights

As mentioned after learning each permutation of the input decision factors will have a weight attached to it (e.g. Table 6.2) which will then be translated to a probability of an event type corresponding to that specific permutation (e.g. refer to Table 6.3). In the absence of annotated events, these weights can be configured manually.

9 Conclusions

This deliverable provides an end-to-end description of the edge node functionalities. It is obvious from the material presented that early into the third year that all the edge node modules are mature and are providing their share of the metadata necessary for supporting the applications at the local level.

Any important further modifications will be reported in the final version of the integration deliverable, D4.3.1, the second version of [4].

10 References

[1] S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 2007. http://alchemy.cs.washington.edu/

[2] SMART project, D.4.2, “Social Networks and Sensor Networks Integration”

[3] Domingos, Pedro, et al. "Unifying logical and statistical AI." *Proceedings of the Twenty-First National Conference on Artificial Intelligence*. 2006.

[4] SMART project, D.4.3, “Sensors and multimedia data knowledge representation”

[5] SMART project, D.4.1, “SMART Distributed Knowledge Base and Open Linked Data Mechanisms”

[6] SMART project, D.3.1, “Sensors and multimedia data knowledge representation”

[7] SMART project, D.3.2, “Video Signal Processing Prototypes”

[8] SMART project, D.5.2, “Report on Query Submission, Processing and Routing”