SEVENTH FRAMEWORK PROGRAMME

Networked Media

Specific Targeted Research Project

SMART

(FP7-287583)

Search engine for MultimediA environment generated contenT

D3.1 Sensors and multimedia data knowledge representation

Due date of deliverable: 01-10-2012

Actual submission date: 15-10-2012

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

Summary of the document

Code:	**D3.1: Sensors and multimedia data knowledge representation**
Last modification: **State:**	11-10-2012 Final
Participant Partner(s): **Author(s):** **Fragment:**	AIT, TELESTO, ATOS, IBM Aristodemos Pnevmatikakis (AIT), Christos Smailis (TELESTO), Nikolaos Katsarakis (AIT), John Soldatos (AIT), Josemi Garrido (ATOS), Zvi Kons (IBM); Elena Garrido Ostermann (ATOS) No
Audience:	☒ public ☐ restricted ☐ internal
Abstract:	A report on the representation of sensors and multimedia data based on the outcomes of task 3.1. The report documents the mechanisms selected in order to deal with multimedia data and metadata in SMART.
Keywords:	Sensor Streams, Data Feeds Modelling, XML Schema, JSON
References:	WP2 Deliverables D2.2 and D2.3 are background documents for the present one. D2.2 and D2.3 facilitate the (better) understanding of several of the technical choices that are illustrated in this document. DoW

Table of Contents

1 Executive Summary 4
1.1 Scope 4
1.2 Audience 4
1.3 Summary 4
1.4 Structure 5
2 Introduction 6
2.1 Different Types and Role of Data Feeds in SMART 6
2.2 Evaluation criteria for Feed and Feed Output Data Modelling Technologies 6
3 Sensor and multimedia modelling technologies 8
3.1 Modelling Feeds and Feed Output Data 8
3.2 State of the art in data description 8
3.3 The SMART approach 15
4 Modelling Feeds in SMART Edge Node 16
4.1 Feed Specification in SMART Edge Node 16
4.2 Comparative synopsis of sensor and multimedia modelling technologies 19
4.3 Example of a Physical Feed Definition 20
4.4 Example of a Mixed Feed Definition 21
4.5 Example of a Virtual Feed Definition 24
5 Feed Output Models used in SMART Edge Node 25
5.1 Encoding data extracted from Feeds 25
5.2 Physical Measurement Data modelling example 25
5.3 Audio/Video data modelling example modelling 26
5.4 Social Feed data modelling 26
5.5 Modelling other types of data 27
6 Conclusions 28
7 BIBLIOGRAPHY AND REFERENCES 29
8 ANNEXES 30
8.1 A: Feed Description XSD file 30
8.2 MPEG-7 compliant description of output metadata 33
8.3 SMART metadata in RDF triples 35

1 Executive Summary

1.1 Scope

The SMART search engine is designed to search information and identify patterns over arbitrary large numbers of sensor repositories, which are populated by data stemming from physical and virtual sensors. Within the SMART architecture, physical sensors, perceptual components (i.e. A/V signal processing algorithms) and social networks processing algorithms (i.e. the so-called social sensors) act as data and are referred to as Data Feeds. These sensor / algorithm pairs produce data and metadata, which will be accordingly become searchable by SMART multimedia search engine. The purpose of the present deliverable is to illustrate the modelling of Data Feeds within the SMART system and more specifically within the edge nodes of the SMART system. The modelling of these feeds emphasizes the simplicity of their management, while also boosting the openness of the SMART system. The latter openness is boosted on the basis of the ability to easily define and add new feeds. As part of the deliverable, we also illustrate different data modelling options that were explored prior to the selection and consolidation of the SMART data models for the various sensor feeds. Note that this deliverable is the first among a series of project documents that will define data modelling and representation in the scope of SMART system. This is due to the fact, that data related to sensor measurements are not only managed and stored in the edge server. Those data actually flow to the upper layers of the SMART system and are persisted at other points as well.

1.2 Audience

The modelling and representation of sensor data within SMART is of great interest to a number of stakeholders including:

- **Developers of the SMART open source project**: SMART open source developers will use the models and schemas presented in this document as part of their effort to provide the open source implementation of the SMART system. Data models described within this deliverable will be used in order to define and manage the sensor inputs to the edge server.
- **The Open Source Community**: The models and representation of this deliverable will serve as a basis for the definition of new sensor streams as inputs to the SMART system. Hence, open source community developers will benefit from the present document in their effort to define and attach new data feeds to the SMART system.
- **SMART project members (notably members working in WP4 and WP5 of the project)**: The present deliverable will provide valuable insights to all SMART project members in terms of how feeds and their outputs are modelled through the system. Such insights will be extremely useful for project members in charge of data access/representations in WP4/WP5.

1.3 Summary

The objective of this document is to describe the sensors and multimedia data knowledge representation within the SMART project. This document describes both the technical details of the data representation and the process which has led us to this choice.

Description of sensors and multimedia data is a growing topic which has advanced considerably in recent years. Our work here was influenced considerably by those advancements and tries to match state-of-art in respect to current standards and trends. On the same time we had to take into account other considerations which are specific to the SMART system such as the real-time operation, its open-source nature and the requirements of the different stack holders which are described in D2.1.

We start by presenting current state-of-art in sensor and sensors output information models. This includes SensorML which is an XML standard used for describing sensors and network of sensors. Next we describe the MPEG-7 standard which can describe media and media streams. We follow with RDF which is more general format for describing various resources and can also be applied for sensors and media, Mpeg-7 which is a standard for modelling audio-visual resources as well as three

multimedia ontologies based on MPEG 7.

The selected SMART approach is somewhat simplified version of SensorML which uses XML for describing the feed specifications. This approach was selected because it can reduce the overhead of more complex standards while still being able to give full description of the sensors and their data.

The edge-nodes are described as a system which extracts and stores data from multiple heterogeneous physical feeds (E.g. Sensors) or virtual feeds (the latter being the results of various processing units or software that processes data from social networks).

The output of each Data Feed can be sensor measurements, results from sensors processing algorithms or text from social components. Feeds have to encode their output data in a JSON Format, before it is sent to the Edge Node. However Feed output data that is already stored in the Edge Node is encoded in RDF before being sent to different layers of the architecture.

It is sometime easier to visualise and verify that those concepts can be applied to the real world by giving examples. Therefore, we provide several examples illustrating the definition of feeds with physical, virtual and mixed components. We also provide examples for the different data output encodings.

As can be seen from those examples, the SMART modelling formats are simple, yet powerful enough to give complete description both of the feeds and their data streams.

1.4 Structure

The document is structured as follows:

- Section 2 provides an introduction to the scope of the data models and related representation techniques that are described as part of this deliverable.
- Section 3 illustrates techniques for modelling sensors and data feeds considered by the project, along with the rationale of the final selection.
- Section 4 focuses on the presentation of the techniques for feeds modelling and definition, which include the metadata, that describe the characteristics of a Feed. These metadata refer to the way a Feed is described/represented in the SMART system.
- Section 5 focuses on the presentation of the models and techniques for sensor data modelling, which include the output data that are emitted by a sensor (physical or virtual) to the SMART edge node. The section concludes with some illustrative examples of modelling specific sensor data and metadata.
- Section 6 concludes the deliverable.

Note that the deliverable includes also an Appendix where several modelling examples based on different data modelling technologies are presented.

2 Introduction

Within SMART, every component that contributes data to the platform is considered a **Data Feed**. Therefore the Data Feed concept represents two main categories of data sources that can contribute data to the SMART Platform: **Physical Feeds** as well as **Virtual (software) Feeds**. Therefore in order to support data extraction from the aforementioned multiple heterogeneous Data Feeds, the SMART Platform needs methods that enable the formal description of their specifications as well as the data they produce.

2.1 Different Types and Role of Data Feeds in SMART

- Physical Feeds
  Hardware sensor components responsible for providing measurement data or detecting events in the physical world.
- Virtual Feeds (defined on the basis of the above)
  Virtual feeds are software components that use algorithms in order to produce output data that is emitted to a Smart Edge Node.
  E.g. Social Feed:
  Software components responsible for detecting events using data from Social Networks.
- Mixed Feeds
  Mixed Feeds contain both hardware sensors as well as software processing components
  E.g. Perceptual Components (A/V Processing):
  Components related to the classification of events that stem from audio or video data. Such data are processed by algorithms and together with the sensor / algorithm pair forms a mixed feed.

Methods that allow the formal description of data feed specifications and outputs are needed. Specifically we need a **Feed Model** that will allow the description of the specifications of each attached feed as well as a **Feed Output Model** that will allow the formal encoding of the measurements or events that were observed by each feed. The requirements for these models should satisfy are listed in sections 2.2 and 2.3.

2.2 Evaluation criteria for Feed and Feed Output Data Modelling Technologies

The following requirements are used as criteria for evaluating the reviewed modelling technologies in the next section. The following criteria were formed by keeping in mind the needs of developers, users and the design requirements of the SMART Platform.

- Simplicity (easy to define, parse, integrate)
  The simplicity of the Model is expected to increase its usability in terms of parsing and integration.
- Generality
  A Feed model should be able to handle the description of Feeds with very different specifications, since some of them will be hardware components, others will be software components, while some will include both hardware or software parts. On the other hand, a Feed Output Data model should be able to handle the description of measurements as well as events with very different specifications, since some of them will stem from a variety of highly heterogeneous data sources.
- Openness
  Is the Model freely available or is it propriety?
- Extensibility
  The Feed Model is expected to be easily extensible, if needed, in order to handle needs that arise, as the project evolves. The Feed Output Data model should be able to handle the

description of numerous types of measurements and events due to the high heterogeneity of feeds.

- Multimedia Orientation
  A Feed Model should be able to describe the specifications of Audio-Visual Sensors. Feed Output Data models should be able to handle the description of numerous types of measurements and events due to the high heterogeneity of feeds.
- Standard Compliance
  The selected Model should contain structures that provide compatibility with relevant market standards.

3 Sensor and multimedia modelling technologies

3.1 Modelling Feeds and Feed Output Data

In the scope of this section we present different technologies, which were considered for the modelling of Feeds and their output data within the SMART system. Note that the main objective of this deliverable is to provide open and extensible data models for both sensors and their data. Feed modelling is performed on the basis of metadata that describe the purpose and the structure of each data source. Feed models should be able to support physical, virtual and social sensors. At the same time, sensor data modelling refers to the modelling of data that are fed (from the sensors) to the edge nodes.

In following paragraphs we review alternative technologies considered for the implementation of Feed and Feed output modelling in SMART. The scoring formula we use, is based upon the criteria mentioned in section 2.2.

3.2 State of the art in data description

General Data Standards

XML

The XML (Extensible Mark-up Language) is mainly considered for modelling of output data produced by SMART Feeds. XML is a widespread data-interchange format, designed with human readability in mind.

Score

- Simplicity : 10
  XML is a widespread tool for modelling metadata as key value pairs in a formal manner.
- Generality:10
  It can be used to model any type of information
- Openness:10
  XML is free to use
- Extensibility:10
  XML can be accompanied by the use of XML Schema, a dialect used for creating schemas for validating XML Documents
- Standard Compliance:10
  XML is a standard.
- Multimedia Orientation: 7
  XML is a generic tool for data modelling and thus can be used for describing audio-visual content. However it does not provide any specialized structures for this task.
- Final Score: 9.5
  XML perfectly suits the requirements of feeds that need a simple and lightweight tool for encoding their output data

JavaScript Object Notation

The JSON (JavaScript Object Notation) is another option mainly considered for modelling of output data produced by SMART Feeds. JSON[1] is a lightweight data-interchange format, designed with human readability in mind. JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, structure, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

As mentioned in D2.3, "Multimedia Search Framework Open Architecture and Technical Specifications." due to providing a lightweight and formal way for describing metadata, JSON has been adopted by the SMART Architecture for encoding the output data of feeds that are attached to an Edge Node.

Score

- Simplicity : 10
  JSON is a minimal tool for modelling metadata as key value pairs in a formal manner.
- Generality:10
  It can be used to model any type of information
- Openness:10
  JSON is free to use
- Extensibility:10
  JSON can be accompanied by the use of JSON Schema, a dialect used for creating schemas for validating JSON Documents
- Standard Compliance:10
  JSON is a standard.
- Multimedia Orientation: 7
  JSON is a generic tool for data modelling and thus can be used for describing audio-visual content. However it does not provide any specialized structures for this task.
- Final Score: 9.5
  JSON perfectly suits the requirements of feeds that need a simple and lightweight tool for encoding their output data

Specific Data Standards

SensorML

Formal descriptions of the capabilities of sensor based systems, is a necessary prerequisite in order to make hardware detectable, accessible as well as reusable within multiple heterogeneous systems or networks. The aforementioned goal can be accomplished using SensorML. SensorML is an XML Schema based standard, for modelling sensor systems as well as the processing procedures that are associated with them. More specifically SensorML can make feasible, the modelling of sensor systems as well as non-physical processes by providing the means necessary for creating metadata, able to describe both (physical or non-physical) processes, as well as the order of their execution. Using this approach SensorML enables the creation of measurement genealogies while simplifying their on demand processing. Advantages acquired by the adoption of SensorML standard in a system include, the support it offers for tasking, observation and alert web services, simplifying the inclusion of self-organization related capabilities to the system and finally archiving sensor parameters.

[1] Description of the JSON format can be found at: http://www.json.org/

Basic structural elements of SensorML

- **Abstract Processes**
  The most important structural element of SensorML for describing processes is the class AbstractProcess. This class offers a general framework for modelling processes of any type. It includes the following processes that may be inherited from processes belonging to specialized categories:
  - Process Name
  - Description of the performed functionality
  - Inputs categorized either as observed events or incoming raw data depending on the nature of the process
  - Outputs, the nature of whom also depends on the type of the process
  - Process Parameters.

- **Non-Physical Processes**
  Mathematical procedures that are applied to data can be considered as non-physical processes. In SensorML individual non-physical processes may be modelled through the class Process Model which inherits and extends the AbstractProcess class by incorporating a property for explicitly describing each mathematical procedure. Complex non-physical processes can be represented through the class ProcessChain. Each chain processes, is in fact a collection of ProcessModels or other ProcessChains which are executed in a serial manner in order to produce an arithmetic result.

- **Physical processes**
  Physical Processes represent devices or complex sensor systems. The most basic class for describing physical processes within SensorML is Component. It is used for representing sensor devices (or physical components that cannot be divided to simpler concepts further). Complex sensor systems that may also include a variety of physical and non-physical processes are represented by the class System.

Score

- Simplicity : 5
  Although the general mechanics may seem simple SensorML contains a huge amount of extra elements that tend to complicate feed modelling
- Generality:9
  Although SensorML is a very general tool for encoding hardware as well as sensor systems in any level of detail, its verbose schema may complicate the low level description of some systems related to sensing such as cameras. However it can model both Physical, Virtual as well as Mixed feeds since these concepts are equivalent to the ProcessModel, Component and System classes of SensorML respectively.
- Openness:10
  SensorML is free to use
- Extensibility:6
  Although it is feasible, extending the original SensorML schemas requires an expert.
- Standard Compliance:10
  SensorML is an Open Geospatial Consortium standard.
- Multimedia Orientation: 8
  SensorML is a standard that mainly focuses on modelling non AV sensors, but can also model the specifications of AV sensors.
- Final Score: 8
  All in all SensorML is a good candidate for modelling the specifications of a feed in a formal manner. However it is quite verbose and difficult to extend.

MPEG-7

One of the options for modelling SMART Feeds and their output data is the MPEG-7 standard. This option was considered mainly due to the fact that SMART emphasizes on the inclusion of several audio and video signal processing algorithms as virtual sensors. The outputs of these algorithms and their interrelationships can be modelled using MPEG-7 [Puri11]. The latter standard incorporates standardized description schemes that are mainly targeting efficient retrieval of information on multimedia streams. Hence, MPEG-7 can be used to represent audio-visual information in various forms of media, such as pictures, 2D/3D models, audio, speech, and video [Martınez02]. It can also be used to capture both simple and more sophisticated representations of audio-visual metadata. In the Appendix of this document we provide a couple of MPEG-7 compliant descriptions of metadata stemming from visual processing algorithms developed by SMART partner AIT.

Despite its appropriateness for modelling multimedia data/metadata, MPEG-7 is not very appropriate for modelling non A/V sensors and social sensors, given that these sensors could not directly benefit from the media-related metadata structures of MPEG-7. MPEG-7 would be ideal for modelling a system that exploits solely audio-visual sensors and related signal processing algorithms. This is not the case however for SMART and its applications, which are expected to make heavy use of non-audio/visual sensors, various virtual sensors, as well as social networks information processing algorithms.

Note that as part of the evaluation of MPEG-7 as a potential solution to data streams representation, the consortium has created sample MPEG-7 documents on the basis of the outputs of the visual processing components of the partners. Specifically, the MPEG-7 compliant modeling of AIT's person tracker output is listed in the second annex of this document. Despite the appropriateness of MPEG-7 for modelling data and metadata of the visual processing components of the project, one has to note that the capabilities of the standard are only poorly explored in those cases. This is because the metadata of the visual processing components of the project does not capture/include the relationships between MPEG-7 multimedia objects. As already outlined, MPEG-7 was not finally selected mainly due to its inappropriateness for capturing the data of the non-audio/visual sensors of the project in an open and extensible way. Following paragraphs score the MPEG-7 modelling solution against the various criteria set as part of the format selection process.

Score

- Simplicity : 7
  Mpeg7 is a very complex tool for modelling multimedia related data
- Generality:2
  Mpeg7 can only be used for modelling audio-visual content
- Openness:10
  Mpeg7 is free to use
- Extensibility:5
  Extensive knowledge of XML Schema and multimedia modelling is required
- Standard Compliance:10
  Mpeg7 is a standardized.
- Multimedia Orientation: 10
  Mpeg7 is limited on modelling audio-visual related content.
- Final Score: 7.3
  MPEG 7 is restricted in modelling only audio-visual content, which it is unable to handle the issues caused by the highly heterogeneous output data produced by SMART Feeds.

Resource Description Format

The RDF (Resource Description Framework) is another option considered for data modelling of SMART Feeds and their output data. RDF is a legacy semantic web technology that is nowadays considered the primary option for the representation of LinkedData, which provide a set of best practices for publishing and interlinking structured data on the Web [Heath11]. RDF provides a single unifying data model for LinkedData, which provides globally unique identification of entities and enables different schemata to be used in parallel to represent data («Web-of-Data» concept). SMART could exploit information from the Web-of-Data cloud, such as Geonames (www.geonames.org), which is an open-license geographical database that publishes Linked Data about 8 million locations. Also, LinkedGeoData [Stadler12], provides a Linked Data conversion of data from the OpenStreetMap project, which includes information about more than 350 million spatial features. Locations in Geonames and LinkedGeoData are interlinked with corresponding locations in DBpedia, ensuring there is a core of interlinked data about geographical locations.

As illustrated in deliverables D2.3 and D4.1 of the project, SMART has ultimately adopted RDF for describing/modelling the (usually high-level) outputs of edge nodes of the system. Nevertheless, RDF has not been selected and used for modelling lower level sensor and sensor data that are attached/fed to the SMART edge nodes. The reason for this has mainly been the needs for simplicity and openness, which are boosted by the selection of simpler data structures, which however feature weaker semantics.

Score

- Simplicity : 10
  RDF is a minimal tool for modelling semantic metadata
- Generality:10
  RDF can be used to model any type of information
- Openness:10
  RDF is free to use
- Extensibility:10
  Is already extended by RDF Schema as well as OWL Full (which was designed to preserve some compatibility with RDF Schema)
- Standard Compliance:10
  RDF is a standard.
- Multimedia Orientation: 7
  RDF is a generic tool for data modelling and thus can be used for describing audiovisual content. However it does not provide any specialized structures for this task.
- Final Score: 9.5
  RDF is a very general tool for modelling triples of semantic metadata. It is useful for encoding feed output data when transferred between the higher levels of the SMART platform architecture, where structures that provide basic semantic interoperability are required.

Multimedia ontologies

Multimedia ontologies are useful mainly for construction semantic search engines. In the case of SMART, it is not an objective creating a complete semantic search engine. A description of some multimedia search engine and related technologies can be found in [Troncy11].

An example of multimedia search engine including a multimedia ontology can be Buscamedia [Busca12]. As part of the development efforts, Buscamedia developed a new multimedia ontology called the M3 (Multimedia, Multilingual, Multidomain). A common problem with ontologies is that are only valid for some specific domains. In the case of Buscamedia, the scope of the project was reduced to a subset of multimedia contents about live news, education and sports. Even for these reduced domains, the effort to create and populate the ontologies is not trivial.

A common problem with semantics search engines that hinder its adoption in practice is that the speed of semantics repositories is sometimes too slow for real time search or real time annotation. A common approach is to design the system using semantics, but using traditional applications for indexing and searching (as Lucene in Buscamedia). In practice, SMART is adopting a quite similar approach but using Terrier for indexing.

Some other popular multimedia ontologies, include among others the Visual Descriptor Ontology (VDO) [Simou05]and Multimedia Structure Ontology (MSO) which were developed in the scope of the Ace Media project using the Resource Description Framework Schema (RDFS) as well as the Core Ontology for MultiMedia (COMM) [Arndt07], which was developed using OWL DL. All the aforementioned ontologies focus on providing semantic formalizations of specific parts of the MPEG-7 which used xml to describe multimedia metadata and thus did not provided semantic interoperability. For the aforementioned reason these ontologies inherit the drawbacks of MPEG-7 (E.g. they are not appropriate for modelling non A/V sensors and social sensors as well as their output data). The main utility of these MPEG-7-related ontologies is the annotation of events in a semantic search engine, so they do not apply to SMART.

Additionally none of the aforementioned ontologies has been standardized.

Score

- Simplicity: 3
  They tend to be more complex than MPEG 7, since they include complex semantic capabilities. Understanding any ontology requires previous knowledge about semantics. This requirement can prevent the adoption by open source developers.
- Generality: 4
  Similar to MPEG 7 they can only be used to model audio-visual content. An ontology is valid only in its scope or domain.
- Openness: 9
  Most ontologies are free to use, but they can be subject to copyright laws, the same as source code.[2]
- Extensibility: 8
  Usually ontologies are generalized by extension. It is quite common for an ontology being an extension of another. On the other hand extension would require an ontology expert with very good MPEG-7 knowledge.
- Standard Compliance: 6
  None of the aforementioned ontologies is a standard; however the concepts they include can be mapped to MPEG 7 (but not SensorML or any equivalent modelling tool). Generally, ontologies are published but usually not standardized. Some well-known ontologies are standards by fact,

[2] http://www.benedict.com/Digital/Software/Ontology.aspx

but not formally.

- Multimedia Orientation: 10
  These ontologies focus on describing multimedia content.
- Final Score: 6.7
  The aforementioned multimedia ontologies inherit the disadvantages of MPEG-7(for being unable to model other types of data except multimedia content), while introducing complexities related to semantics, that are not necessary to the SMART Platform.

3.3 The SMART approach

In order to simplify the use of the SMART platform, despite of the multiple possible data sources and types, we avoided the verbose approach of SensorML and developed our own XML Schema that follows the generic approach of SensorML for describing Data Feeds, but simplifies it as presented in the following sections. After defining our model, we evaluate it, using the scoring formula introduced in section 2.2. Additionally we created very simple but flexible schemas for encoding both Feed Output measurements when emitted to the edge node as well as when transferred from the edge node to other important layers of the SMART Architecture. The SMART metadata come in RDF triples, and they are formatted either in JSON or in XML. We thus take advantage of the nice features of these data description standards.

4 Modelling Feeds in SMART Edge Node

The Smart Edge Node will be able to collect data related to the physical world, through a variety of sources. These sources consist of hardware and/or software feeds that will contribute their output data to the system. In SMART we model all the attached systems using the notion of a "Data Feed", which contains the description of all subsystems that need to interface with the Edge Node. Due to the high level of the data source heterogeneity, the SMART platform should benefit from a simple and coherent method of describing data feeds. This method should set the foundations for:

- Modelling structured metadata, describing certain properties of data feeds.
- Registering new heterogeneous feeds in a uniform manner.
- Retrieving information for a feed
- Understanding the semantics of the retrieved data.

In order to achieve the aforementioned goals we have developed an XML Schema for describing data feeds. As mentioned above, the choice for developing a new XML schema for this end was made in order to avoid the use of verbose and bloated similar approaches such as SensorML, which would complicate the development cycle by introducing extra unneeded features and would require additional processing power for real-time indexing. This XML schema defines a general class for describing feeds. Each feed is associated with a unique URI which is composed of the ID of the Edge Node as well as the ID of the feed (which is provided after registration). Moreover every feed should be accompanied by a title and a textual description of the function it performs. Also every feed automatically obtains a feed type (Physical / Virtual / Mixed), based on the components that are included in the feed. Obligatory properties of a feed also include descriptions of the components attached to the feed, as well as the outputs it produces. Each feed has also a small set of optional, but commonly used, properties which include: Geolocation, Description Tags and Contact Information. In order to enable modelling of feed components, the SMART Edge Node XML Schema incorporates an abstract component type which has two specializations: one for representing Physical Components as well as one for Virtual (Software) Components. Finally this XML Schema allows the definition of various outputs produced by the abovementioned components.. These outputs are characterised by properties such as the Type of the retrieved measurement data, the Measurement Unit (Optional), etc. The specification details of the SMART Edge Node XML Schema for describing a feed are listed below.

4.1 Feed Specification in SMART Edge Node

It should be noted that the specification below is slightly complex, in order to cover the broadest possible scenarios, and the creation of the described XML for feed registration can be tiresome and prone to errors. Therefore, a tool will be developed, providing help for the novice user and allowing him to easily generate valid feed descriptions.

Feed

Property	**Data Type**
Id	URI (e.g. EdgenodeId:unique# per EdgeNode)*, will be provided upon successful execution of the command to register a new feed.*
Type	*Will be inferred by the types of Components included in the feed. Type is restricted to the following values:* Physical (for feeds containing Temperature/humidity sensors) Virtual (for feeds related to twitter, other social media, multimedia processing algorithms) Mixed (for feeds that include both software and hardware parts e.g. video analysis feeds)
Title	Text (Feed Name)
Description	Text (Feed Description)
Description Tags (Optional)	URITags: Space-separated URIs TextTags: Space-separated Keywords (the special sequence %20 can be used to represent a keyword containing spaces)

Geolocation (Optional)	Longitude: double [-180,180) Latitude: double (-90,90) Elevation (optional): double
Contact Info (Optional)	WebSiteURL: URI ContactEmail: text, must have email address form, a@b.c
Components	List of **<Component>**
Outputs	List of **<Output>**

Component

The components have an abstract type, Physical or Virtual, described below.

Physical Component

Property	**Data Type**
Name	Text (Component Name, unique per feed, containing lowercase Latin characters (a-z), digits (0-9), or the underscore character _)
Description	Text (Component Description)
Description Tags (Optional)	URITags: Space-separated URIs TextTags: Space-separated Keywords (use %20 for spaces in keywords)
Geolocation (Optional)	Longitude: double [-180,180) Latitude: double (-90,90) Elevation (optional): double
Type	Text (e.g. camera, microphone, temperature/dust/vibration/gas sensor) It is suggested to restrict the type to a set of pre-defined values that will expand as the need arises.
Exposure	*One of the following:* Indoor Outdoor
Disposition	*One of the following:* Fixed Mobile
Serial Number (Optional)	Text
Part Number (Optional)	Text
Manufacturer (Optional)	Text
Working Status (Optional)	*One of the following:* ON OFF
Efficiency (Optional)	Double [0-1] (the accuracy of the component outputs)

Virtual Component

Property	Data Type
Name	Text (Component Name, unique per feed, containing lowercase Latin characters (a-z), digits (0-9), or the underscore character _)
Description	Text (Component Description)
Description Tags (Optional)	URITags: Space-separated URIs TextTags: Space-separated Keywords (use %20 for spaces in keywords)
Type	Text (e.g. Camera processing, mobile, twitter) As is the case with the physical components, it is suggested to restrict the type to a set of pre-defined values that will expand as the need arises.
Efficiency (Optional)	Double [0-1] (the accuracy of the component outputs)

Output

Property	Data Type
Name	Text (Output name, unique per component producing the output, containing lowercase Latin characters (a-z), digits (0-9), or the underscore character _)
Produced By	Text, corresponding to one of the defined components
Description	Text (Component Description)
Description Tags (Optional)	URITags: Space-separated URIs TextTags: Space-separated Keywords (use %20 for spaces in keywords)
Type	*One of the following:* boolean integer double string If a component produces an unknown number of outputs of a specific type, it can be represented with the notation "array(type)", where type is any of the previously defined.
Unit (optional)	Text (e.g. m/s, kg, Celcius degrees)
Has Confidence	Boolean (Whether the measurement provides confidence at each temporal instance)

The complete XSD code of the aforementioned schema is provided in Annex 8.1.

Score

- Simplicity : 10
  The SMART Feed Model only defines concepts useful within the scope of the project

- Generality:10
  This model was designed in order to provide an abstraction level that would eliminate the heterogeneity of the Data Feeds used within the SMART Project.
- Openness:10
  Similar to the rest of the SMART software components, the aforementioned model will also be free as open source software.
- Extensibility:10
  Extension would require basic knowledge of XML Schema.
- Standard Compliance:7
  The aforementioned model is not a standard; however the concepts it defines can be converted to equivalent concepts of SensorML (which is also a good but rather verbose tool for modelling data feeds)
- Multimedia Orientation: 7
  The SMART Feed Model is quite generic and can thus support the description of multimedia sensors
- Final Score: 9
  The SMART Feed Model was designed by keeping in mind the needs of both developers and users of the SMART Platform. Thus it is able to model any type of Data Feed using minimal concepts that mainly focus on associating feed components with the types of output data that they produce.

4.2 Comparative synopsis of sensor and multimedia modelling technologies

The following table presents a synopsis of the evaluation of the reviewed sensor and multimedia modelling technologies

	Simplicity	Generality	Openness	Extensibility	Standard Compliance	Multimedia Orientation	Final
SensorML	5	9	10	6	10	8	8
MPEG 7	7	2	10	5	10	10	7
Multimedia Ontologies	3	4	9	8	6	10	6.7
SMART Feed Model	10	10	10	10	7	7	9

4.3 Example of a Physical Feed Definition

An example for the definition of a physical feed comprised of a temperature and humidity sensor. Each sensor component produces a specific output as shown below. It should be noted that the URI description tags would be preferably limited within a pre-defined set, detailed in Deliverable 4.1, “SMART Distributed Knowledge Base and Open Linked Data” and shown for completeness of this document in Annex 8.3.

```
<?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>edgenodeName:temphumfeed</Id>
	<Type>Physical</Type>
	<Title>Temperature and Humidity Sensor Feed</Title>
	<Description>The feed contains an ARDUINO DUEMILANOVE, an LM335A Temperature
Sensor and a Relative Humidity Sensor </Description>
	<DescriptionTags>
		<TextTags>Temperature Humidity Arduino</TextTags>
		<URITags>smartfp7:temperature smartfp7:humidity</URITags>
	</DescriptionTags>
	<ContactInfo>
		<ContactEmail>a@b.c</ContactEmail>
	</ContactInfo>
	<Components>
		<Physical>
			<Name>temperature_sensor</Name>
			<Description>LM335A Temperature Sensor</Description>
			<DescriptionTags>
				<TextTags>Temperature Sensor Arduino</TextTags>
				<URITags>smartfp7:temperature</URITags>
			</DescriptionTags>
			<Geolocation>
				<Longitude>-139.8</Longitude>
				<Latitude>41.8</Latitude>
			</Geolocation>
			<Type>Thermometer</Type>
			<Exposure>Indoor</Exposure>
			<Disposition>Fixed</Disposition>
			<Efficiency>0.80</Efficiency>
		</Physical>
		<Physical>
			<Name>humidity_sensor</Name>
			<Description>Relative Humidity Sensor</Description>
			<DescriptionTags>
				<TextTags>Humidity Sensor Arduino</TextTags>
				<URITags>smartfp7:humidity</URITags>
			</DescriptionTags>
			<Geolocation>
				<Longitude>-139.8</Longitude>
				<Latitude>41.8</Latitude>
			</Geolocation>
			<Type>Humidity</Type>
			<Exposure>Indoor</Exposure>
			<Disposition>Fixed</Disposition>
			<Efficiency>0.90</Efficiency>
		</Physical>
	</Components>
```

```
        <Outputs>
                <Output>
                        <Name>temperature</Name>
                        <ProducedBy>temperature_sensor</ProducedBy>
                        <Description>Temperature Reading</Description>
                        <DescriptionTags>
                                <TextTags>Temperature Output Arduino</TextTags>
                                <URITags>smartfp7:temperature</URITags>
                        </DescriptionTags>
                        <Type>double</Type>
                        <Unit>Celcius degrees</unit>
                        <HasConfidence>false</HasConfidence>
                </Output>
                <Output>
                        <Name>humidity</Name>
                        <ProducedBy>humidity_sensor</ProducedBy>
                        <Description>Relative Humidity Reading</Description>
                        <DescriptionTags>
                                <TextTags>Humidity Output Arduino</TextTags>
                                <URITags>smartfp7:humidity</URITags>
                        </DescriptionTags>
                        <Type>double</Type>
                        <HasConfidence>false</HasConfidence>
                </Output>
        </Outputs>
</Feed>
```

4.4 Example of a Mixed Feed Definition

In this section a significantly more complex feed is described, containing both physical components such as a microphone and a camera, along with two virtual components, implementing crowd analysis in 2 different regions of the camera view.

It should be noted that in this example the crowd analysis component can provide the timestamp of the measurement, indicated as "**time**". The type of the timestamp is "**string**", therefore it should be human-readable, conforming to "xsd:dateTime" specification. Alternatively, a more compact and computer-friendly representation would be to have a type of "**integer**", indicating the number of **milliseconds** since 1 January 1970 (Unix epoch). The required millisecond precision is very important, as many components can provide metadata multiple times per second; typically a vision-based component processes 10-60 frames/sec. Furthermore, if needed it would be easy to include a different time for any of the other components, by just including an identically named output for them. However, since the two crowd components "calle_monasterio" and "plaza_ayuntamiento" receive data from the same camera, there was no need to include the same timestamp a second time.

Finally, the "colour" output of "calle_monasterio" component shows the use of the array() type. The resulting data stream representation in JSON format can be seen in section 5.3, where the compactness of this type becomes obvious.

```
<?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>edgenodeName:audiocrowdfeed</Id>
        <Type>Mixed</Type>
        <Title>Audio and Video processing Feed</Title>
        <Description>The feed represents a simple sound level meter and a more
elaborate crowd analysis component from camera </Description>
        <DescriptionTags>
                <TextTags>sound camera</TextTags>
                <URITags>smartfp7:audio smartfp7:video smarfp7:crowd</URITags>
```

```
        </DescriptionTags>
        <ContactInfo>
                <ContactEmail>a@b.c</ContactEmail>
        </ContactInfo>
        <Components>
                <Physical>
                        <Name>microphone</Name>
                        <Description>Sound level meter</Description>
                        <DescriptionTags>
                                <TextTags>sound</TextTags>
                                <URITags>smartfp7:audio</URITags>
                        </DescriptionTags>
                        <Geolocation>
                                <Longitude>-139.8</Longitude>
                                <Latitude>41.8</Latitude>
                        </Geolocation>
                        <Type>sound</Type>
                        <Exposure>Outdoor</Exposure>
                        <Disposition>Fixed</Disposition>
                        <Efficiency>0.90</Efficiency>
                </Physical>
                <Physical>
                        <Name>camera</Name>
                        <Description>AXIS camera covering 2 areas of interest, a park
and a public square</Description>
                        <DescriptionTags>
                                <TextTags>camera</TextTags>
                                <URITags>smartfp7:camera</URITags>
                        </DescriptionTags>
                        <Geolocation>
                                <Longitude>-139.823</Longitude>
                                <Latitude>41.812</Latitude>
                        </Geolocation>
                        <Type>camera</Type>
                        <Exposure>Outdoor</Exposure>
                        <Disposition>Fixed</Disposition>
                </Physical>
                <Virtual>
                        <Name>calle_monasterio</Name>
                        <Description>Crowd analysis for camera region focusing on road
(Calle de Jesus del Monasterio)</Description>
                        <DescriptionTags>
                                <URITags>smartfp7:crowd smartfp7:road</URITags>
                        </DescriptionTags>
                        <Type>crowd</Type>
                </Virtual>
                <Virtual>
                        <Name>plaza_ayuntamiento</Name>
                        <Description>Crowd analysis for camera region focusing on
public square (Plaza del Ayuntamiento)</Description>
                        <DescriptionTags>
                                <URITags>smartfp7:crowd smartfp7:square</URITags>
                        </DescriptionTags>
                        <Type>crowd</Type>
                </Virtual>
        </Components>
        <Outputs>
                <Output>
                        <Name>audio_level</Name>
                        <ProducedBy>microphone</ProducedBy>
                        <Description>Ambient Noise level</Description>
                        <DescriptionTags>
                                <URITags>smartfp7:sound</URITags>
```

```
                </DescriptionTags>
                <Type>double</Type>
                <Unit>dB</Unit>
                <HasConfidence>false</HasConfidence>
            </Output>
            <Output>
                <Name>exposure</Name>
                <ProducedBy>camera</ProducedBy>
                <Description>Exposure time</Description>
                <DescriptionTags>
                    <URITags>smartfp7:exposure</URITags>
                </DescriptionTags>
                <Type>double</Type>
                <Unit>ms</Unit>
                <HasConfidence>false</HasConfidence>
            </Output>
            <Output>
                <Name>time</Name>
                <ProducedBy>calle_monasterio</ProducedBy>
                <Description>Measurement timestamp</Description>
                <DescriptionTags>
                    <URITags>smartfp7:time</URITags>
                </DescriptionTags>
                <Type>string</Type>
                <HasConfidence>false</HasConfidence>
            </Output>
            <Output>
                <Name>density</Name>
                <ProducedBy>calle_monasterio</ProducedBy>
                <Description>crowd density from public square</Description>
                <DescriptionTags>
                    <URITags>smartfp7:crowd_density</URITags>
                </DescriptionTags>
                <Type>double</Type>
                <HasConfidence>true</HasConfidence>
            </Output>
            <Output>
                <Name>colour</Name>
                <ProducedBy>calle_monasterio</ProducedBy>
                <Description>The most prominent colours, listed as XXXX.YYY,
where XXXX is the colour number encoded as RR*256^2 + GG*256 + BB and YYY is the
percentage of that colour in the image</Description>
                <DescriptionTags>
                    <URITags>smartfp7:colour</URITags>
                </DescriptionTags>
                <Type>array(double)</Type>
                <HasConfidence>true</HasConfidence>
            </Output>
            <Output>
                <Name>density</Name>
                <ProducedBy>plaza_ayuntamiento</ProducedBy>
                <Description>crowd density from park</Description>
                <DescriptionTags>
                    <URITags>smartfp7:crowd_density</URITags>
                </DescriptionTags>
                <Type>double</Type>
                <HasConfidence>true</HasConfidence>
            </Output>
        </Outputs>
</Feed>
```

4.5 Example of a Virtual Feed Definition

This feed is a software component responsible for processing Twitter data stemming from the Social Network Manager, in order to extract dates and times related to the specified query term.

```
<?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>edgenodeName:socialfeed</Id>
	<Type>Virtual</Type>
	<Title>Social Feed</Title>
<Description>
This feed processes data from the Social Network Manager and emits results to the
Edge Node.
</Description>
	<DescriptionTags>
		<TextTags>social</TextTags>
		<URITags>smartfp7:social</URITags>
	</DescriptionTags>
	<ContactInfo>
		<ContactEmail>a@b.c</ContactEmail>
	</ContactInfo>
	<Components>
		<Virtual>
			<Name>twitter_time_extractor</Name>
			<Description>Social demo</Description>
			<Type>social</Type>
		</Virtual>
	</Components>
	<Outputs>
		<Output>
			<Name>is_active</Name>
			<ProducedBy>twitter_time_extractor</ProducedBy>
			<Description>active or not</Description>
			<Type>boolean</Type>
			<HasConfidence>false</HasConfidence>
		</Output>
		<Output>
			<Name>has_date</Name>
			<ProducedBy>twitter_time_extractor</ProducedBy>
			<Description>event date</Description>
			<Type>string</Type>
			<Unit>ms</Unit>
			<HasConfidence>false</HasConfidence>
		</Output>
		<Output>
			<Name>has_temporal_hint</Name>
			<ProducedBy>twitter_time_extractor</ProducedBy>
			<Description>list of temporal hints</Description>
			<Type>array(string)</Type>
			<HasConfidence>false</HasConfidence>
		</Output>
	</Outputs>
</Feed>
```

5 Feed Output Models used in SMART Edge Node

Unlike the Feed Model which describes the attached Feeds to the SMART Edge Nodes, the Feed Output Models are responsible for transferring data between the different components and layers of the SMART architecture.

5.1 Encoding data extracted from Feeds

In order to make the platform attractive to novice programmers as well as relieving feed developers from the burden of encoding data in a heavyweight schema, input from attached feeds should be encoded using simple JSON text encoding in key-value pairs, according to the following format:

```
{
	time: "timestamp in xsd:datetime or milliseconds since Unix epoch",
(optional)
	component_name_1 : {
		output_name_1 : "string value of output 1",
		output_name_2 : 1234,
		output_name_3 : true,
		output_name_N : "string value of output N"
	},
	component_name_2 : {
		output_name_1 : "string value with spaces",
		output_name_2 : 456.78,
		output_name_M : "string value of output M"
	}
}
```

Output names are expected to correspond to the output names specified during the feed description that was submitted to the system during the feed registration process. The Edge Node can handle the parsing of output values using the description of their type that is provided by the Output Class of the Feed XML Schema.

5.2 Physical Measurement Data modelling example

An example instance of the Feed Output Model that is used for encoding measurements from a simple temperature and humidity feed which is described in section 4.2, can be represented as follows:

```
{
	"temperature_sensor" : {
		"temperature" : 28.1
	},
	"humidity_sensor" : {
		"humidity" : 64.2
	}
}
```

The above format is very easy to implement, even for embedded devices with low capabilities

5.3 Audio/Video data modelling example modelling

Data produced by the complex feed, described in section 4.4 can be **represented** as follows:

```
{
	"microphone" : {
		"audio_level" : 24,
	},
	"camera" : {
		"exposure" : 10,
	},
	"calle_monasterio" : {
		"density" : 0.170000,
		"motion_horizontal" : 0,
		"motion_vertical" : 0,
		"motion_spread" : 2.200000,
		"colour" : [32.125100, 2113632.106200, 2113600.092100,
4218976.066300, 4210752.061800, 2105376.042200, 8256.036100, 2105408.033300,
8413280.030500, 8224.024800, 0.023300, 10535136.022300, 6316160.021400,
4210784.017800, 2113664.017800, 12640480.017000]
	},
	"plaza_ayuntamiento" : {
		"density" : 0.250000
	},
	"audio_analysis" : {
		"crowd_probability" : 0.9,
		"traffic_probability" : 0.1,
		"music_probability" : 0.3,
		"applause_probability" : 0.02
	}
}
```

It can be seen that, even though the feed was much more complicated, the output data representation remains relatively simple. The use of "array" allows for easy submission of an arbitrary number of values. Furthermore, it can be seen that the "**density**" outputs for which the tag "**HasConfidence**" was true are accompanied by the tag "**density_confidence**". If the output "**colour**" had the tag "**HasConfidence**" enabled, we would retrieve a similarly-sized array of "**colour_confidence**", comprised of numbers in the range [0,1].

5.4 Social Feed data modelling

In order to detect certain types of events the SMART platform will utilize data from Social Networks . Social events will be detected by exploiting feeds created specifically for this purpose. To relieve feed developers from the burden of accessing multiple Social Networks through heterogeneous APIs, each edge node will provide **searching functionalities for multiple social networks through** a unified interface known as the Social Network Manager. After extracting data from the Social Network Manager, each social feed will perform extra processing in order to transform data into valuable output information that should be inserted to the edge node. At this point it should be stressed that social feeds do not require any different handling than any other conventional feed. Therefore the data they produce should also be encoded in the JSON encoding defined in section 5.1.

An example of the output data produced by a social feed corresponding to the virtual feed definition of section 4.5 is given below:

```
{
	"twitter_time_extractor" : {
		"isActive" : true,
		"hasDate" : "26-07-2012",
		"hasTemporalHint" : ["10:45", "10:45", "12:15", "10:00"]
	}
}
```

5.5 Modelling other types of data

Transferring data between different layers of Smart

In order to enable integration of semantic web technologies within the SMART project, data that is transferred between different layers Smart Architecture, is encoded in RDF triplets that have the form: [subject, predicate, object].

The RDF can be presented in XML as follows:

```
<subject>
	<predicate1>object1</predicate1>
	<predicate2>object2</predicate2>
	...
	<predicateN>objectN</predicateN>
</subject>
```

To facilitate reporting multiple similar measurements, we can indicate that they are produced from similar components with different IDs, as defined in the feed description. In RDF representation they can be placed in groups with different IDs as tag attributes. For example:

```
<measurement>
	<crowd ID="calle_monasterio">
		<density>.03</density>
	</crowd>
	<crowd ID="plaza_ayuntamiento">
		<density>.28</density>
	</crowd>
</measurement>
```

A detailed description of the various possible tags can be found in annex 8.3.

6 Conclusions

The SMART architecture (introduced in deliverable D2.3) specifies the SMART search systems as multi-level multi-layer systems, which enable scalable indexing and retrieval over distributed repositories of multimedia environment generated content. These multi-layer systems involve several points where data and knowledge is modeled and stored. Hence, information delivery in the scope of the SMART search engine relies on the modeling of environment generated data and their subsequent transformations to the various formats specified in the various layers of the system. In this deliverable we have focused on the description of the formats and techniques selected for representing data and metadata at the lower-levels of the edge node of the SMART system i.e. at the level where sensors feed the SMART system with data. In particular, this deliverable introduces the techniques used in SMART for modelling sensor specifications, sensor output as well as multimedia related content, within the various layers of the SMART Architecture.

As part of the sensor data and metadata modeling specification process, we first reviewed several methods for encoding and representing sensors/feeds specifications and output data, using a scoring formula for evaluation purposes. The scoring formula was used to perform a comparative evaluation of various options including standards-based options such as the use of MPEG-7, JSON, SensorML and multimedia ontologies. Special emphasis was paid into the requirements for open, modular and extensible feeds modeling, which were taken into account during the scoring formula development and use processes. Along with the scoring process, we have also identified and elaborated the pros and cons of the considered solutions.

On the basis of the evaluation, we justified the use of a simple custom model for modelling the specifications of each Feed. The choice of such a model is not in-line with popular media standards (such as MPEG), but it is compliant to the design and implementation of on-line cloud platforms that aggregate sensor data and metadata (including user generated data/content). Moreover we specified how output data should be modelled when emitted from a feed to an Edge Node (using JSON) as well as when transferred from an Edge Node, to higher layers of the SMART Architecture (using RDF). The use of these models was presented using real world examples, including practical examples associated with the sensors/feeds that will be used in the scope of the validation and demonstration of the SMART search system. In a nutshell emphasis was given on selecting generic tools that would both satisfy the requirements of the SMART Platform for syntactic interoperability, generality as well as simplicity, without sacrificing semantic interoperability at the layers it is required.

The present deliverable is the first of series of deliverables that will deal with data modeling and knowledge representation at the different layers of the SMART system. Each of these deliverables will focus on the data modeling requirements at the specific layer that it deals with. Hence, the present deliverable has emphasized simplicity, openness and modularity with a view to facilitating the addition of data feeds into the SMART system. Other deliverables will pay more emphasis in semantic power and the possibility of reasoning, which are extensively dealt in WP4 of the project. Note that the contents of this deliverable will drive the implementation of extensible feeds modelling and management mechanisms, as part of the edge node implementation in the scope of the SMART open source project (available at: opensoftware.smartfp7.eu).

7 BIBLIOGRAPHY AND REFERENCES

[Arndt07] Arndt R., Troncy R., Staab S., Hardman L.,Miroslav V. COMM: designing a well-founded multimedia ontology for the web. 6th International Semantic Web Conference (ISWC'2007) pp. 30-43. Busan, 2007

[Borden11] Pachube Internet of Things "Bill of Rights" Ed Borden, in pachube.com (2011)

[Busca12] Buscamedia Consortium, D3.3.1 Formalización e implementación de la ontología (Formalization and implementation of the ontology) , www.cenitbuscamedia.es/index.php/documentacion.html

[Heath11] Tom Heath and Christian Bizer (2011) Linked Data: Evolving the Web into a Global Data Space (1st edition). Synthesis Lectures on the Semantic Web: Theory and Technology, 1:1, 1-136. Morgan & Claypool.

[Hellmann09] Sebastian Hellmann et. Al., «Linkedgeodata - adding a spatial dimension to the web of data», In Proceedings of the International Semantic Web Conference, 2009. http://dx.doi.org/10.1007/978-3-642-04930-9_46DOI: 10.1007/978-3-642-04930-9_46

[Martınez02] Martınez, J.:MPEG-7: Overview of MPEG-7 Description Tools, Part 2. IEEE MultiMedia 9(3), 83–93 (2002)

[Mislove10] A. Mislove, “Pulse of the nation: U.S. mood throughout the day inferred from twitter,” 2010.

[Motoyama10] M. Motoyama, B. Meeder, K. Levchenko, G. M. Voelker, and S. Savage, “Measuring online service availability using twitter.” Workshop on online social networks, Boston, Massachusetts, USA, 2010.

[OGC] Sensor Model Language (SensorML), The Open Geospatial Consortium, http:// www.opengeospatial.org/standards/sensorml/.

[Puri01] Shih-Fu Chang Puri, A. Sikora, T. Hongjiang Zhang, “Introduction to the special issue on MPEG-7”, IEEE Transactions on Circuits and Systems for Video Technology, Volume: 11 Issue: 6, June 2011, pp. 685 – 687

[Sakaki10] T. Sakaki, M. Okazaki, and Y. Matsuo, “Earthquake shakes twitter users: real-time event detection by social sensors,” in 19th international conference on World wide web, Raleigh, North Carolina, USA, 2010.

[Stadler12] Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer: LinkedGeoData: A Core for a Web of Spatial Open Data, Semantic Web Journal, 2012

[Simou05] Simou, N., Tzouvaras, V., Avrithis, Y., Stamou, G., Kollias S.. A visual descriptor ontology for multimedia reasoning. Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS), 2005

[Troncy11] Troncy R, Huet, B, Schenk, S (editors) Multimedia semantics : metadata, analysis and interaction. 2011 John Wiley & Sons Ltd. Chichester, United Kingdom

8 ANNEXES

8.1 A: Feed Description XSD file

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

	<xs:simpleType name="FeedCategory">
		<xs:restriction base="xs:string">
			<xs:enumeration value="Physical"/>
			<xs:enumeration value="Virtual"/>
			<xs:enumeration value="Mixed"/>
		</xs:restriction>
	</xs:simpleType>

	<xs:simpleType name="ExposureType">
		<xs:restriction base="xs:string">
			<xs:enumeration value="Indoor"/>
			<xs:enumeration value="Outdoor"/>
		</xs:restriction>
	</xs:simpleType>

	<xs:simpleType name="DispositionType">
		<xs:restriction base="xs:string">
			<xs:enumeration value="Fixed"/>
			<xs:enumeration value="Mobile"/>
		</xs:restriction>
	</xs:simpleType>

	<xs:simpleType name="WorkingStatusType">
		<xs:restriction base="xs:string">
			<xs:enumeration value="ON"/>
			<xs:enumeration value="OFF"/>
		</xs:restriction>
	</xs:simpleType>

	<xs:simpleType name="StringList">
		<xs:list itemType="xs:string"/>
	</xs:simpleType>

	<xs:simpleType name="IntegerList">
		<xs:list itemType="xs:integer"/>
	</xs:simpleType>

	<xs:simpleType name="URIList">
		<xs:list itemType="xs:anyURI"/>
	</xs:simpleType>

	<xs:element name="DescriptionTagsList" type="xs:anySimpleType" abstract="true"/>
	<xs:element name="URITags" type="URIList" substitutionGroup="DescriptionTagsList"/>
	<xs:element name="TextTags" type="StringList" substitutionGroup="DescriptionTagsList"/>

	<xs:simpleType name="LongitudeDouble">
		<xs:restriction base="xs:double">
			<xs:minInclusive value="-180.0"/>
			<xs:maxExclusive value="180.0"/>
		</xs:restriction>
	</xs:simpleType>

	<xs:simpleType name="LatitudeDouble">
		<xs:restriction base="xs:double">
			<xs:minExclusive value="-90.0"/>
			<xs:maxExclusive value="90.0"/>
		</xs:restriction>
	</xs:simpleType>

	<xs:simpleType name="perCentDouble">
		<xs:restriction base="xs:double">
			<xs:minInclusive value="0.0"/>
			<xs:maxInclusive value="1.0"/>
		</xs:restriction>
```

```
</xs:simpleType>

<xs:simpleType name="emailAddress">
	<xs:restriction base="xs:string">
		<xs:pattern value="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"/>
	</xs:restriction>
</xs:simpleType>

<xs:simpleType name="NameType">
	<xs:restriction base="xs:string">
		<xs:pattern value="([a-z_0-9])+"/>
	</xs:restriction>
</xs:simpleType>

<xs:simpleType name="OutputDataType">
	<xs:restriction base="xs:string">
		<xs:enumeration value="boolean"/>
		<xs:enumeration value="integer"/>
		<xs:enumeration value="double"/>
		<xs:enumeration value="string"/>
		<xs:enumeration value="array(boolean)"/>
		<xs:enumeration value="array(integer)"/>
		<xs:enumeration value="array(double)"/>
		<xs:enumeration value="array(string)"/>
	</xs:restriction>
</xs:simpleType>

<xs:complexType name="GeolocationType">
	<xs:sequence>
		<xs:element name="Longitude" type="LongitudeDouble"/>
		<xs:element name="Latitude" type="LatitudeDouble"/>
		<xs:element name="Elevation" type="xs:double" minOccurs="0"/>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="ContactInfoType">
	<xs:sequence>
		<xs:element name="Website" type="xs:anyURI" minOccurs="0"/>
		<xs:element name="ContactEmail" type="emailAddress"/>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="PhysicalComponent">
	<xs:sequence>
		<xs:element name="Name" type="NameType"/>
		<xs:element name="Description" type="xs:string"/>
		<xs:element name="DescriptionTags" minOccurs="0">
			<xs:complexType>
				<xs:sequence>
					<xs:element ref="DescriptionTagsList" maxOccurs="2"/>
				</xs:sequence>
			</xs:complexType>
		</xs:element>
		<xs:element name="Geolocation" type="GeolocationType" minOccurs="0"/>
		<xs:element name="Type" type="xs:string"/>
		<xs:element name="Exposure" type="ExposureType"/>
		<xs:element name="Disposition" type="DispositionType"/>
		<xs:element name="SerialNumber" type="xs:string" minOccurs="0"/>
		<xs:element name="PartNumber" type="xs:string" minOccurs="0"/>
		<xs:element name="Manufacturer" type="xs:string" minOccurs="0"/>
		<xs:element name="WorkingStatus" type="WorkingStatusType" minOccurs="0"/>
		<xs:element name="Efficiency" type="perCentDouble" minOccurs="0"/>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="VirtualComponent">
	<xs:sequence>
		<xs:element name="Name" type="NameType"/>
		<xs:element name="Description" type="xs:string"/>
		<xs:element name="DescriptionTags" minOccurs="0">
			<xs:complexType>
				<xs:sequence>
					<xs:element ref="DescriptionTagsList" maxOccurs="2"/>
				</xs:sequence>
			</xs:complexType>
		</xs:element>
		<xs:element name="Type" type="xs:string"/>
		<xs:element name="Efficiency" type="perCentDouble" minOccurs="0"/>
	</xs:sequence>
</xs:complexType>
```

```
	<xs:complexType name="OutputType">
		<xs:sequence>
			<xs:element name="Name" type="NameType"/>
			<xs:element name="ProducedBy" type="NameType"/>
			<xs:element name="Description" type="xs:string" minOccurs="0"/>
			<xs:element name="DescriptionTags" minOccurs="0">
				<xs:complexType>
					<xs:sequence>
						<xs:element ref="DescriptionTagsList" maxOccurs="2"/>
					</xs:sequence>
				</xs:complexType>
			</xs:element>
			<xs:element name="Type" type="OutputDataType"/>
			<xs:element name="Unit" type="xs:string" minOccurs="0"/>
			<xs:element name="HasConfidence" type="xs:boolean"/>
		</xs:sequence>
	</xs:complexType>

	<xs:element name="ComponentsType" type="xs:anyType" abstract="true"/>
	<xs:element name="Physical" type="PhysicalComponent" substitutionGroup="ComponentsType"/>
	<xs:element name="Virtual" type="VirtualComponent" substitutionGroup="ComponentsType"/>

	<xs:complexType name="FeedType">
		<xs:sequence>
			<xs:element name="Id" type="xs:anyURI"/>
			<xs:element name="Type" type="FeedCategory"/>
			<xs:element name="Title" type="xs:string"/>
			<xs:element name="Description" type="xs:string"/>
			<xs:element name="DescriptionTags" minOccurs="0">
				<xs:complexType>
					<xs:sequence>
						<xs:element ref="DescriptionTagsList" maxOccurs="2"/>
					</xs:sequence>
				</xs:complexType>
			</xs:element>
			<xs:element name="Geolocation" type="GeolocationType" minOccurs="0"/>
			<xs:element name="ContactInfo" type="ContactInfoType" minOccurs="0"/>
			<xs:element name="Components" minOccurs="0">
				<xs:complexType>
					<xs:sequence>
						<xs:element ref="ComponentsType"
maxOccurs="unbounded"/>
					</xs:sequence>
				</xs:complexType>
			</xs:element>
			<xs:element name="Outputs">
				<xs:complexType>
					<xs:sequence>
						<xs:element name="Output" type = "OutputType"
maxOccurs="unbounded"/>
					</xs:sequence>
				</xs:complexType>
			</xs:element>
		</xs:sequence>
	</xs:complexType>

	<xs:element name="Feed" type="FeedType" />

</xs:schema>
```

8.2 MPEG-7 compliant description of output metadata

The following sections contain the MPEG-7 compliant description of output metadata stemming from AIT's visual processing feed.

MPEG-7 – body tracker output

This is an example of the body tracker output in a frame with three targets.

```
<StillRegion id="SRimg_1225116711.549879.jpg">
   <StructuralUnit href="D:\Frames\img_1225116711.549879.jpg">
          <Name>Frame</Name>
   </StructuralUnit>
   <SpatialDecomposition>
          <StillRegion id="SR0fromSRimg_1225116711.549879.jpg">
                 <StructuralUnit href="D:\Frames\img_1225116711.549879.jpg">
                        <Name>Target with in the frame</Name>
                 </StructuralUnit>
                 <Semantic>
                        <Label>
                               <Name>Relative location of the object</Name>
                        </Label>
                        <Relation              source="#Object300"              target="#Table"
type="urn:mpeg:mpeg7:cs:SpatialRelationCS:2001:Near"/>
                        <SemanticBase id="Object300" xsi:type="ObjectType">
                               <Label>
                                      <Name>Object300</Name>
                               </Label>
                        </SemanticBase>
                 </Semantic>
                 <SpatialLocator>
                        <Box mpeg7:dim="2 2" xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001">
                               300 294 30 20
                        </Box>
                 </SpatialLocator>
          </StillRegion>
          <StillRegion id="SR1fromSRimg_1225116711.549879.jpg">
                 <StructuralUnit href="D:\Frames\img_1225116711.549879.jpg">
                        <Name>Target with in the frame</Name>
                 </StructuralUnit>
                 <Semantic>
                        <Label>
                               <Name>Relative location of the object</Name>
                        </Label>
                        <Relation              source="#Object301"              target="#Room"
type="urn:mpeg:mpeg7:cs:SpatialRelationCS:2001:Inside"/>
                        <SemanticBase id="Object301" xsi:type="ObjectType">
                               <Label>
                                      <Name>Object301</Name>
                               </Label>
                        </SemanticBase>
                 </Semantic>
                 <SpatialLocator>
                        <Box mpeg7:dim="2 2" xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001">
                               55 356 100 10
                        </Box>
                 </SpatialLocator>
          </StillRegion>
          <StillRegion id="SR2fromSRimg_1225116711.549879.jpg">
                 <StructuralUnit href="D:\Frames\img_1225116711.549879.jpg">
                        <Name>Target with the frame</Name>
                 </StructuralUnit>
                 <Semantic>
                        <Label>
                               <Name>Relative location of the object</Name>
                        </Label>
```

```
                        <Relation              source="#Object302"              target="#Room"
type="urn:mpeg:mpeg7:cs:SpatialRelationCS:2001:Inside"/>
                        <SemanticBase id="Object302" xsi:type="ObjectType">
                                <Label>
                                        <Name>Object302</Name>
                                </Label>
                        </SemanticBase>
                </Semantic>
                <SpatialLocator>
                        <Box mpeg7:dim="2 2" xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001">
                                187 75 20 76
                        </Box>
                </SpatialLocator>
        </StillRegion>
   </SpatialDecomposition>
</StillRegion>
```

Table 1: MPEG-7 fragment including an instance of the output of AIT's tracker (notably labels and bounding box information for a frame with three targets)

This is an example of the body tracker output in a frame with no targets.

```
<StillRegion id="SRimg_1225116702.025676.jpg">
   <StructuralUnit href="D:\Frames\img_1225116702.025676.jpg">
          <Name>Frame</Name>
   </StructuralUnit>
</StillRegion>
```

Table 2: MPEG-7 fragment including an instance of the output of AIT's tracker (corresponding to an empty frame)

Notes on the (MPEG-7) Examples

With respect to the above examples, note that the information about each frame is bundled together under a "StillRegion" tag, with an ID corresponding to the filename of that frame on disk. Each frame is divided into a list of sub-segments under the "SpatialDecomposition" tag, each sub-segment pertaining to a single target and containing the following information:

- A link to an image filename on disk: It is assumed that these are cropped versions of the original frame containing just the target in question, which could be useful when trying to create a collage of all detections of a given target.
- A "Semantic" tag with some simple textual information about the target, namely the target ID (which is generated by the Body Tracker to enforce consistency between successive detections of the same target across a number of frames); again, this could be used to answer a retrieval query of the sort "Give me all frames where 'Person X' appears on the video".
- A "SpatialLocator" tag, which indicates the location of the target bounding box with respect to the whole frame size. Each "SpatialLocator" contains a "Box" element, enumerating the (x,y)-coordinates [presumably of the upper left corner of the bounding box] and the (width,height) pair; all quantities are measured in pixels.

It also seems possible to integrate "quality" metrics about the tracked targets (e.g. face recognition confidence) by using the "PointOfView"/"Importance" tags as follows:

```
<PointOfView viewpoint="Importance">
    <Importance>
        <Value>.4</Value>
    </Importance>
</PointOfView>
```

There is also an element called "DominantColour", which contains information about the most prominent colour in a sub-region of the frame (or even the whole frame). This element could be filled by a perceptual component dedicated to this task, in order for example to support PRISA use case (which foresees colour trends).

8.3 SMART metadata in RDF triples

The subjects of crowd, any gas from a chemical sensor, environment (from sensor networks) and activity (from social networks) are shown, together with their associated predicates. This list is discussed in Deliverable 4.1, "SMART Distributed Knowledge Base and Open Linked Data" and will be changing as new components are added to the system.

Subject	Predicate	Object	Comment
Crowd	Time	Date-Time	Measurement time
	Density	Numeric	Continuous metadata
	Colour	Colour spec (integer part) + frequency (decimal part(	
	Primary horizontal motion	Numeric	
	Primary vertical motion	Numeric	
	Motion spread	Numeric	
	Status	Logical	Low-level event
	Uniform colour	Colour specification	
Audio	Level	Numeric	Continuous metadata
	Applause	Numeric	Probability of the event
	Crowd	Numeric	
	Music	Numeric	
	Traffic	Numeric	
Gas	Concentration	Numeric	Continuous metadata
	Limit	Exceeding/Normal	Low-level event
Environment	Humidity	Numeric	Continuous metadata
	Temperature	Numeric	
	Dew point	Numeric	
	Wind chill	Numeric	
	Pressure	Numeric	
	Wind average	Numeric	
	Wind gust	Numeric	
	Wind direction	Direction string	
	Rain since last reading	Numeric	
	Rain last hour	Numeric	
	Rain last 24 hours	Numeric	
	Rain last 7 days	Numeric	
	Rain last 30 days	Numeric	
	Is raining	Logical	Low-level event
	Is hot	Logical	
	Is cold	Logical	
Activity	Name	String	Filtered low-level event from social networks
	isActive	Logical	
	Date	Date this event refers to	
	temporalHint	Time this event refers to	