Tailored IoT & BigData Sandboxes and Testbeds for Smart, Autonomous and Personalized Services in the European Finance and Insurance Services Ecosystem

D4.5 – Semantics Streams Analytics Engine II

Lead Beneficiary	NUIG
Task Reference	T4.2
Revision Number	2.0
Deliverable Type	Report (R)
Dissemination Level	Public (PU)
Due Date	2021-05-31
Delivered Date	2021-08-31
Internal Reviewers	LINX, NUIG-Insight
Quality Assurance	INNOV
Review Status	Internally Reviewed and Quality Assurance Reviewed
Acceptance	WP Leader Accepted and/or Coordinator Accepted
EC Project Officer	Pierre-Paul Sondag

HORIZON 2020 - ICT-11-2018

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no 856632

Contributing Partners

Partner Acronym	Role	Author(s)
NUIG	Lead Beneficiary	Yasar Khan Martin Serrano Alex Acquier

Revision History

Version II	Date	Partner(s)	Description
1.1	2021-03-10	NUIG	ToC Version reviewed
1.2	2021-03-15	NUIG, N OVA	Updated ToC with Improved APIs and first Implementation
1.3	2021-04-10	NUIG	Updated TOC contributions
1.4	2021-04-10	NUIG	Integration of CI/CD Process and implementation
1.5	2021-05-15	NUIG	Additional contributions on Semantic Engine Documentation
1.6	2021-06-20	NUIG	Updates in Executive Abstract, Section 4, Section 5 and Section 6
1.7	2021-07-15	NUIG	Contributions in Section 5 and Section 6
1.8	2021-07-20	NUIG	Contributions in Section 7
1.9	2021-08-21	NUIG, INNOV	Pre-Final Version for Internal Review
1.95	2021-08-30	NUIG, LINX	Version with Quality Assurance
2.0	2021-08-30	NUIG	Version for Submission

Version I	Date	Partner(s)	Description
0.1	2020-08-08	NUIG	ToC Version
0.2	2020-08-08	NUIG, N OVA	Updated ToC with requested contributions Standard Vocabularies from Stakeholders on Hold until new online events are organised
0.3	2020-09-09	NOVA	Updated contributions
0.4	2020-10-10	NUIG	Integration of contributions Vocabularies form D4.1 integrated
0.5	2020-11-11	NUIG	Additional contributions on FIBO, FIGI and LKIF vocabularies and taxonomies
0.6	2020-11-11	NUIG	Updates in Section 1, Section 3 and Section 4
0.7	2020-12-20	NUIG	Contributions in Section 4 and Section 5
0.8	2020-12-23	NUIG	Final Contributions in Section 5
0.9	2020-12-23	NUIG, INNOV	First Version for Internal Review
0.95	2020-12-23	NUIG, LINX	Version for Quality Assurance
1.0	2020-12-23	NUIG	Version for Submission

Executive Summary

Semantics Stream Analytics Engine - Version II

This deliverable is the second version of the second version of the INFINITECH Semantics Streams Analytics Engine (SeSA-ME) and the related tools for enabling semantic data exchange. In this version of this deliverable the Prototype implementation of the engine has been produced and improved after initial feedback after producing the specification as part of ask 4.2 activity in WP4.

A section named New APIs has been included in this version to clearly identify the extended APIs that complement the SeSA-Me engine Prototype implementation.

This document is the second version of three where the basic services and tools for data interoperability and their use in particular use cases or pilots are described. In this document and in relation with the previous version the implementation of the SeSA-ME component is provided as an open source implementation that can be used when data sharing and data exchange is required. The Prototype provided and documented is a reference implementation that was improved following general requirements coming from the study and purposes at INFINITECH pilot level following stakeholder's requirements and also from particular domains where the use of a semantic engine for mashup building with semantic interoperability capabilities.

As a result of the REVIEW Period I, this deliverable suffered a delay in submission, mainly because the prototype implementation that was taking into consideration the progress of the project with the stakeholders from the financial sector and their data models (i.e. mainly banking data models) that were initially planned to be used suffer a change because some of the pilot re-focus their objectives and thus new vocabularies were updated. This delay did not affect the overall progression of the project because the involvement of re-focused pilots and their stakeholders will come into few months later and thus pilot's stakeholders requirements are yet in time to be integrated. This delay in the submission of this second version of the deliverable was agreed and informed to the coordination and consortium and informed in the WP4 online meetings of the INFINITECH project.

Semantics Stream Analytics Engine - Version I

In the first version of this deliverable the INFINITECH Semantics Streams Analytics Engine (SeSA-ME) and the related tools for enabling semantic data exchange were introduced, The SeSA-ME and its tool are based on the development of an interoperability (ontology-based) database/registry supporting linking of diverse systems and datasets based on shared semantics, as well as semantically interoperable analytics.

The Semantic Engine is an extension of the Super Stream Collider (SSC) tool, which provides a set of web-based interfaces and tools for building data mashups combining semantically annotated Linked Stream and Linked Data sources into easy-to-use data mashups for applications. The SeSA-ME system includes tools along with a visual SPARQL query editor using Swagger APIs and visualization tools for novice users while supporting full access and control over the data mashups for expert users. Tied with the development of the SeSA-ME platform is the development and deployment of the INFNITECH Graph Data Model which enables the support for both the design and deployment of stream-based web applications in a very simple and intuitive way and the analytics services using stream-based applications and services.

In the first version of the deliverable we also introduce the INFINITECH Graph Data Model as an Ontology or set of Standards Ontologies:

(a) to model and represent Finance and Insurance concepts with additional concepts in related relevant areas – e.g., from the Security Transactions domain, Security and Privacy domain – within the INFINITECH project stakeholders,

(b) to enable the semantic interoperability between Internet-connected objects for Finance and Insurance applications in diversity of applications and services settings, and

(c) to enable the application of analytics services and reasoning algorithms for seamless automated information exchange for more complex services and combined applications.

INFINITECH Graph Data Model is composed by following Core ontology standards, such as FIBO, FIGI and LKIF standards and we bootstrap the implementation and deployment of the Semantic Analytics Engine from those existing efforts towards the ontological descriptions of concepts, applications and online services, etc. relevant for the INFINITECH project pilots.

Table of Contents

List of Figures

List of Tables

Abbreviations/Acronyms

AI	Artificial Intelligence
CQELS	Continuous Query Evaluation over Linked Streams
DnS	Descriptions and Solutions
DOI	Digital Object Identifier
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering
DUL	DOLCE+DnS Ultralite
FIBO	Financial Industry Busines Ontology
FIGI	Financial Instrument Global Identifier
FOAF	Friend of a Friend
GIS	Geographic information system
GSN	Global Sensor Networks
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
ICT	Information and Communications Technology
LD	Linked Data
LKIF	Legal Knowledge Interchange Format
LOD	Linking Open Data
MIME	Multipurpose Internet Mail Extensions
NOAA	National Oceanic and Atmospheric Administration
OGC	Open Geospatial Consortium
OMG	Object Management Group
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema
RFS	Request for Service
SaS	Sensing-as-a-Service
SIOC	Semantically Interlinked Online Communities
SLA	Service Level Agreement
SOAP	Simple Object Access Protocol
SPARQL	SPARQL Protocol and RDF Query Language
TaS	Traceability-as-a-Service
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifiers
URN	Uniform Resource Name
USB	Universal Serial Bus
W3C	World Wide Web Consortium
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

1 Introduction

This deliverable is the second version of the second version of the INFINITECH Semantics Streams Analytics Engine (SeSA-ME) and the related tools for enabling semantic data exchange. In this version of this deliverable the Prototype implementation of the engine has been produced and improved after initial feedback after producing the specification as part of ask 4.2 activity in WP4.

The Semantic Stream Analytics Middleware-Engine (SeSA-ME) is an extension of the Super Stream Collider (SSC) platform and tools, The SeSA-ME system includes tools along with a visual SPARQL query editor using Swagger APIs and visualization tools for novice users while supporting full access and control over the data mashups for expert users. The development of the SeSA-ME platform requires the use of an associated data model, thus it is also part of this development the development and deployment of the INFINITECH Graph Data Model which follows the specifications from D4.1. The INFINITEC Graph Data Model enables supporting both the design and deployment of stream-based web applications in a very simple and intuitive way and the extension to analytics services using stream-based applications and services.

A section named New APIs has been included in this version to clearly identify the extended APIs that complement the SeSA-Me engine Prototype implementation.

1.1 Objective of the Deliverable

This deliverable reports the implementation and improvement of the Semantic Stream Analytics Engine (SeSA-ME) as a framework and tool for interoperability and data exchange. This document is the second version of three where the basic services and tools for data interoperability and their use in particular use cases or pilots are described. In this document and in relation with the previous version the implementation of the SeSA-ME component is provided as an open source implementation that can be used when data sharing and data exchange is required. The Prototype provided and documented is a reference implementation that was improved following general requirements coming from the study and purposes at INFINITECH pilot level following stakeholder's requirements and also from particular domains where the use of a semantic engine for mashup building with semantic interoperability capabilities seems to be an alternative to resolve some of the issues identified and described in this document.

This deliverable also provide a revision of the INFINITECH Core Ontology, it also refers to the online tool for providing Financial Industry Busines Ontology (FIBO) from the EDM council, the Financial Instrument Global Identifier (FIGI), an established global standard issued under the guidelines of the Object Management Group (OMG) and the Legal Knowledge Interchange Format (LKIF) Ontologies.

The reference to build an online tool comes from the need to have not only machine readable ways to these ontologies but also from the need that derivates from (a) the requirements from the use case descriptions, i.e. the involved concepts and relationships between them identified in Task 4.1, (b) the set of related ontologies relevant to INFINITECH identified in task 4.2 as reported in their deliverables and (c) the relevance of some terms used in different domains that can be used for exchange data, "relevance" refers to the overlap between the concepts and relationships of the INFINITECH use cases in the different domains (i.e. finance and insurance for example) and the ones described by the existing ontologies.

1.2 Insights from other Tasks and Deliverables

This deliverable is a updated report that describes the implementation of the semantic engine and its components that make it functional and easy to use, this deliverable also includes references to an online semantic framework that makes use of the data model principles described in the Deliverable D4.1 already as initial design and implementation.

This deliverable extends the functionalities and capabilities about the INFINITECH Semantic Interoperability Framework, introduced and explained in the Deliverable D4.1, where basic data models are described.

This deliverable refers to section 4 in the Deliverable D4.2 Semantic Models and Ontologies II, where INFINITECH Core Data Model & Semantic online tools are included, that D4.2 deliverable's section provides an overview of the online FIBO, FIGI and LKIF standard ontologies for the INFINITECH application domains while also highlighting online the core concepts, terms and vocabularies that will be part of the INFINITECH core semantic models.

1.3 Structure

The overall content of this document focuses on the New APIs design, implementation and deployment of the Semantic Stream Analytics Engine (SeSA-ME),

The previous sections providing the comprehensive analysis and overall information around related areas to graph data modelling, stream processing and data mashups building is provided from version I of this deliverable and left in the document to make it a live document that is self-comprehensive and self-contain.

Section 2 outlines the background knowledge needed to understand the major topics addressed in this deliverable. The first part covers the notion of Linked Data unifying principle for sharing and linking data from different sources; the second part introduces ontologies and the state-of-the-art concept to add semantics to data for enabling data discovery and reasoning over the annotated data and outlines the ontological requirements derived from the INFINITECH use cases and relates them with the INFINITECH ontology/vocabulary.

Section 3 reviews related work. This mainly includes existing works towards adding semantics to the finance sector addressing the overview of the most used standards, as well as an overview to existing stream and mashup builders' platforms together with a brief outline how interoperability and heterogeneity is addressed in these platforms.

Section 4 represents the core part of the deliverable, introducing the SeSA-ME Specification. The rationale is that SeSA-ME follows the recommended best-effort practice to reuse existing, popular ontologies/vocabularies as much as possible. For each included vocabulary, the corresponding subsection highlights the basic defined concepts and relationships between concepts and argues the potential relevance for INFINITECH project. Section 4.2.3 is now the new APIs

Section 5 is the documentation for the CI/CD process documenting the deployment and procedure to allow developers and people interested in the use of the SeSA-ME component.

Section 6 present the conclusions and present some pointers in how the INFINTECH SeSA-ME component will follow the sandboxes design in the INFINITECH project and outlines the support of pilots following the proposed methodology. Section 6 includes a list of relevant references alike the ones used across this document.

2 Background Knowledge and Pre-requisites

Semantic Streams and Mashups Processing requires a basic know-how on data modelling and processing. This section describes the basic concepts that are used in this deliverable and also in the design and implementation of the Semantics Streams Analytics Engine (SeSA-ME) architecture, it also follow the design principles for high level architectures [Boots 2017]. It is highly recommended to follow these sections carefully not only to understand the different terms and concepts introduced but also to understand the use of the semantics in the context of the INFINITECH Graph Data model construction.

2.1 Linked Data

Linked Data is the basic mechanism recognised in the semantic web that is used for Data Sharing and Data Exchange when implementing semantic web applications [Heitman 2009], it is not until recently that in the landscape of data on the Web, Linked Data was comprised by the existence of self-contained data repositories (Data Silos). Basically, each Web application or platform used to maintain its own repository, even if there was a significant overlap between these datasets and data that was publicly accessible. From a knowledge and information retrieval perspective. The integration of different kinds of data sources yields significant added value. However, the use of different formats and different technologies has made such integration challenging and even today it remains as one of the principal challenges for data sharing.

These challenges spurred the development and success of the concept of a process that allows to logically define and practically establish connections between parts of the information. For instance this is the case of a location of a person with a nationality according to that geographically location. This process that may sound trivial but that without the proper context, this association cannot be done. Let us think in the way a person learns for the first time that someone that was born in a country has a nationality associated to it since the day he or she was born there. What if this person decided to migrate to another country where he spent a defined period of time and his nationality is granted by the time of residence in that new location, a new relationship will be established.

Linked Data [LOD-Project] is the mechanism that helps to define and establish those relationships. This concept describes a method of publishing all kinds of structured data so that it can be interlinked and become more useful in the form of accessible online documents that contains those relationships and definitions that are helpful for machine-readable solutions and humans, giving the priority to humans-centric because yet it will be a support to understand multiplicity of the use with the purpose of data interoperability functions (i.e. sharing, exchange, access, etc.) and also supporting machine-driven solutions.

Linked Data is built upon standard Web technologies such as HTTP and URIs, but rather than using them to serve web pages for human readers, it extends them to share information in a way suitable for reading them automatically by computers. This enables data from different sources to be connected and queried.

2.2 Resource Description Framework & Serialisation Formats

Linked Data is based on the notion of describing real world things using the Resource Description Framework (RDF)[W3C-RDF]. The following paragraphs introduce the basics about RDF model, and then outline existing formats to serialise data modelled in RDF.

2.2.1 The RDF Data Model

RDF is considered a simple, flexible, and schema-less model suitable to express and process a series of simple assertions. Consider the following finance data related example: "FinanceOperartion measures 100.00 Euro." Each statement, i.e., piece of information, is represented in the form of *triples* (RDF triples) that link a *subject* ("FinaceOperation"), a *predicate* ("measures"), and an *object* ("100.00Euro"). The subject is the thing that is described, i.e., the resource in question. The predicate is a term used to describe or modify some aspect of the subject. It is used to denote relationships between the subject and the object. The object is, in RDF, it's the "target" or "value" of the triple. It can be another resource, or just a literal value such as a number or word.
In RDF, resources are represented by Uniform Resource Identifiers (URIs). The subject of RDF triples must always be a resource. The typical way to represent an RDF triple is a graph, with the subject and object being nodes and the predicate a directed edge from the subject to the object. So, the above example statement could be turned into an RDF triple illustrated in Figure 1.

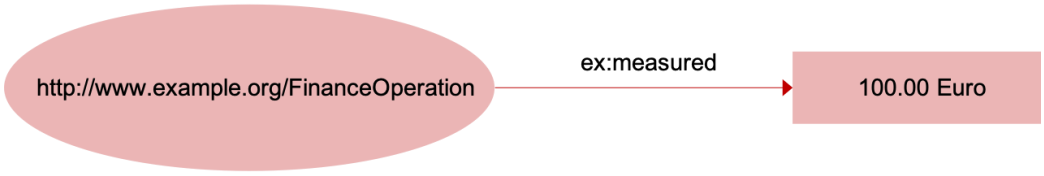


Figure 1. RDF triple in graph representation describing "FinanceOperation measures 100 Euro."

Since objects can also be resources with predicates and objects on their own, single triples are connected to a so-called RDF graph. In terms of graph theory, the RDF graph is a labelled and directed graph. As illustration we extend the previous example, replacing the literal "100Euro" by a resource "Measurement" for the object in the RDF triple in Figure 1. The resource itself has two predicates assigning a unit and the actual value to the measurement. The unit is again represented by a resource and the value is a numerical literal. The resulting RDF graph looks as follows:

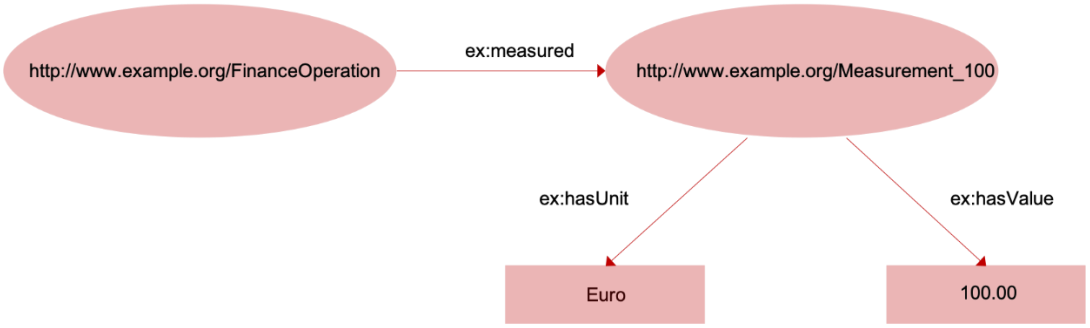


Figure 2. Simple RDF graph including the example RDF triple.

2.2.2 Serialisation Formats

The RDF data model itself does not describe the format in which the data, i.e., the RDF graph structure, is stored, processed, or transferred. Several formats exist, whose purpose is to serialise RDF data; the following overview list the most popular formats, including a short description about their main characteristics and examples.

Figure 2 shows a simple RDF graph to serve as a starting point for the explanations.

2.2.2.1 RDF/XML

The RDF/XML syntax [W3C-RDF] is standardised by the W3C and is widely used to publish Linked Data on the Web. On the downside however, the XML syntax is also regarded as difficult for humans to read and write. This indicates a need for consideration of (a) other serialisation formats in data management and control workflows that involve human intervention and (b) the provision of alternative serialisations for consumers who may wish to examine the raw RDF data. The RDF/XML syntax is described in detail as part of the W3C RDF Primer. The MIME type that should be used for RDF/XML within HTTP content negotiation is application/rdf+xml. The listing shown in Figure 3 below shows the RDF/XML serialisation for the RDF graph.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:ex="http://www.example.org/"
<rdf:Description rdf:about=" http://www.example.org/FinaceOperation">
  <ex:title>100.00Euro</ex:title>
</rdf:Description>
</rdf:RDF>
```

Figure 3. RDF/XML Serialisation Example

2.2.2.2 Turtle

Turtle (Terse RDF Triple Language) [W3C-Turtle] is a plain text format for serialising RDF data. It provides support for namespace prefixes and other shorthands, making Turtle typically the serialisation format of choice for reading RDF triples or writing them by hand. A detailed introduction to Turtle is given in the W3C Team Submission document Turtle. It was accepted as a first working draft by the World Wide Web Consortium (W3C) RDF Working Group in August 2011, and parsing and serialising RDF data is supported by many RDF toolkits. The MIME type for Turtle is text/turtle;charset=utf-8. The Figure 4 shows the serialisation listing for the example RDF graph in Turtle syntax.

```
@prefix : <http://www.example.org/> .
:FinanceOperation :measures “100Euro”
```

Figure 4. Turtle Serialisation Example

2.2.2.3 N-Triples

The N-Triples syntax [W3C-N-Triples] is a subset of Turtle, excluding features such as namespace prefixes and shorthands. Since all URIs must be specified in full in each triple, this serialisation format involves a lot of redundancy, typically resulting in large N-Triples particularly compared to Turtle but also to RDF/XML. This redundancy, however, enables N-Triples files to be parsed one line at a time, benefitting the loading and processing of large data files that will not fit into main memory. The redundancy also allows compressing N-Triples files with a high compression ratio, thus reducing network traffic when exchanging files. These two factors make N-Triples the de facto standard for exchanging large dumps of Linked Data. The complete definition of the N-Triples syntax is given as part of the W3C RDF Test Cases recommendation. The following listening in Figure 5 represents the N-Triples serialisation of the example RDF graph.

```
<http://www.example.org/FinancerOperation>
   <http://www.example.org/measures>
      “100Euor”@en-IE .
```

Figure 5. N-Triples Serialisation Example

2.2.2.4 RDFa

RDFa [W3C-RDFa] allows embedding RDF triples directly in (X)HTML documents using a set of attributes of the (X)HTML elements. The RDF data is not embedded in comments within the HTML document but interwoven within the HTML Document Object Model (DOM). This means that existing content within the page can be marked up with RDFa by modifying HTML code, thereby exposing structured data to the Web. It doesn’t require separate documents, but instead allows people to add structure to an existing content. A detailed introduction into RDFa is given in the W3C RDFa Primer. RDFa is popular in contexts where data publishers can modify HTML templates but have relatively little additional control over the publishing infrastructure. The RDFa serialisation shown in the example RDF graph is shown in the listening below as Figure 6. To ease presentation, all example used throughout this document are written in the Turtle syntax. This includes the usage of the following namespaces.

```
@prefix ssn:<http://www.w3.org/2005/Incubator/ssn/ssnx/ssn#> .
@prefix spitf:<http://spitfire−project.eu/ontology/ns#> .
@prefix dul :<http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#> .
@prefix f:<http://events.semantic-multimedia.org/ontology/2008/12/15/model.owl#> .
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:<http://www.w3.org/2002/07/owl#> .
@prefix xsd:<http://www.w3.org/2001/XMLSchema#> .
@prefix:<http://www.example.org/ns#> .
@prefix muo:<http://purl.oclc.org/NET/muo/muo#> .
@prefix ucum-unit:<http://purl.oclc.org/NET/muo/ucum/unit/> .
@prefix unit:<http://www.w3.org/2007/ont/unit#> .
@prefix foaf:<http://xmlns.com/foaf/0.1/> .
@prefix dbpedia:<http://dbpedia.org/ontology> .
@prefix ao:<http://purl.org/ontology/ao/associationontology.html#> .
@prefix sweet:<http://sweet.jpl.nasa.gov/2.2/sweetAll.owl#> .
```

Figure 6. RDFa Serialisation Example

2.3 SPARQL: Querying Linked Data

SPARQL (SPARQL Protocol and RDF Query Language) [W3-SPARQL] is the most popular query language to retrieve and manipulate data stored in RDF, and it became an official W3C Recommendation in 2008. Depending on the purpose, SPARQL distinguishes the following for query variations:

- SELECT query: extraction of (raw) information from the data
- CONSTRUCT query: extraction of information and transformation into RDF
- ASK query: extraction of information resulting in a True/False answer

- DESCRIBE query: extraction of RDF graph that describes the resources found

Given that RDF forms a directed, labelled graph for representing information, the most basic construct of a SPARQL query is a so-called *basic graph pattern*. Such a pattern is very similar to an RDF triple with the exception that the subject, predicate or object may be a variable. A basic graph pattern matches a subgraph of the RDF data when RDF terms from that subgraph may be substituted for the variables and the result is RDF graph equivalent to the subgraph. Using the same identifier for variables also allow combining multiple graph patterns. To give an example, the SPARQL query returns the name of all pairs of people where ?person1 knows ?person2 (note that foaf:knows is not defined as symmetric relation) see this in Figure 7 as follow:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name1  ?name2
FROM <http://example.org/foaf>
WHERE {
    ?person1 foaf:knows ?person2 .
    ?person1 foaf:name  ?name1 .
    ?person2 foaf:name  ?name2 .
}
```

Figure 7. Simple SPARQL query utilising basic graph patterns

Besides the aforementioned graph patterns, the SPARQL 1.0 standard also supports the sorting (ORDER BY), and the limitation of result sets (LIMIT, OFFSET), the elimination of duplicates (DISTINCT), the formulation of conditions over the value of variables (FILTER), and the possibility to declare a constraint as OPTIONAL. As an illustration, we modify the example query in Figure 7. The query now depicted in Figure 8 retrieves all persons that Alice knows including, if available, the URL of their homepages. The results are sorted with respect to the name of know persons, and finally limited to the first 20 entries.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?hpage
FROM <http://example.org/foaf>
WHERE {
    ?person1 foaf:knows ?person2 .
    ?person1 foaf:name  “Alice” .
    ?person2 foaf:name  ?name .
    OPTIONAL { ?person2 foaf:hompage ?hpage }
}
ORDER BY ?name
LIMIT 20
```

Figure 8. Example SPARQL 1.0 query

The SPARQL 1.1 standard significantly extended the expressiveness of SPARQL. In more detail the new features include

- Grouping (GROUP BY), and conditions on groups (HAVING)
- Aggregates (CONT, SUM, MIN, MAX, AVG, etc.)
- Subqueries to embed SPARQL queries directly within other queries
- Negation to, e.g., check for the absence of data triples
- Project expression, e.g., to use numerical result values in the SELECT clause within a mathematical formulas and assign new variable names to the result
- Update statements to add, change, or delete statements

- Variable assignments to bind expressions to variables in a graph pattern
- New built-in functions and operators, including string functions (e.g., CONCAT, CONTAINS), string digest functions (e.g., MD5, SHA1), numeric functions (e.g., ABS, ROUND), or date/time functions (e.g., NOW, DAY, HOURS)

Again, to give a short example, the query in Figure 9 counts for each person the number of contacts, i.e., the number of others each person knows. Note that we can use a blank node (_:a), i.e., generic placeholder, since we were in this case not interested in any additional information about contacts. The results are sorted with respect to the number of contacts in a descending manner.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name COUNT(*) AS ?numberOfContacts
FROM <http://example.org/foaf>
WHERE {
    ?person foaf:knows _:a .
    ?person foaf:name  ?name .
}
GROUP BY ?name
ORDER BY DESC(COUNT(*))
```

Figure 9. Example SPARQL 1.1 query

2.4 Web of Data

The *Web of Data*, or the *Semantic Web*, is the continuously growing result of the Linked Data idea and goals. A large and increasing number of individuals, organisations, public bodies, etc. publish their data in line with the principles of Linked Data, instead of just putting them on the Web as content of traditional websites. Due to the links between resources of different data sources, the Web of Data can be seen as giant RDF graph forming a unified, global data space. At the time of writing, this RDF graph contains billions of triples spanning all kinds of knowledge domains.

The Web of Data can before being described by the following characteristics:

1) **Generic:** The simple data model of RDF can contain any type of data and enables the implementation of generic tools for data access and discovery as well as the implementation of generic optimisation techniques.

2) **Open:** There are no access restrictions to the Web of Data. Anyone can publish data as Linked Data, create links to other data sources, and thus contribute to the Web of Data RDF graph.

3) **Unconstrained:** The Web of Data can contain statements that represent a disagreement or a contradiction about described resources.

4) **Flexible:** The Web of Data does not constrain or limit data publishers to a specific set of vocabularies with which to represent their data. Publishers can choose their own vocabulary and can also combine or extend them.

5) **Self-describing:** If an application consuming Linked Data encounters data described with an unfamiliar vocabulary, the application can dereference the URIs that identify vocabulary terms to find their definition.

6) **Standardised:** All underlying technologies of the Web of Data, including RDF for modelling the data and HTTP for accessing the data, are standardised. This simplifies the data access and processing compared to Web APIs that rely on heterogeneous data models and access interfaces.

The origins of this Web of Data lie in the efforts of the Semantic Web research community and particularly in the activities of the W3C Linking Open Data (LOD) project [LOD-Project], a grassroots community effort founded in January 2007. The founding aims of the project, which has spawned a vibrant and growing Linked Data community, was to bootstrap the Web of Data by identifying existing data sets available under open licenses, convert them to RDF according to the Linked Data principles, and to publish them on the Web. As a key principle, the project has always been open to anyone who publishes data according to the Linked Data principles. This openness is a likely factor in the success of the project in bootstrapping the Web of Data.

2.5 Data Integration

The main aspired benefit of the Linked Data idea lies in the interlinking of data between different sources, eventually resulting in the Web of Data. On the "traditional" Web, the user can browse information without any knowledge of the underlying technical structure, and the browsing experience is seamless even when linking from one website to another. Similarly, with Linked Data, it should be possible to browse datasets, and link from one dataset to another, even if they are stored in different places and in different formats. The applied technologies and the resulting characteristics in terms of data, however, involve several challenges when it comes to integrating different data sources, some listed as follow:

1) The flexible modelling of information, in general, implies that the same kind of information can be modelled in more than one way. For example, the home of a person can be modelled by linking the resource describing the person to literal nodes containing the street name, house number, etc., or linking the resource to a dedicated address resource which itself then has links to the specific address information.

2) If two different data sources contain information resources referring to the same real-world concepts, these resources are typically identified via different URIs. Thus, in different data sources the same real-world concept is often represented differently. There is not inherent connection between the corresponding resources.

3) Publishers of Linked Data might user different vocabularies, i.e., speak a different language. For example, a contact relationship between persons can be names as "has-contact", "knows", "is-acquainted-with", or similar. Although the semantics between these notions is the same – and is understandable for humans – the different syntax makes the integration of this information difficult at the machine level.

Given these challenges, basic best-practice techniques have been formulated and are promoted. Firstly, while anyone is free to provide their own ontology, Linked Data publishers are encouraged to use existing ontologies as much as possible. In a nutshell, ontologies define the basic terms (i.e., the vocabulary) and relations of a domain of interest, as well as the rules for combining these terms and relations. Ontologies are used for communication between people and organisations by providing a common terminology over a domain. They provide the basis for interoperability

between systems. The idea of reusing existing ontologies resulted in the definition of ontologies typically addressing terms of a specific domain. Often their definition is done not by data publishers but by ontology maintainers as supporting third parties among the data publishers and data consumers. And secondly, even when data publishers use the same ontology, different data sources still might contain equivalent resources, i.e., referring to the same real-world concept, but featuring different URIs. This creates the need to explicitly interconnect these resources via relations that indicate that both resources represent the same real-world concept. We address the related important notion of ontologies in more detail in the following Section 2.6.

2.6 Ontologies

Ontologies aim to add and formalise semantics, i.e., meaning, to provide reviewed information to allow for analytics and reasoning services over the data of multiple datasets to derive further knowledge. In the deliverable D4.1. The process to design an Ontology called “Ontology Engineering Method” is explained and proposed for the INFINITECH project to one of the alternatives enable data interoperability. In this deliverable D4.2 and particularly in this section a review about the basic terminology in ontologies is presented, details on how ontologies are used in INIFINITECH is already explained and presented in the Deliverable D4.1, Therefore here we only focus on providing a short summary of the related concepts and their implication when ontologies are used for data sharing.

2.6.1 Basic Concepts

The notion of ontology is originally taken from philosophy and it is now a common concept in various fields including computer science. Nevertheless, the use of the ontologies is quite similar in different domains the meaning of the term varies among these fields. In this section we give a concise introduction to the notion of ontology and related concepts. For this, we first give a short overview of the most relevant concepts in the context of semantics.

2.6.1.1 Vocabulary

A vocabulary is a set of terms (controlled) with informal natural language definitions that specify meaning. A controlled vocabulary is typically more about the terms rather than the underlying concepts, i.e., the terms’ definitions do not include any specific structural order. The emphasis on “controlled” mainly refers to the requirement that there should be governance or agreed-upon procedures in case the vocabulary needs to be changed (either by adding or removing terms). Between the terms of a vocabulary there are no relationships defined. Thus, pure vocabularies, as simple list of words with no relationships, do not allow for reasoning.

2.6.1.2 Taxonomy

A taxonomy is a controlled vocabulary that is organised into a hierarchy. Each term names a category, kind or class. Compared to vocabularies, taxonomies introduce relationships between terms. However, taxonomies utilise only one type of relationship: it means “is-a” or “is-a-kind-of” and corresponds to a subclass relationship. According to its strict definition each term in a taxonomy has exactly one parent term. Thus, taxonomies typically have a tree-like structure. In general, however, the term “taxonomy” often refers to hierarchies with multiple parents. (It is also sometimes loosely used to refer to networks with more than one kind of relationship between terms). If the strict definition of taxonomy is used, subclass reasoning is supported. This mainly includes that all information associated to a parent class is also associated with all its ancestor classes.

2.6.1.3 Ontology

An ontology features terms that name classes. The set of classes is organised into a network with arbitrarily many kinds of relationships. The set of relationship types typically also includes a subclass relationship, which as outlined above, forms a taxonomy often representing the backbone of the ontology. The relationships themselves have properties that are used for inference. For example, a relationship "same-as" can be defined as symmetric, while relationship "contains" cannot. The meaning of a type of relationship between two classes in an ontology is always formal and well defined. That allows for automated reasoning beyond the limited subclass reasoning within taxonomies. For example, the information that "Finance contains Payments" and "Payments is the same as the Transaction" allows deriving the knowledge that "Finance contains Transaction".

In terms of expressiveness, ontologies are the most powerful for to define concepts. Particularly for adding semantics to the data, vocabularies or taxonomies by themselves are too limited in their expressiveness and the included support for reasoning. This makes ontologies the concept of choice for the Web of Data. An ontology is a description (like a formal specification of a program) of the concepts and relationships relevant in the abstract model of some domain-specific knowledge agreed by a group of stakeholders. This conceptualisation describes knowledge about the domain rather than states, thus the ontology changes very rarely. A conceptualisation can be defined as an intentional semantic structure encoding the implicit knowledge that constrains the structure of part of a domain.

An ontology is a partial specification of the whole intentional semantic structure of a domain, in which the possible use of constructs is restricted. It is usually a logical theory that expresses the conceptualisation explicitly. Ontologies are important tools for enabling knowledge sharing and reuse. Ontology represents ontological commitments, i.e., agreeing on the usage of a vocabulary in a way that is consistent (but not necessarily complete) with respect to the theory specified by an ontology. Every knowledge base (or corresponding agent) is committed to some conceptualisation. We can describe the ontology of a program by defining a set of representational terms. In such an ontology, the names of entities in the universe of discourse (i.e., set of objects that can be represented, e.g., classes, relations, functions) are associated with both descriptions of what the names mean and formal axioms. Such axioms constrain the interpretation and well-formed use of the corresponding terms. Digital agents can commit to ontologies and ontologies are designed so that the knowledge can be shared among agents.

Currently, ontologies are commonly used for data integration, i.e., using a conceptual representation consisting of ontological terms of data and of their relationships, to eliminate heterogeneities. So far, ontologies have been applied to several fields, for example, search engines (e.g., Yahoo! categories), on-line shopping (e.g., Amazon's product catalogue), life science (e.g., UMLS [UMLS-Ontology] and Gene Ontology [GENE-ontology]). Additionally, an online lexicon database, WordNet [WordNet-Lexicon] is widely used for discovery of semantic relationships between concepts (e.g., homonyms, synonyms, sub concepts, etc.)

Ontologies are expressed using a formal representation to be machine-processable. There exist several formal languages for this purpose, each characterised by different levels of expressivity. A specification is considered formal when at least one relation is defined between terms in a formal language, so that new conclusions can be inferred. As already mention, the "is-a" relation can be represented in a formal way to express a hierarchical classification, expressing subsumption. For instance, A subsumes B meaning that everything that is in A is also in B. More expressive formal languages are those providing a set of constructs to describe classes, instances, relations and constraints.

The most formal and expressive ones are those that use full logics. On the other hand, the expressiveness of a language directly affects the performance of subsequent reasoning over the data. Simply speaking, the more expressive a language is in describing classes, relations, etc., the higher the resulting reasoning complexity, and vice versa. During ontology development it is usually better to choose an expressive language. Afterwards, in case the performance is not acceptable, the ontology can be reduced to a subset for some levels of automatic processing.

In terms of data management, INFINITECH is about data processing, sharing and discovery, i.e., to deal with the heterogeneity of finance data from different sources and to support the interoperability between these sources. INFINTECH will leverage from existing efforts in finance and insurance areas. This includes, firstly, Linked Data principles and the involved technologies (RDF, SPARQL, etc.) to provide a common data model, and secondly, the notion of an ontology to add semantics / meaning to data, particularly for the support of data discovery and reasoning.

Ontologies in a general perspective are conceptual representations consisting of defined terminology about data and of their relationships. Ontologies are used when different source of data using different data representations needs to be mapped and/or aligned to eliminate heterogeneities. An ontology typically refers to (a) a controlled vocabulary, i.e., a set of terms with informal natural language definitions that specify meaning, (b) a taxonomy, i.e., a basic hierarchical organisation of the terms of the vocabulary, and (c) additional types of relationships between the terms to specify the meaning of these relationships.

Ontologies are created for a specific domain to ensure a manageable size of the vocabulary. When developing a new ontology, it is desirable to reuse existing ontologies as much as possible, this simplifies the development since one can focus on the domain or application-specific knowledge only. Future integration between applications is facilitated since common parts of ontologies will be shared. When multiples Ontologies are created, as result of diversity on application scenarios requiring various operations between the different ontologies, the alignment of ontologies is required. This deliverable, therefore, describes the basic mechanisms for modular reuse of multiple ontologies, and features a comprehensive list of exiting ontologies whose covered domains overlap with the application scenarios of INFINITECH pilots.

2.6.2 Semantics for the Web of Data

The Semantic Web is an effort supported and started by the World Wide Web Consortium (W3C) to make all information available on the Web, “understandable” and processable by machines. Thus, humans would be able to easily find required knowledge rather than just web documents in which the knowledge is hidden and sparse. Like the Web that is a distributed hypertext system, the Semantic Web is a distributed knowledge base system. Consequently, agreed concepts using common vocabularies named ontologies are needed to define meaning and relations of distributed heterogeneous data items to reduce ambiguity, and thus they can be used to represented information in a proper machine-understandable manner. The W3C recommends RDF, the Resource Description Framework, for this purpose. RDF has been extended by other formalisms, but it is still the core framework. INFINITECH will use the “Full Owl extension”.

2.6.2.1 RDFS.

RDF Schema (RDFS) [W3C-RDFSchema] is a set of primitives to describe lightweight ontologies by using the RDF model and syntax itself. The described ontologies are the used to type resources and relations in the target domain. RDFS adds classes, subclasses, and properties to resources, supporting the description of taxonomies of classes and properties. An RDFS vocabulary defines the

allowable properties that can be assigned to RDF resources within a given domain. RDFS also allows you to create classes of resources that share common properties. Using the same triples paradigm defined by RDF, RDFS triples consist of classes, class properties, and values that define the classes and relationships between the resources within a particular domain. More specifically – but not exhaustive – RDFS allows

- to name and declare a vocabulary, i.e., to name resource types and binary relation types called properties),
- to specify the signature of properties, including the type of the domain (rdfs:domain), i.e., type of the subject, and type of the range (rdfs:range), i.e., type of the object),
- to specify the (multiple)-inheritance links between types of classes (rdfs:subClassOf),
- to specify the (multiple)-inheritance links between types of properties (subPropertyOf), and
- to provide labels (rdfs:label) and comments (rdfs:comment) in natural language to document and display these primitives.

2.6.2.2 OWL.

The Web Ontology Language (OWL) [W3C-OWL] extends RDF and RDFS with the goal to enhance the expressiveness and reasoning power. Essentially, it defines more classes that let creators of ontologies define more of the meaning of their predicates. For example, it allows defining relations between classes (e.g., disjoints), cardinality (e.g., "exactly one"), equality, richer typing of properties, characteristics of properties (e.g., symmetry), and enumerated classes. Like RDFS, OWL utilises the RDF triple paradigm for the definition of the ontologies.

As mentioned above, an increase of expressiveness potentially leads to an increase of complexity when it comes to reasoning over the described data. Therefore, OWL comes in three different flavours – OWL Lite, OWL DL, and OWL Full – that entail clear boundaries with respect to their expressiveness:

1) **OWL Lite**
   OWL Lite supports those users primarily needing a classification hierarchy and simple constraints. For example, while it supports cardinality constraints, it only permits cardinality values of 0 or 1. It is simpler to provide tool support for OWL Lite than its more expressive relatives, and OWL Lite provides a quick migration path for thesauri and other taxonomies. OWL Lite also has a lower formal complexity than OWL DL.
2) **OWL DL**
   OWL DL supports those users who want the maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time). OWL DL includes all OWL language constructs, but they can be used only under certain restrictions. For example, while a class may be a subclass of many classes, a class cannot be an instance of another class. OWL DL is so named due to its correspondence with description logics, a field of research that has studied the logics that form the formal foundation of OWL.
3) **OWL Full**
   OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF. For example, in OWL Full a class can be treated simultaneously as a collection of individuals and as an individual. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. On the other hand, OWL Full is not

a description logic. It is, therefore, unlikely that any reasoning software will be able to support complete reasoning for every feature of OWL Full.

Each of these sublanguages is an extension of its simpler predecessor, both in what can be legally expressed and in what can be validly concluded. The choice of the language eventually depends on the needs of an ontology developer. While OWL Lite is typically not expressive enough, the choice between OWL DL and OWL Full mainly depends on the extent to which users require the meta-modelling facilities of RDF Schema, and on the extent to which users require fully predictable reasoning support.

2.6.3 Modularisation of Ontologies

The purpose of authoring ontologies is also reusing of knowledge. Once an ontology is created for a domain, it should be (at least to some degree) reusable for other applications in the same domain. To simplify both ontology development and reuse, a modular design is beneficial. The modular design uses inheritance of ontologies - upper ontologies describe general knowledge, and application ontologies describe knowledge for a particular application, as illustrated in Figure 10 depicting the modularisation of ontologies depending on the scope and partial ordering defined by inheritance.

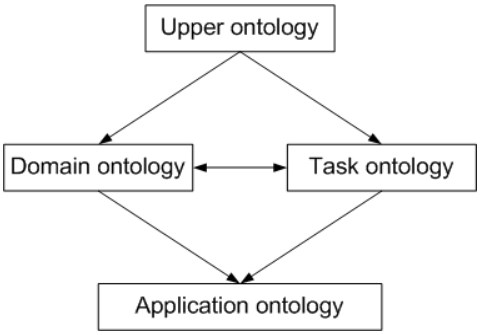


Figure 10. Example Modularisation of ontologies.

With respect to Figure 10, ontologies can be classified according to their scope. The resulting four classes of ontologies are defined as follows:

- *upper / generic / top-level ontology*
  Upper ontologies describe general knowledge that is independent from any specific domain or application. Typical examples are ontologies describing the concepts of space and time.
- *domain ontology*
  Domain ontologies cover concepts of broader areas of interest, e.g., the medical domain or electrical engineering domain, or narrower one, e.g., the financial sector domain.
- *task ontology*
  Task ontologies describe knowledge that refer to a general or more specific task or process. Such a task can be the assembling of individual parts or the observation of events.
- *application ontology*
  Application ontologies are the most specific ontologies describing the knowledge that is specific to a given application. To derive from a previous example, an application ontology can address the observation of measurements in a network of finance operations.

Figure 10 above represents the highest level of modularisation. However, modularisation can be used at each lower level as well. For example, an upper ontology may consist of modules for real

numbers, topology, time, and space (these parts of the upper ontology are usually called generic ontologies). Ontologies at lower levels import ontologies from upper levels and add additional specific knowledge. Task and domain ontologies may be independent and are merged for application ontology, or it is possible that for example task ontology imports domain ontology. The upper ontologies are the most reused ones while application ontologies may be suitable for one application only. When developing new ontology, it is desirable to reuse existing ontologies as much as possible. A new ontology should be started when another appropriate ontology does not exist, and by importing upper-level ontologies. This will simplify the development since one can focus on the domain or application specific knowledge only. It will also simplify integration between applications in the future since well-defined parts of ontologies will be shared.

2.6.4 Operations with Ontologies

It is possible that one application uses multiple ontologies, especially when using modular design of ontologies or when we need to integrate with systems that use other ontologies. In this case, some operations on ontologies may be needed to work with all of them. There are various operations on ontologies defined, such as merging, unification, and refinement of ontologies. In the context of this deliverable, however, we focus on the following two operations particularly relevant for the definition of the INFINITECH Core ontology.

2.6.4.1 Mapping of ontologies

The mapping from one ontology to another one is the expression of the way to translate statements from the first ontology to the other one. Often this means the translation between concepts and relations. In the simplest case it is a mapping from one concept of the first ontology to one concept of the second ontology. Such a straightforward mapping is not always possible, and some information can be lost in the mapping. While this is typically acceptable, a mapping may not introduce any direct inconsistencies.

2.6.4.2 Alignment of ontologies

Alignment is a process of mapping between ontologies in both directions. If such mappings are not directly possible, an alignment requires the modification of one or both original ontologies to enable such bidirectional mapping, without losing any information during the mapping. Thus, it is possible to add new concepts and relations to ontologies that would form suitable equivalents for the mappings. The specification of alignment is called articulation. Alignment, as well as mapping, may sometimes be only partial.

2.6.4.3 Ontology inheritance

When an Ontology A inherits from Ontology B, Ontology A inherits all concepts, relations and restrictions or axioms and there is no inconsistency introduced by additional knowledge contained in ontology A. This term is important for modular design of ontologies where an upper ontology describes general knowledge and lower application ontologies add knowledge needed only for each particular application. Inheritance defines a partial ordering between ontologies.

3 Related Work

3.1 Super Stream Collider (SSC)

The SSC enables the distributed cloud-based high-performance processing of semantically linked streams and it can be considered as an enabler for semantic analytics. SSC is used as a reference implementation for analytics over semantically unified/interoperable streams, as well as in the KYC and customer-centric services pilots. In INFINITECH we extended SSC to create SeSA-ME and evolve the utility and provisioning of services not only to data streams but also multi-domain data streams by using semantic application services.

The Super Stream Collider (SSC) platform and tools, provides a web-based interface and tools for building sophisticated mashups combining semantically annotated Linked Stream and Linked Data sources into easy-to-use resources for applications. The system includes the construction tools for continuous query processing using a CQELS editor and provides a visualization tool for novice users while supporting full access and control for expert users at the same time. Tied in with SSC development platform is a cloud deployment architecture which enables the user to deploy the generated mashups into a cloud, thus supporting both the design and deployment of stream-based web applications in a very simple and intuitive way.

The SSC platform is designed as a classical dataflow/workflow execution environment connecting processing input/outputs through pipelines for creating data mashups. Conceptually, each operator has multiple input streams and one output stream as illustrated in Figure 11. The inputs can be in any format while the output is RDF. Only the final operator of a workflow can return a format other than RDF, if necessary. Operators can be of three classes: A data acquisition operator is used to collect or receive data from data sources or gateways and can be pull-based or push-based. In these operators the data transformation and alignment can be done to produce a normalized RDF output format. A stream processing operator defines stream processing functionalities in a declarative language, e.g., CQELS. A streaming operator streams the outputs of the final operator of a workflow to the consuming applications.

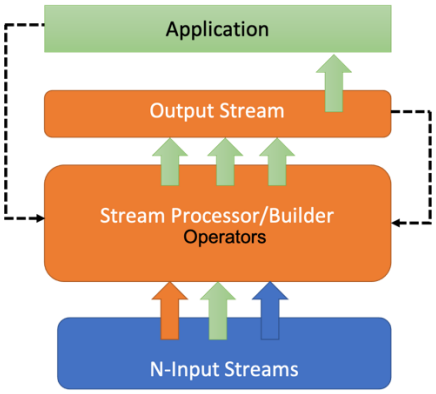


Figure 11. Super Stream Collider Platform Architecture

SSC can flexibly answer to dynamic load-profiles which are common in stream-based applications. In a concrete workflow, two connected operators can be executed in different execution containers. For instance, the data acquisition operator for collecting Tweets can stream data via the network to the stream processing engine. The external computing services such as SPARQL endpoints or web services are called external execution containers. To support the easy and intuitive definition of data processing workflows in a “box-and-arrows” fashion, the SSC platform offers a visual programming environment.

The interactive process of creating a mashup with SSC features context-aware discovery services for data sources. This process enables the user to incrementally build a workflow in a step-by-step fashion by dragging & dropping the required building blocks and connecting and parametrizing them. Also, this supports visually debugging the workflow of the mashup. When the user finishes a mashup, it can be deployed to the SSC cloud to be re-used as a data source or an operator.

The flows of data from the sources to the final output are defined by wiring the blocks with configured parameters. As reference example, the live visualisations of operator outputs are shown in Figure 12. The output of the workflow is a live mashup data stream which can be published, visualized, and queried. Currently SSC supports several types of live data sources, live streams like twitter streams in this example and DBPedia data sources, among others, which can be discovered by the SSC discovery component. This context-aware discovery service uses relevant text, location, sensor data sources that the user has typed and chosen as inputs to form the queries to such systems to find useful data items to recommend to the user.

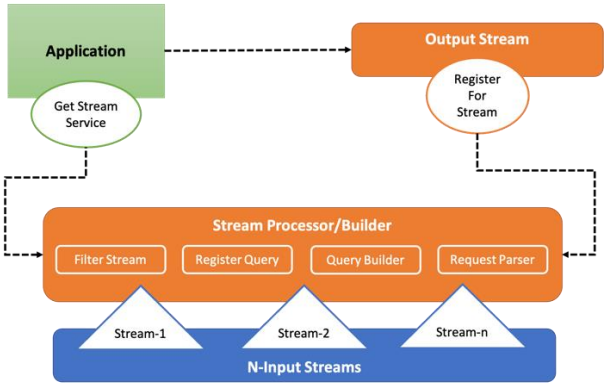


Figure 12. Super Stream Collider Functional Blocks.

Figure 13 shows an example where the result of the mashup data can be shown as raw data, RDF data or can be visualized in different types of charts, so that users can easily monitor their data processing workflows. In Figure 13, the output is a merge of multiple input streams. Another typical example of stream data is Twitter data as shown in Figure 5. In this example, the SSC collects all tweets mention about the user-specified topic and provides them as an RDF stream. For this the user only needs to drag an operator into the editor and enter the topic of interest.

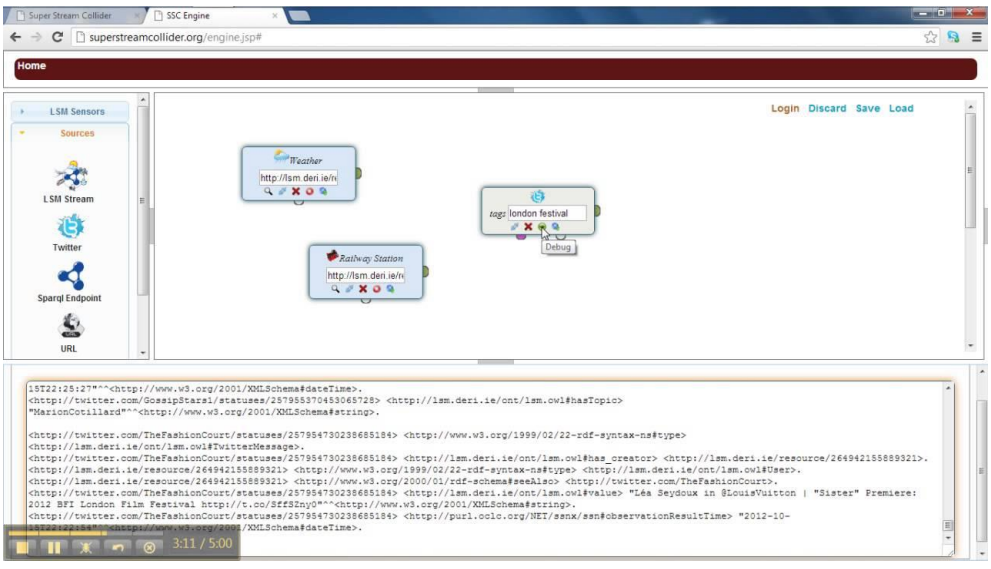


Figure 13. Super Stream Collider Mashups Builder

This section overviews interesting SSC functionalities. Due to space constraints we cannot go into great detail, but extensive documentation is available at https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.248.5925&rep=rep1&type=pdf. SSC provides a wide range of data acquisition operators which enable access to huge amount of data sources. The data wrappers allow SSC users to collect data directly from data sources. The RDF-izing operators extended from Any231 help to convert dynamic web data sources to RDF-based streams. We also implemented wrappers for transforming social stream data to RDF streams, as already mentioned for Twitter. For output streams of SSC mashups we support streaming protocols such as PubSubHubbub2 , XMPP3 and WebSockets.4 SSC also provides developers with various data manipulation operators. For RDFbased data mashups and data consolidation, we extended and support the operators of DERI Pipes [Le-Phuoc 2011] . To filter data streams, we use CQELS engine for constructing window-based filters with the full expressive power of SPARQL 1.1 (CQELS is an extension of SPARQL 1.1). To reduce the effort of learning SPARQL and CQELS, SSC also offers visual SPARQL and CQELS editor which enable the user to build SPARQL/CQELS queries interactively and a step-by-step way. Furthermore, this interactive workflow editing process is leveraged by the context-based discovery services which recommend potentially useful data sources and data items in every step of building a mashup in SSC. These services are powered by LSM's sensor database, and other online SPARQL endpoints such as Dbpedia, LinkedGeoData, etc. The user can add more knowledge by pointing SSC to further SPARQL endpoints.

3.2 Basic Design Principles for Building Mashups

The term Linked Data refers to a set of best practices for publishing and interlinking structured data on the Web. These best practices were introduced by Tim Berners-Lee in his Web architecture note Linked Data [Heitmann et al, 2009] and have become known as the Linked Data principles.
These principles are the following:

3.2.1 Use of Uniform Resource Identifiers (URIs) as names.

This principle advocates using URIs references to anything, i.e. extending the scope of the Web from online resources to encompass any object or concept in the world. Thus, things are not just Web documents and digital content, but also real-world objects and abstract concepts. These may include tangible things such as people, places and cars, or those that are more abstract, such as the relationship type of knowing somebody, the set of all green cars in the world, or the colour green itself.
To publish data on the Web, the things need to be uniquely identified. As Linked Data builds directly on the Web architecture [Jacobs 2004], the Web architecture term resource is used to refer to these things of interest, which are, in turn, identified by HTTP URIs. Linked Data uses only HTTP URIs, avoiding other URI schemes such as Uniform Resource Names (URN) [IEFT-URN] and Digital Object Identifier (DOI) [DOI]. The benefits of HTTP URIs are: (a) they provide a simple way to create globally unique names in a decentralised fashion, and (b) they serve not just as a name but also as a means of accessing information describing the identified entity.

3.2.2 Use HTTP URIs, so that names can be looked up by using those URIs.

The HTTP protocol is the Web's universal access mechanism. In the classic Web, HTTP URIs are used to combine globally unique identification with a simple, well-understood retrieval mechanism. Thus, this Linked Data principle advocates the use of HTTP URIs to identify objects and abstract concepts, enabling these URIs to be dereferenced (i.e., looked up) over the HTTP protocol to obtain a description of the identified object or concept. As a result, any HTTP client can look up the URI using the HTTP protocol and retrieve a description of the resource that is identified by the URI. This applies to URIs that are used to identify classic HTML documents, as well as URIs that are used in the Linked Data context to identify real-world objects and abstract concepts.

In case of URIs identifying real-world objects, it is essential to distinguish these objects themselves from the Web documents that describe them. It is, therefore, common practice to use different URIs to identify the real-world object and the document that describes it, in order to be unambiguous. This practice allows separate statements to be made about an object and about a document that describes that object. For example, the creation date of a person may be rather different to the creation date of a document that describes this person. Being able to distinguish the two through use of different URIs is critical to the coherence of the Web of Data.

3.2.3 Provide useful information, using the RDF standard, for looking up for URIs.

In order to enable a wide range of different applications to process Web content, it is important to agree on standardised content formats. The agreement on HTML as a dominant document format was an important factor that made the Web scale. The third Linked Data principle therefore advocates use of a single data model for publishing structured data on the Web – the Resource Description Framework (RDF).

RDF provides a graph-based data model that is extremely simple on the one hand but strictly tailored towards Web architecture on the other hand. RDF itself is just describing the data model, it does not address the format in which the data is eventually stored and transferred. To be published on the Web, RDF data can be serialised in different formats. The two RDF serialisation formats most commonly used to publish Linked Data on the Web are RDF/XML and RDFa.

3.2.4 Include links to other URIs, so that they can discover more things.

This Linked Data principle advocates the use of hyperlinks to connect not only Web documents, but also any other type of thing. For example, a hyperlink may be set between a person and a place, or between a place and a company. Hyperlinks that connect things in a Linked Data context have types, which describe the relationship between the things. For example, a hyperlink of the type "friend-of" may be set between two people, or a hyperlink of the type "based-near" may be set between a person and a place. Hyperlinks in the Linked Data context are called RDF links to distinguish them from untyped hyperlinks between classic Web documents. The fourth Linked Data principle is to set RDF links pointing into other data sources on the Web. Such external RDF links are fundamental for the Web of Data as they are the glue that connects different data repositories into a global, interconnected data space.

3.3 Semantics for the Finance and Insurance Sector

Semantic Web technologies address the limitation in usage of XML, which means that ad-hoc mapping of different schemas are needed to integrate data. Here, neither semantic interoperability is enabled nor is reasoning supported. This weakness is addressed by using machine-understandable descriptions of resources, which do not require any ad-hoc schema. The de-facto standard as a representation model is RDF and the meaning of each term can be determined automatically by checking the corresponding vocabulary definition.

Currently, a wide array of data representation standards for finance and insurance applications have emerged as a means of enabling data interoperability and data exchange between different systems/applications. Prominent examples include: (i) The Financial Industry Business Ontology (FIBO®) [FIBO-MDF] by OMG and the Enterprise Data Management (EDM) Council, which defines financial industry terms, definitions and synonyms using semantic web principles such as RDF/OWL and widely adopted OMG modelling standards such as UML; (ii) The Financial Instrument Global Identifier (FIGI®) [FIGI] which aims at unifying terms and definitions for financial security and related contextual information used during trade negotiation, execution, settlement, and clearing processes; (iii) FinRegOnt is a core ontology integrating legal and financial information, which is based on the integration of concepts from FIBO and the Legal Knowledge Interchange Format (LKIF) [LKIF].

These standards provide the means for common representation of domain specific datasets, which provide the means for data interoperability (including in several cases semantic interoperability) across diverse databases and datasets. However, they are still not widely deployed by financial organizations, as they are not accompanied by proper tools for high-performance semantic querying that could be used in analytics applications for the finance and insurance sectors. Existing tools and inference engines for semantic reasoning (such as the CEL DL (Description Logic) reasoner, the Euler inference engine, the FaCT++ OWL-DL reasoned, the HermiT18 OWL reasoned and the JESS (Java Expert System Shell)) cannot be easily deployed in massively parallelized cloud environments as required for dealing with large scale semantic datasets.

3.4 Mash-up Building Features

Existing BigData/IoT applications in the financial and insurance sectors form in most cases disaggregated (data) “silos”, which are hardly interoperable with systems and application of other financial institutions and administrative domains. Likewise, there is very poor interoperability across the diverse datasets that are typically collected and used in financial/insurance applications (including FinTech and InsuranceTech applications).

A proposition in INFINITECH is to introduce building blocks in the form of tools for semantic interoperability and interoperable data exchange capabilities, as means of facilitating the development and deployment of innovative applications that span multiple systems and stakeholders in the financial supply chain (e.g., cross-border transactions, SWIFT network payments analytics, as well as a variety blockchain applications). In addition to the core INFINITECH technology building blocks for data interoperability, data management and analytics, it is expected that INFINITECH pilots will take advantage of the data modelling that can be used in other specific-related analytics services.

4 SeSA-ME Specification and Implementation

INFINITECH devises a semantic interoperability solution based on a combination of concepts from FIBO, FIGI and LKIF as well as based on the selective enhancement of these ontologies with new concepts as needed by the project's pilots use cases. SeSA-ME solution is designed in the form to be a shared semantics solution, which will take advantage of transformation of data schemas to our common INFINITECH semantics.

4.1 SeSA-ME Architecture

Leveraging on NUIG's Super Stream Collider (SSC) solution, the INFINITECH project is providing the means for the deployment and provisioning of semantic reasoning and analytics capabilities in massive, distributed computing systems (i.e., large scale cloud data centres such as those hosting the INFINITECH testbeds) by implementing the Semantics Stream Analytics Middleware-Engine (SeSA-ME). In this way, INFINITECH's SeSA-ME aims for offering capabilities for live semantic data processing and on-demand access to smart semantic analytics services. Figure 14 depicts the SeSA-ME Architecture where, it is observed the different components and how it interacts with data sources alike it provides data sharing applications.

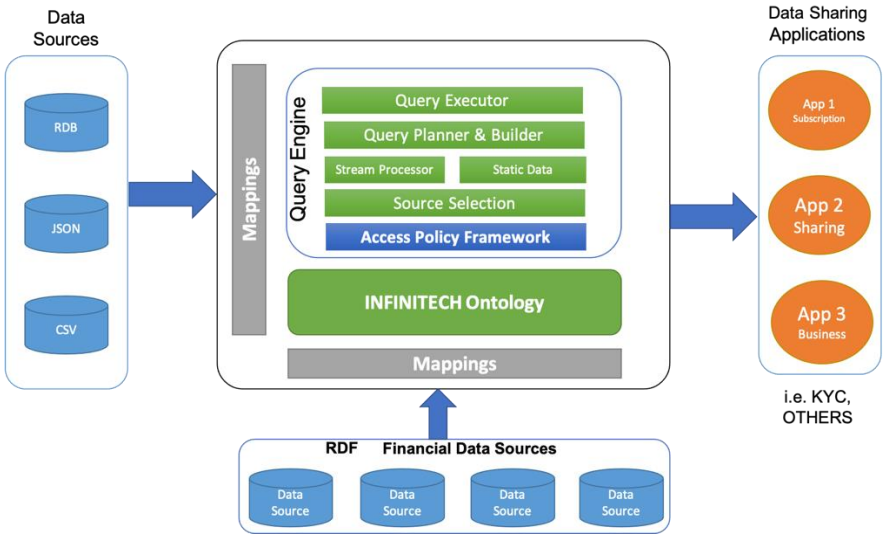


Figure 14. Semantic Stream Analytics Middleware-Engine Architecture

The high-performance semantic stream analytics functionalities of the SeSA-ME component are made available through Open APIs and will be deployed on the project's sandboxes and testbeds as, in this section the specification for the different blocks of the SeSA-ME engine are described.

4.1.1 Source Selection

Table 1: Source Selection - Component Description and API Documentation

Attribute	Documentation & Example
Component ID	INF-DSM-130-S

Component Name	Source Selection
Description	This component is responsible for selecting data sources which could potentially return results for a given request. Usually there are many data sources available to get data from but all of them might not be relevant for the request. Hence sending requests to all of them would incur extra load on the data sources and the SeSA-ME engine and would cause delay in response to the request. So, identifying relevant data sources for a request is important to avoid any delays and unnecessary requests to data sources. The source selection process is performed based on the availability of pre-processed information, e.g., meta data, from data sources or availability of mechanisms to inquire about information from data sources at run-time. The identification of selecting relevant sources for a request will also contribute to building requests for each individual data source.
Icon	N/A
IP Owner & Partner in Charge	NUIG
INFINITECH Component Category	Data Semantics
IRA - BDVA Layer	Data Processing
Input (Required by the Component)	Request in JSON format, list of data sources and meta-data of data sources.
Output (Produced by the Component)	List of relevant data sources against data requested.
Technology or Platform to be used	Java
Part of INFINITECH Core	Yes

MARKETPLACE	Yes if it will be part of the Marketplace
Microservice	Yes if it is a dockerized microservice component
Endpoint/REST API	To be defined
License	To be defined
Other Information / Remarks	N/A
Detailed Documentation	N/A

4.1.2 Query Planner

Table 2: Query Planner – Component Description and API Documentation

Attribute	Documentation & Example
Component ID	INF-DSM-131-S
Component Name	Query Planner
Description	The query planner identifies the order in which the queries will be executed on the relevant data sources. This step is performed after the source selection step. The inputs from the source selection component is utilised to plan the queries, their order and the data source on which each query will be executed.
Icon	N/A

IP Owner & Partner in Charge	NUIG
INFINITECH Component Category	Data Semantics
IRA - BDVA Layer	Data Processing
Input (Required by the Component)	List of queries and list of data sources on which the query will be executed.
Output (Produced by the Component)	Query plan
Technology or Platform to be used	Java
Part of INFINITECH Core	Yes
MARKETPLACE	Yes, it will be part of the Marketplace
Microservice	Yes, it is a dockerized microservice component
Endpoint/REST API	To be defined
License	To be defined
Other Information / Remarks	N/A
Detailed Documentation	N/A

4.1.3 Query Builder

Table 3: Query Builder - Component Description and API Documentation

Attribute	Documentation & Example
Component ID	INF-DSM-134-S
Component Name	Query Builder
Description	As its name suggests, the query builder will build the actual queries, as identified in the query planning step, from existing query templates. For example, customer profile building queries.
Icon	N/A
IP Owner & Partner in Charge	NUIG
INFINITECH Component Category	Data Semantics
IRA - BDVA Layer	Data Processing
Input (Required by the Component)	Query Plan
Output (Produced by the Component)	SPARQL query or CQELS or C-SPARQL query
Technology or Platform to be used	Java
Part of INFINITECH Core	Yes

MARKETPLACE	Yes, it will be part of the Marketplace
Microservice	Yes, it is a dockerized microservice component
Endpoint/REST API	To be defined
License	To be defined
Other Information / Remarks	N/A
Detailed Documentation	N/A

4.1.4 Query Executor

Table 4: Query Executor - Component Description and API Documentation

Attribute	Documentation & Example
Component ID	INF-DSM-132-S
Component Name	Query Executor
Description	The query executor component will be responsible for executing the queries generated based on the API templates on the desired data source. For example, executing SPARQL query using Jena ARQ library on a data source, e.g. triple store.
Icon	N/A
IP Owner & Partner in Charge	NUIG

INFINITECH Component Category	Data Semantics
IRA - BDVA Layer	Data Processing
Input (Required by the Component)	SPARQL query and data source on which the query will be executed.
Output (Produced by the Component)	Result set in JSON format.
Technology or Platform to be used	Java
Part of INFINITECH Core	Yes
MARKETPLACE	Yes, it will be part of the Marketplace
Microservice	Yes, it is a dockerized microservice component
Endpoint/REST API	To be defined
License	To be defined
Other Information / Remarks	N/A
Detailed Documentation	N/A

4.1.5 Stream Processor

Table 5: Stream Processor - Component Description and API Documentation

Attribute	Documentation & Example
Component ID	INF-DSM-133-S
Component Name	Stream Processor
Description	This component will be responsible for managing RDF streams of data coming from streaming data sources. It will execute queries on streaming data and provide streams of output data to the requesting entity, based on the frequency and time frame specified in the query. For example, executing a CQEL or C-SPARQL query using a stream processing engine.
Icon	N/A
IP Owner & Partner in Charge	NUIG
INFINITECH Component Category	Data Semantics
IRA - BDVA Layer	Data Processing
Input (Required by the Component)	C-SPARQL or CQELS query and data source on which the query will be executed.
Output (Produced by the Component)	Data streams in JSON format
Technology or Platform to be used	Java.

Part of INFINITECH Core	Yes
MARKETPLACE	Yes, it will be part of the Marketplace
Microservice	Yes, it is a dockerized microservice component
Endpoint/REST API	To be defined
License	To be defined
Other Information / Remarks	N/A
Detailed Documentation	N/A

4.1.6 Access Policy Framework

Table 6: Access Policy Framework - Component Description and API Documentation

Attribute	Documentation & Example
Component ID	INF-DSM-135-S
Component Name	Access Policy Framework
Description	The access policy framework will be used to perform authorization of users based on the access policy rules defined. This component works after the authentication step which is not part of this. This component is composed of user profiles and access policies. User profiles should be stored and access policies on the underlying data based on the user profiles must be defined, initialised and stored.
Icon	N/A

IP Owner & Partner in Charge	NUIG
INFINITECH Component Category	Data Semantics
IRA - BDVA Layer	Data Processing
Input (Required by the Component)	SPARQL or CQELS query, data source and user information.
Output (Produced by the Component)	Boolean flag which will represent whether access is granted or denied.
Technology or Platform to be used	N/A
Part of INFINITECH Core	Yes
MARKETPLACE	Yes, it will be part of the Marketplace
Microservice	Yes, it is a dockerized microservice component
Endpoint/REST API	To be confirmed
License	To be confirmed
Other Information / Remarks	N/A
Detailed Documentation	N/A

4.2 SeSA-ME APIs

The high-performance semantic analytics functionalities of the project are made available through Open APIs and will be deployed on the project's sandboxes and testbeds as described in the following sections. The APIs provided by SeSA-ME Engine are divided into two categories, namely Static Data APIs and Streaming Data APIs.

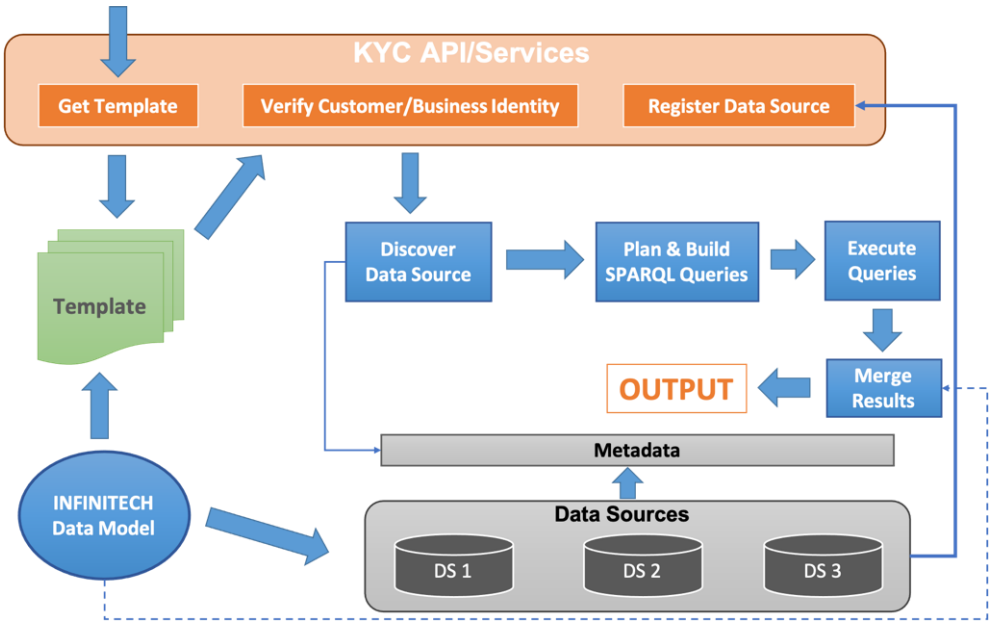


Figure 15. Semantic Stream Analytics Middleware-Engine API Services

4.2.1 Static Data APIs – Version I

4.2.1.1 Know Your Customer (KYC) Profiler

Know Your Customer (KYC) is the process where businesses can verify the identity of their customer to ascertain the legitimacy and credibility. The KYC process is mostly used by financial institutions, such as banks, insurance companies etc. to verify their customers. This section describes RESTful APIs provided by SeSA-ME Engine for the KYC use case.

There are two perspectives of KYC, one is KYC Data Consumer, the consumer's perspective of KYC services and the other is KYC Data Provider, the data provider's perspective of KYC services. In the former, financial institutions consume the KYC services provided to verify the identity of their customers and in the later data providers provide their data to be used as a source for verifying the identity of customers.

4.2.1.1.1 KYC Data Providers

This section describes the KYC APIs provided for the data providers, whose data can be used to verify the identity of customers. To be able to become a data provider for KYC services, the data source must get registered with SeSA-ME Engine.

4.2.1.1.1.1 Data Source Registration API

The data source registration API is used to register a data source with SeSA-ME engine for the purpose of providing their data to be used for customer identity verification. The data source must

supply the required information to this API. This information includes attributes, such as “name”, “type” and “params” for this data source. The “type” attribute specifies the type of data source, e.g., SPARQL Endpoint, Data World Endpoint, Graph DB endpoint etc. The “params” attribute specifies the parameters needed to access the data source, e.g., access URL, username, passwords etc.

Table 7: Example Data Source Registration Information

Attributes	Values
name	Bank of Ireland
type	SPARQL_ENDPOINT
accessURL	http://localhost:8890/sparql

The details needed to use the data source registration API are listed in the table below. This table lists the example input and output in the form of JSON along with their JSON schemas.

Table 8: Example Register Data Source Functionality and URL notation

Functionality:	**Register a Data Source**
URL:	**/registerDatasource**
Method:	**POST**
Registers a data source whose data can be used by KYC Data Consumers for verifying the identity of their customer.	

Table 9: Example KYC Data Consumer Method using JSON Schema

Input	JSON example	{
		"name": "DS-1",
		"type": "SPARQL_ENDPOINT",
		"params": {
		"accessURL": "http://localhost:8890/sparql"
		}
		}

	JSON schema	{
		"type": "object",
		"properties": {
		"name": {
		"type": "string"
		},
		"type": {
		"type": "string"
		},

```
  "params": {
   "type": "object",
   "properties": {
    "accessURL": {
     "type": "string"
    }
   },
   "required": [
    "accessURL"
   ]
  }
 },
 "required": [
  "name",
  "type",
  "params"
 ]
}
```


Output

JSON example


```
{
 "message": "Data source is registered successfully."
}
```


JSON schema


```
{
 "type": "object",
 "properties": {
  "message": {
   "type": "string"
  }
 },
 "required": [
  "message"
 ]
}
```


4.2.1.1.2 KYC Data Consumers

This section describes the KYC APIs provided for KYC data consumers, who can use these APIs to verify the identity of a customer. We have identified two scenarios in the KYC use case, i.e. Identity verification and Business Verification, described in the next sections. Templates for KYC Consumers

Table 10: Example Template for Identity Verification

Attributes	Values
identifier	ABC-12345
firstName	Martin
middleName	Serrano
surname	Orozco
dateOfBirth	12-01-1975
gender	Male

addressline1	House No. 111
addressline2	Lower Dangan
addressline3	Newcastle
city	Galway
postalCode	SE06

Table 11: Example Template for Business Verification

Attributes	Values
registrationNumber	HDBAKSOWI12839HGD4747
businessName	XYZ Inc.
dateOfIncorporation	12-12-2012
addressline1	Building No. 13
addressline2	IDA Business Park
addressline3	Newcastle
city	Galway
postalCode	SE06

4.2.1.1.2.1 Get Template API (Identity Verification)

Table 12: Example Get Template Functionality and URL notation

Functionality:	**Get Templates**
URL:	**/getTemplate**
Method:	**POST**

Get the template that should be provided for verification of an identity or any other purpose.

Table 13: Example Identity Verification method using JSON Schema

Input	JSON example	{
		"fieldsFor": "Identity Verification"
		}

JSON schema

```
{
 "title": "ListFields",
 "type": "object",
 "properties": {
  "fieldsFor": {
   "title": "fieldsFor"
   "type": "string",
   "description": "The purpose for which fields are requested"
  }
 }
}
```

Output

JSON example

```
{
 "dataFields": {
  "person": {
   "type": "object",
   "attributes": {
    "identifier": {
     "type": "string"
    },
    "firstName": {
     "type": "string"
    },
    "middleName": {
     "type": "string"
    },
    "surname": {
     "type": "string"
    },
    "maidenName": {
     "type": "string"
    },
    "dateOfBirth": {
     "type": "date"
    },
    "gender": {
     "type": "string"
    },
    "physicalAddress": {
     "type": "object",
     "attributes": {
      "addressline1": {
       "type": "string"
      },
      "addressline2": {
       "type": "string"
      },
      "addressline3": {
       "type": "string"
      },
      "city": {
       "type": "string"
      },
      "postalCode": {
       "type": "string"
      }
     }
    }
   }
  }
 }
}
```

JSON schema

```
{
 "title": "DataFields",
 "type": "object",
 "properties": {
  "dataFields": {
   "type": "object",
   "properties": {
    "person": {
     "type": "object",
     "properties": {
      "type": {
       "type": "string"
      },
      "attributes": {
       "type": "object",
       "properties": {
        "identifier": {
         "type": "object",
         "properties": {
          "type": {
           "type": "string"
          }
         }
        },
        "firstName": {
         "type": "object",
         "properties": {
          "type": {
           "type": "string"
          }
         }
        },
        "middleName": {
         "type": "object",
         "properties": {
          "type": {
           "type": "string"
          }
         }
        },
        "surname": {
         "type": "object",
         "properties": {
          "type": {
           "type": "string"
          }
         }
        },
        "maidenName": {
         "type": "object",
         "properties": {
          "type": {
           "type": "string"
          }
         }
        },
        "dateOfBirth": {
         "type": "object",
         "properties": {
          "type": {
           "type": "string"
          }
         }
        },
```

```
        "gender": {
         "type": "object",
         "properties": {
          "type": {
           "type": "string"
          }
         }
        },
        "physicalAddress": {
         "type": "object",
         "properties": {
          "type": {
           "type": "string"
          },
          "attributes": {
           "type": "object",
           "properties": {
            "addressline1": {
             "type": "object",
             "properties": {
              "type": {
               "type": "string"
              }
             }
            },
            "addressline2": {
             "type": "object",
             "properties": {
              "type": {
               "type": "string"
              }
             }
            },
            "addressline3": {
             "type": "object",
             "properties": {
              "type": {
               "type": "string"
              }
             }
            },
            "city": {
             "type": "object",
             "properties": {
              "type": {
               "type": "string"
              }
             }
            },
            "postalCode": {
             "type": "object",
             "properties": {
              "type": {
               "type": "string"
              }
             }
            }
           }
          }
         }
        }
       }
      }
```

```
      }
     }
    }
   }
}
```

4.2.1.1.2.2 Get Templates API (Business Verification)

Table 14: Example Get List of Fields Functionality and URL notation

Functionality:	**Get List of Fields**
URL:	**/listFields**
Method:	**POST**

Get the list of fields that should be provided for verification of a business or any other purpose.

Table 15: Example Business Verification method using JSON Schema

Input	JSON example	{
		"fieldsFor": "Business Verification"
		}

	JSON schema	{
		"type": "object",
		"properties": {
		"fieldsFor": {
		"title": "fieldsFor"
		"type": "string",
		"description": "The purpose for which fields are requested"
		}
		}
		}

Output

JSON example


```
{
 "dataFields": {
  "business": {
   "type": "object",
   "attributes": {
    "registrationNumber": {
     "type": "string"
    },
    "businessName": {
     "type": "string"
    },
    "dateOfIncorporation": {
     "type": "date"
    },
    "physicalAddress": {
     "type": "object",
     "attributes": {
      "addressline1": {
       "type": "string"
      },
      "addressline2": {
       "type": "string"
      },
      "addressline3": {
       "type": "string"
      },
      "city": {
       "type": "string"
      },
      "postalCode": {
       "type": "string"
      }
     }
    }
   }
  }
 }
}
```


JSON schema


```
{
 "type": "object",
 "properties": {
  "dataFields": {
   "type": "object",
   "properties": {
    "business": {
     "type": "object",
     "properties": {
      "type": {
       "type": "string"
      },
      "attributes": {
       "type": "object",
       "properties": {
        "registrationNumber": {
         "type": "object",
         "properties": {
          "type": {
           "type": "string"
          }
         }
        },
        "businessName": {
         "type": "object",
```

```
        "properties": {
         "type": {
          "type": "string"
         }
        }
       },
       "dateOfIncorporation": {
        "type": "object",
        "properties": {
         "type": {
          "type": "string"
         }
        }
       },
       "physicalAddress": {
        "type": "object",
        "properties": {
         "type": {
          "type": "string"
         },
         "attributes": {
          "type": "object",
          "properties": {
           "addressline1": {
            "type": "object",
            "properties": {
             "type": {
              "type": "string"
             }
            }
           },
           "addressline2": {
            "type": "object",
            "properties": {
             "type": {
              "type": "string"
             }
            }
           },
           "addressline3": {
            "type": "object",
            "properties": {
             "type": {
              "type": "string"
             }
            }
           },
           "city": {
            "type": "object",
            "properties": {
             "type": {
              "type": "string"
             }
            }
           },
           "postalCode": {
            "type": "object",
            "properties": {
             "type": {
              "type": "string"
             }
            }
           }
          }
```

```
            }
           }
          }
         }
        }
       }
      }
     }
    }
   }
}
```


4.2.1.1.3 Identity Verification

4.2.1.1.3.1 Verify Identity API

Table 16: Example Verify Customer Identity Functionality and URL notation

Functionality:	**Verify Customer Identity**
URL:	**/verifyIdentity**
Method:	**POST**

Verify the identity of a customer based on the customer information provided to the API.

Table 17: Example Verify Customer Identity method using JSON Schema

Input

JSON example


```
{
 "dataFields": {
  "person": {
   "identifier": "ABC 12345",
   "firstName": "Martin",
   "middleName": "Serrano",
   "surname": "Orozco",
   "dateOfBirth": "12-01-1975",
   "gender": "Male",
   "physicalAddress": {
    "addressline1": "House No. 111",
    "addressline2": "Lower Dangan",
    "addressline3": "Newcastle",
    "city": "Galway",
    "postalCode": "SE06"
   }
  }
 }
}
```


JSON schema


```
{
 "type": "object",
 "properties": {
  "dataFields": {
   "type": "object",
   "properties": {
```

```
    "person": {
     "type": "object",
     "properties": {
      "identifier": {
       "type": "string"
      },
      "firstName": {
       "type": "string"
      },
      "middleName": {
       "type": "string"
      },
      "surname": {
       "type": "string"
      },
      "maidenName": {
       "type": "string"
      },
      "dateOfBirth": {
       "type": "string"
      },
      "gender": {
       "type": "string"
      },
      "physicalAddress": {
       "type": "object",
       "properties": {
        "addressline1": {
         "type": "string"
        },
        "addressline2": {
         "type": "string"
        },
        "addressline3": {
         "type": "string"
        },
        "city": {
         "type": "string"
        },
        "postalCode": {
         "type": "string"
        }
       }
      }
     }
    }
   }
  }
}
```

Output JSON example


```
{
 "verificationId": "XYZ-22222-5555-DDD",
 "verificationDate": "2020-12-01T11:50:23",
 "verification": {
  "verificationStatus": "verified",
  "verificationResults": [
   {
    "verifiedFrom": "BOI",
    "verifiedAttributes": [
     {
      "attribute": "identifier",
      "status": "verified"
     },
     {
      "attribute": "firstName",
      "status": "verified"
     },
     {
      "attribute": "middleName",
      "status": "verified"
     },
     {
      "attribute": "surname",
      "status": "verified"
     },
     {
      "attribute": "dateOfBirth",
      "status": "verified"
     },
     {
      "attribute": "gender",
      "status": "verified"
     },
     {
      "attribute": "addressline1",
      "status": "verified"
     },
     {
      "attribute": "addressline2",
      "status": "verified"
     },
     {
      "attribute": "addressline3",
      "status": "verified"
     },
     {
      "attribute": "city",
      "status": "verified"
     },
     {
      "attribute": "postalCode",
      "status": "verified"
     }
    ]
   }
  ],
  "errors": [],
  "rule": {
   "ruleName": "",
   "ruleDescription": ""
  }
 }
}
```

JSON schema


```
{
 "type": "object",
 "properties": {
  "verificationId": {
   "type": "string"
  },
  "verificationDate": {
   "type": "string"
  },
  "verification": {
   "type": "object",
   "properties": {
    "verificationStatus": {
     "type": "string"
    },
    "verificationResults": {
     "type": "array",
     "items": [
      {
       "type": "object",
       "properties": {
        "verifiedFrom": {
         "type": "string"
        },
        "verifiedAttributes": {
         "type": "array",
         "items": [
          {
           "type": "object",
           "properties": {
            "attribute": {
             "type": "string"
            },
            "status": {
             "type": "string"
            }
           }
          },
          {
           "type": "object",
           "properties": {
            "attribute": {
             "type": "string"
            },
            "status": {
             "type": "string"
            }
           }
          },
          {
           "type": "object",
           "properties": {
            "attribute": {
             "type": "string"
            },
            "status": {
             "type": "string"
            }
           }
          },
          {
           "type": "object",
           "properties": {
            "attribute": {
```

```
        "type": "string"
      },
      "status": {
        "type": "string"
      }
    }
  },
  {
    "type": "object",
    "properties": {
      "attribute": {
        "type": "string"
      },
      "status": {
        "type": "string"
      }
    }
  },
  {
    "type": "object",
    "properties": {
      "attribute": {
        "type": "string"
      },
      "status": {
        "type": "string"
      }
    }
  },
  {
    "type": "object",
    "properties": {
      "attribute": {
        "type": "string"
      },
      "status": {
        "type": "string"
      }
    }
  },
  {
    "type": "object",
    "properties": {
      "attribute": {
        "type": "string"
      },
      "status": {
        "type": "string"
      }
    }
  },
  {
    "type": "object",
    "properties": {
      "attribute": {
        "type": "string"
      },
      "status": {
        "type": "string"
      }
    }
  },
  {
    "type": "object",
```

```
                "properties": {
                 "attribute": {
                  "type": "string"
                 },
                 "status": {
                  "type": "string"
                 }
                }
               },
               {
                "type": "object",
                "properties": {
                 "attribute": {
                  "type": "string"
                 },
                 "status": {
                  "type": "string"
                 }
                }
               }
              ]
             }
            }
           }
          ]
         },
         "errors": {
          "type": "array",
          "items": {}
         },
         "rule": {
          "type": "object",
          "properties": {
           "ruleName": {
            "type": "string"
           },
           "ruleDescription": {
            "type": "string"
           }
          }
         }
        }
       }
      }
     }
    }
```


4.2.1.1.4 Business Verification

4.2.1.1.4.1 **Verify Business API**

Table 18: Example Verify Business API Functionality and URL notation

Functionality:	**Verify Business**
URL:	**/verifyBusiness**

Method:	**POST**

Verify a business based on the business information provided to the API.

Table 19: Example Verify Business method using JSON Schema

Input

JSON example

```
{
 "dataFields": {
  "business": {
   "registrationNumber": "HDBAKSOWI12839HGD4747",
   "businessName": "XYZ Inc.",
   "dateOfIncorporation": "12-12-2012",
   "physicalAddress": {
    "addressline1": "Building No. 13",
    "addressline2": "IDA Business Park",
    "addressline3": "Newcastle",
    "city": "Galway",
    "postalCode": "SE06"
   }
  }
 }
}
```

JSON schema

```
{
 "type": "object",
 "properties": {
  "dataFields": {
   "type": "object",
   "properties": {
    "business": {
     "type": "object",
     "properties": {
      "registrationNumber": {
       "type": "string"
      },
      "businessName": {
       "type": "string"
      },
      "dateOfIncorporation": {
       "type": "date"
      },
      "physicalAddress": {
       "type": "object",
       "properties": {
        "addressline1": {
         "type": "string"
        },
        "addressline2": {
         "type": "string"
        },
        "addressline3": {
         "type": "string"
        },
        "city": {
         "type": "string"
        },
        "postalCode": {
         "type": "string"
        }
       }
      }
     }
    }
   }
  }
 }
}
```

```
      }
     }
    }
   }
}
```


Output JSON example


```
{
 "verificationId": "ASD-1133-333-456",
 "verificationDate": "2020-12-01T11:50:23",
 "verification": {
  "verificationStatus": "verified",
  "verificationResults": [
   {
    "verifiedFrom": "BOI",
    "verifiedAttributes": [
     {
      "attribute": "registrationNumber",
      "status": "verified"
     },
     {
      "attribute": "businessName",
      "status": "verified"
     },
     {
      "attribute": "dateOfIncorporation",
      "status": "verified"
     },
     {
      "attribute": "addressline1",
      "status": "verified"
     },
     {
      "attribute": "addressline2",
      "status": "verified"
     },
     {
      "attribute": "addressline3",
      "status": "verified"
     },
     {
      "attribute": "city",
      "status": "verified"
     },
     {
      "attribute": "postalCode",
      "status": "verified"
     }
    ]
   }
  ],
  "errors": [],
  "rule": {
   "ruleName": "",
   "ruleDescription": ""
  }
 }
}
```

JSON schema


```
{
 "type": "object",
 "properties": {
  "verificationId": {
   "type": "string"
  },
  "verificationDate": {
   "type": "string"
  },
  "verification": {
   "type": "object",
   "properties": {
    "verificationStatus": {
     "type": "string"
    },
    "verificationResults": {
     "type": "array",
     "items": [
      {
       "type": "object",
       "properties": {
        "verifiedFrom": {
         "type": "string"
        },
        "verifiedAttributes": {
         "type": "array",
         "items": [
          {
           "type": "object",
           "properties": {
            "attribute": {
             "type": "string"
            },
            "status": {
             "type": "string"
            }
           }
          },
          {
           "type": "object",
           "properties": {
            "attribute": {
             "type": "string"
            },
            "status": {
             "type": "string"
            }
           }
          },
          {
           "type": "object",
           "properties": {
            "attribute": {
             "type": "string"
            },
            "status": {
             "type": "string"
            }
           }
          },
          {
           "type": "object",
           "properties": {
            "attribute": {
```

```
                "type": "string"
               },
               "status": {
                "type": "string"
               }
              }
             },
             {
              "type": "object",
              "properties": {
               "attribute": {
                "type": "string"
               },
               "status": {
                "type": "string"
               }
              }
             },
             {
              "type": "object",
              "properties": {
               "attribute": {
                "type": "string"
               },
               "status": {
                "type": "string"
               }
              }
             },
             {
              "type": "object",
              "properties": {
               "attribute": {
                "type": "string"
               },
               "status": {
                "type": "string"
               }
              }
             },
             {
              "type": "object",
              "properties": {
               "attribute": {
                "type": "string"
               },
               "status": {
                "type": "string"
               }
              }
             }
            ]
           }
          }
         }
        ]
       },
       "errors": {
        "type": "array",
        "items": {}
       },
       "rule": {
        "type": "object",
        "properties": {
```

```
        "ruleName": {
          "type": "string"
        },
        "ruleDescription": {
          "type": "string"
        }
      }
    }
  }
 }
}
```


4.2.2 Streaming Data APIs – Version I

The second component provided by SeSA-ME Engine is the stream processor, which is responsible for processing multiple available linked data streams and providing the results to the consumers of data streams. This section describes the SeSA-ME APIs for streaming data.

4.2.2.1 Stream Registration

SeSA-ME Engine can process multiple streams available and to consume the available streams of data, the consumer first needs to register for these streams. The consumer needs to have the stream Ids and also callback URL for receiving back the stream. The callback URL should be a RESTful API and the streaming data should be received at this API. The technical details are provided in the next section.

4.2.2.1.1 Register for Streams API

This API is used for registering for linked streams. The example JSON inputs and outputs along with their JSON schemas are provided below.

Table 20: Example Register for Streams API Functionality and URL notation

Functionality:	**Register for Streams**
URL:	**/registerForStream**
Method:	**POST**

Registers for a stream or a list of streams.

Table 21: Example Register for Streams method using JSON Schema

JSON Example

```
{
 "callbackURL": "http://localhost/sesame-client/getRDFStream",
 "streams": [
  {
   "streamId": "http://infinitech.eu/rdf/stream-2",
  },
  {
   "streamId": "http://infinitech.eu/rdf/stream-4",
  }
 ]
}
```

```
{
 "type": "object",
 "properties": {
  "callbackURL": {
   "type": "string"
  },
  "streams": {
   "type": "array",
   "items": [
    {
     "type": "object",
     "properties": {
      "streamId": {
       "type": "string"
      }
     },
     "required": [
      "streamId"
     ]
    },
    {
     "type": "object",
     "properties": {
      "streamId": {
       "type": "string"
      }
     },
     "required": [
      "streamId"
     ]
    }
   ]
  }
 },
 "required": [
  "callbackURL",
  "streams"
 ]
}
```

```
{
 "message": "You have successfully registered for the requested streams."
}
```

```
{
 "type": "object",
 "properties": {
  "message": {
   "type": "string"
  }
 },
 "required": [
  "message"
 ]
}
```

4.2.3 SeSA-ME New APIS Implementation – Version II

The second version implementation component provided by SeSA-ME Engine is the stream processor, which is responsible for processing multiple available linked data streams and providing the results to the consumers of data streams. This section describes the SeSA-ME APIs for streaming data.

4.2.3.1 Stream Registration II

SeSA-ME Engine can process multiple streams available and to consume the available streams of data, the consumer first needs to register for these streams. The consumer needs to have the stream Ids and also callback URL for receiving back the stream. The callback URL should be a RESTful API and the streaming data should be received at this API. The technical details are provided in the next section.

4.2.3.1.1 Unregister Data Source API

The details needed to use the un-register data source API are listed in the table below. This table lists the example input and output in the form of JSON along with their JSON schemas.

Table 22: Example for Unregister Data Source API Functionality and URL notation

Functionality:	**Un-register a Data Source**
URL:	**/unregisterDatasource**
Method:	**POST**

The method to unregister a data source already registered with SeSA-ME and what data can be used.

Table 23: Example unregister a Data Source method using JSON Schema

Input	JSON example	{
		"datasource": "http://localhost:8890/sparql"
		}

	JSON schema	{
		"type": "object",
		"properties": {
		"datasource": {
		"type": "string"
		}
		},
		"required": [
		"datasource"
		]
		}

Output	JSON example	{
		"message": "Data source is un-registered successfully."
		}

	JSON schema	{
		"type": "object",
		"properties": {
		"message": {
		"type": "string"
		}
		},
		"required": [
		"message"
		]
		}

4.2.3.1.2 Run Query API

The details needed to use the run query API are listed in the table below. This table lists the example input and output in the form of JSON along with their JSON schemas.

Table 24: Example for Unregister Data Source API Functionality and URL notation

Functionality:	**Run Query on a Data Source**
URL:	**/runQuery**
Method:	**POST**

Runs a SPARQL query on the specified data source and returns back the obtained results.

Table 25: Example Runs a SPARQL query on the specified data source

Input — JSON example

```
{
 "datasource": "http://localhost:8890/sparql",
 "query": "SELECT * WHERE { ?s ?p ?o . } LIMIT 1"
}
```

Input — JSON schema

```
{
 "type": "object",
 "properties": {
  "datasource": {
   "type": "string"
  },
  "query": {
   "type": "string"
  }
 },
 "required": [
  "datasource",
  "query"
 ]
}
```

Output — JSON example

```
{
 "head": {
  "vars": [
   "s",
   "p",
   "o"
  ]
 },
 "results": {
  "bindings": [
   {
    "s": {
     "type": "uri",
     "value": "http://infinitechproject.eu/data/joabos"
    },
    "p": {
     "type": "uri",
     "value": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
    },
    "o": {
     "type": "uri",
     "value": "https://spec.edmcouncil.org/fibo/ontology/FND/AgentsAndPeople/People/Person"
    }
   }
  ]
 }
}
```

JSON schema

```
{
 "type": "object",
 "properties": {
  "head": {
   "type": "object",
   "properties": {
    "vars": {
     "type": "array",
     "items": [
      {
       "type": "string"
      },
      {
       "type": "string"
      },
      {
       "type": "string"
      }
     ]
    }
   },
   "required": [
    "vars"
   ]
  },
  "results": {
   "type": "object",
   "properties": {
    "bindings": {
     "type": "array",
     "items": [
      {
       "type": "object",
       "properties": {
        "s": {
         "type": "object",
         "properties": {
          "type": {
           "type": "string"
          },
          "value": {
           "type": "string"
          }
         },
         "required": [
          "type",
          "value"
         ]
        },
        "p": {
         "type": "object",
         "properties": {
          "type": {
           "type": "string"
          },
          "value": {
           "type": "string"
          }
         },
         "required": [
          "type",
          "value"
         ]
        },
```

```
            "o": {
             "type": "object",
             "properties": {
              "type": {
               "type": "string"
              },
              "value": {
               "type": "string"
              }
             },
             "required": [
              "type",
              "value"
             ]
            }
           }
          }
         ]
        }
       },
       "required": [
        "bindings"
       ]
      }
     },
     "required": [
      "head",
      "results"
     ]
    }
```


4.2.3.1.3 Run Query Plan API

The details needed to use the run query plan API are listed in the table below. This table lists the example input and output in the form of JSON along with their JSON schemas.

Table 26: Example for Run Query API Functionality and URL notation

Functionality:	**Run a query plan**
URL:	**/runQueryPlan**
Method:	**POST**

Table 27: Example Run a query plan on both static and streaming data sources

Input	JSON example	{
		"staticRequest": [
		{
		"datasource": "http://localhost:8890/sparql",
		"query": "SELECT * FROM <http://infinitechproject.eu/graph> WHERE { ?s ?p ?o . } LIMIT 1"
		}
		],

```
 "streamRequest": [
  {
   "callbackURL": "http://10.196.2.55:8082/sesame-client/getRDFStream",
   "streams": [
    {
     "streamId": "http://infinitech.eu/rdf/stream-1"
    }
   ]
  }
 ]
}
```


JSON schema


```
{
 "type": "object",
 "properties": {
  "staticRequest": {
   "type": "array",
   "items": [
    {
     "type": "object",
     "properties": {
      "datasource": {
       "type": "string"
      },
      "query": {
       "type": "string"
      }
     },
     "required": [
      "datasource",
      "query"
     ]
    }
   ]
  },
  "streamRequest": {
   "type": "array",
   "items": [
    {
     "type": "object",
     "properties": {
      "callbackURL": {
       "type": "string"
      },
      "streams": {
       "type": "array",
       "items": [
        {
         "type": "object",
         "properties": {
          "streamId": {
           "type": "string"
          }
         },
         "required": [
          "streamId"
         ]
        }
       ]
      }
     },
     "required": [
      "callbackURL",
      "streams"
```

```
      ]
     }
    ]
   }
  },
  "required": [
   "staticRequest",
   "streamRequest"
  ]
}
```


Output for static request

JSON example


```
{
  "head": {
    "vars": [
      "s",
      "p",
      "o"
    ]
  },
  "results": {
    "bindings": [
      {
        "s": {
          "type": "uri",
          "value": "http://infinitechproject.eu/data/joabos"
        },
        "p": {
          "type": "uri",
          "value": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
        },
        "o": {
          "type": "uri",
          "value":
"https://spec.edmcouncil.org/fibo/ontology/FND/AgentsAndPeople/People/Person"
        }
      }
    ]
  }
}
```


JSON schema


```
{
  "type": "object",
  "properties": {
    "head": {
      "type": "object",
      "properties": {
        "vars": {
          "type": "array",
          "items": [
            {
              "type": "string"
            },
            {
              "type": "string"
            },
            {
              "type": "string"
            }
          ]
        }
      },
      "required": [
```

```
    "vars"
   ]
  },
  "results": {
   "type": "object",
   "properties": {
    "bindings": {
     "type": "array",
     "items": [
      {
       "type": "object",
       "properties": {
        "s": {
         "type": "object",
         "properties": {
          "type": {
           "type": "string"
          },
          "value": {
           "type": "string"
          }
         },
         "required": [
          "type",
          "value"
         ]
        },
        "p": {
         "type": "object",
         "properties": {
          "type": {
           "type": "string"
          },
          "value": {
           "type": "string"
          }
         },
         "required": [
          "type",
          "value"
         ]
        },
        "o": {
         "type": "object",
         "properties": {
          "type": {
           "type": "string"
          },
          "value": {
           "type": "string"
          }
         },
         "required": [
          "type",
          "value"
         ]
        }
       }
      }
     ]
    }
   },
   "required": [
    "bindings"
```

```
    ]
   }
  },
  "required": [
   "head",
   "results"
  ]
}
```


Output for stream request

JSON example


```
{
  "head": {
   "vars": [
    "s",
    "p",
    "o"
   ]
  },
  "results": {
   "bindings": [
    {
     "s": {
      "type": "uri",
      "value": "http://infinitech.eu/rdf/customer-41"
     },
     "p": {
      "type": "uri",
      "value":
"https://spec.edmcouncil.org/fibo/ontology/FND/AgentsAndPeople/People/hasLastName"
     },
     "o": {
      "datatype": "http://www.w3.org/2001/XMLSchema#string",
      "type": "typed-literal",
      "value": "Last Name 41"
     }
    }
   ]
  }
}
```


JSON schema


```
{
  "type": "object",
  "properties": {
   "head": {
    "type": "object",
    "properties": {
     "vars": {
      "type": "array",
      "items": [
       {
        "type": "string"
       },
       {
        "type": "string"
       },
       {
        "type": "string"
       }
      ]
     }
    },
    "required": [
     "vars"
    ]
   },
```

```
"results": {
 "type": "object",
 "properties": {
  "bindings": {
   "type": "array",
   "items": [
    {
     "type": "object",
     "properties": {
      "s": {
       "type": "object",
       "properties": {
        "type": {
         "type": "string"
        },
        "value": {
         "type": "string"
        }
       },
       "required": [
        "type",
        "value"
       ]
      },
      "p": {
       "type": "object",
       "properties": {
        "type": {
         "type": "string"
        },
        "value": {
         "type": "string"
        }
       },
       "required": [
        "type",
        "value"
       ]
      },
      "o": {
       "type": "object",
       "properties": {
        "datatype": {
         "type": "string"
        },
        "type": {
         "type": "string"
        },
        "value": {
         "type": "string"
        }
       },
       "required": [
        "datatype",
        "type",
        "value"
       ]
      }
     }
    }
   ]
  }
 },
 "required": [
```

```
      "bindings"
     ]
    }
   },
   "required": [
    "head",
    "results"
   ]
  }
```

4.3 Semantic Annotator-Middleware Pre-processing Layer for FinTechs - SAMPLE-FIN

4.3.1 Data Transformation Guide

The following steps are for the purpose of guiding people to transform their data from native format to RDF format. Each step also lists a set of tools which can be used to perform a specific task.

4.3.2 Step 1: Selecting Ontologies

If you want to transform your data to RDF format, the first thing you need to do is to find an ontology which can be used to model your native data in RDF format.

In case of the INFINITECH project, there are several ontologies available. Below is the list of these ontologies.

4.3.2.1 FIBO

The Financial Industry Business Ontology (FIBO) defines the sets of things that are of interest in financial business applications and the ways that those things can relate to one another. In this way, FIBO can give meaning to any data (e.g., spreadsheets, relational databases, XML documents) that describe the business of finance.

Table 28: FIBO Useful Links

Useful Links		
	External	**INFINITECH**
Website	FIBO	FIBO Docs
OWL Files	FIBO OWL Files	FIBO Files

4.3.2.2 FIGI

FIGI is a Financial Industry Global Instrument Identifiers (FIGI) Ontology.

Table 29: FIGI Useful Links

Useful Links		
	External	INFINITECH
Website	FIGI	FIGI Docs
Files		FIGI Files

4.3.2.3 LKIF

The Legal Knowledge Interchange Format (LKIF) is an OWL ontology of legal concepts, allowing legal knowledge bases to be represented in OWL.

Table 30: LKIF Useful Links

Useful Links		
	External	INFINITECH
Website	Project Website	LKIF Docs
LKIF Files	LKIF Github	LKIF FIles
Publications	LKIF Core Ontology	

4.3.2.4 INFINITECH Core

INFINITECH Core defines alignment between FIBO, FIGI & LKIF in a formal way.

Table 31: INFINITECH Core Useful Links

Useful Links	
Website	INFINITECH Core
INFINITECH Core Files	INFINITECH Core FIles

4.3.3 Step 2: Mapping Native Data to Selected Ontologies

When you have selected the ontologies which can be used to model your data, the next step is to specify mapping from entities and attributes in the native data format to entities and attributes in the selected ontologies.

There are some standard mapping languages available which can be used to specify these mappings, such as RML, R2RML etc.

4.3.3.1 RML: RDF Mapping language

RML, a generic mapping language, based on and extending R2RML. The RDF Mapping language (RML) is a mapping language defined to express customized mapping rules from heterogeneous data structures and serializations to the RDF data model. RML is defined as a superset of the W3C-standardized mapping language R2RML, aiming to extend its applicability and broaden its scope, adding support for data in other structured formats. RML follows exactly the same syntax as R2RML; therefore, RML mappings are themselves RDF graphs.

Other than relational databases, currently you can define mappings from sources, such as CSV, TSV, XML and JSON to RDF. Such mappings describe how existing data can be represented using the RDF data model.

Table 32: RDF Mapping Language Useful Links

Useful Links	
Website	RML
Specifications	RML: RDF Mapping Language

4.3.3.2 RML Editor

The RMLEditor offers a Graphical User Interface (GUI) to enable data publishers, who are domain experts, to model knowledge derived from multiple, heterogeneous data sources. The RMLEditor uses RML as its underlying mapping language, offering a uniform GUI to its users to edit rules.

Table 33: RML Editor Useful Links

Useful Links	
Website	RMLEditor
Online Tool:	RMLEditor Web Version

4.3.3.3 R2RML: RDB to RDF Mapping Language

R2RML is a W3C standard to express customized mappings from relational databases to RDF datasets. Such mappings provide the ability to view existing relational data in the RDF data model, expressed in a structure and target vocabulary of the mapping author's choice. R2RML mappings are themselves RDF graphs and written down in Turtle syntax.

Table 34: RDB 2 RDF Mapping Language Useful Link

Useful Links	
Website	R2RML: RDB to RDF Mapping Language

4.3.4 Step 3: Generating RDF

When you have the mappings in place, then the next step is to generate RDF data from native data based on the mappings specified in the previous step.

The following tools can be used to transform your data to RDF:

4.3.4.1 RMLMapper

The RMLMapper executes RML rules to generate Linked Data. It is a Java library, which is available via the command line.

Table 35: RML Mapper Useful Link

Useful Links	
Website	RML Mapper

4.3.5 Step 4: Making data queryable

When the data is transformed to RDF successfully, the next step is to enable querying on the RDF data in order to make it easily accessible. In order to do this, you need to select a triple store and upload your data to it. The following triple stores can be used to make your data queryable.

Table 36: Triple Stores Useful Links

Useful Links	
Virtuoso	Virtuoso
Jena Fuseki	Jena Fuseki

4.3.6 Step 5: Data Transformation Example

This section will explain mapping example data to an ontology and then how the transformed RDF data would look like.

Below is an example database table, i.e. CUSTOMER_TABLE, which contains records of customers. To transform this table to RDF format, you need to create mappings from this table to your selected ontology.

Table 37: Example Customer Table

CUSTOMER_TABLE

CUSTOMER_ID	FIRST_NAME	LAST_NAME	DATE_OF_BIRTH
1	John	Smith	14-04-1985
2	James	Oliver	02-11-1974

Below is an example of mappings generated for transforming the above database table to RDF format.

4.3.6.1 MAPPINGS

Table 38: Data Mapping Example

Example Mapping

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix fibo: <https://spec.edmcouncil.org/fibo/ontology/FND/AgentsAndPeople/People/>.

<#CustomerMap>
   rr:logicalTable [ rr:tableName "CUSTOMER_TABLE" ];
   rr:subjectMap [
      rr:template "http://data.example.com/customer/{CUSTOMER_ID}";
      rr:class ex:Person;
   ];

   rr:predicateObjectMap [
      rr:predicate ex:hasFirstName;
      rr:objectMap [ rr:column "FIRST_NAME" ];
   ];

   rr:predicateObjectMap [
      rr:predicate ex:hasSurname;
      rr:objectMap [ rr:column "LAST_NAME" ];
   ];

   rr:predicateObjectMap [
      rr:predicate ex:hasDateOfBirth;
      rr:objectMap [ rr:column "DATE_OF_BIRTH" ];
   ].
```

The example RDF data generated by transforming the database table, i.e. “CUSTOMER_TABLE” using the above mappings is shown below.

4.3.6.2 RDF DATA

Table 39: Example RDF Data

Example RDF Data

```
@prefix it: <http://data.example.com/customer/> .
@prefix fibo: <https://spec.edmcouncil.org/fibo/ontology/FND/AgentsAndPeople/People/>.

it:1 a fibo:Person ;
   fibo:hasFirstName "John" ;
   fibo:hasSurname "Smith" ;
   fibo:hasDateOfBirth "14-04-1985" ;

it:2 a fibo:Person ;
   fibo:hasFirstName "James" ;
   fibo:hasSurname "Oliver" ;
   fibo:hasDateOfBirth "02-11-1974" ;
```

5 SeSA-ME Continuous Integration/Continuous Development

The following CI/CD documentation is considering you have an instance of the SeSA-ME components running in your machine and all the file and ports are accessible/configured as local host, to run SeSA-ME Engine use the below commands:

5.1 SeSA-ME Engine Development

5.1.1 Run SeSA-ME Engine

To run SeSA-ME Engine use the below command:

mvnw spring-boot:run

Then Navigate to the below URL to check SeSA-ME Engine.

http://localhost:8080/sesame-engine/swagger-ui.html

5.1.2 Build SeSA-ME Engine

To build SeSA-ME Engine run the below command:

mvnw package

5.2 Deployment using Docker

Step 1: Use the following command to build a local docker image of SeSA-ME Engine.

docker build -t sesame-engine-image-local .

Step 2: Use the following command to deploy the docker image of SeSA-ME Engine built in the previous step.

docker run --name sesame-container -d -p 8080:8080 sesame-engine-image-local

Step 3: Navigate to the below URL to check SeSA-ME Engine.

http://localhost:8080/sesame-engine/swagger-ui.html

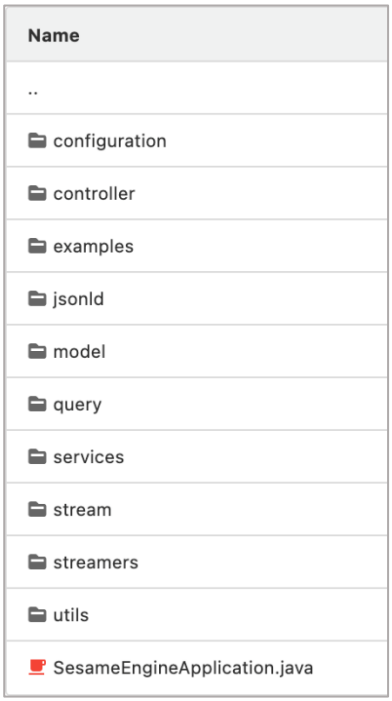
5.3 SeSA-ME Engine APIs

The description of SeSA-ME Engine APIs can be found in the section 4 in this deliverable

5.4 SeSA-ME Engine Deployment

5.4.1 SeSA-ME Engine Project Structure

The SeSA-ME component has the following structure:



5.4.2 Dependencies List

Table 40: SeSA-ME Dependecies

Example RDF Data

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
		xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
		<modelVersion>4.0.0</modelVersion>

		<groupId>eu.infinitech.nuig</groupId>
		<artifactId>sesame-engine</artifactId>
		<version>0.0.1-SNAPSHOT</version>
		<packaging>jar</packaging>

		<name>sesame-engine</name>
		<description>Sesame Engine</description>

		<parent>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-parent</artifactId>
			<version>1.5.4.RELEASE</version>
			<relativePath /> <!-- lookup parent from repository -->
		</parent>

		<properties>
			<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
			<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
```

```
		<java.version>1.8</java.version>
	</properties>

	<dependencies>
		<dependency>
			<groupId>io.springfox</groupId>
			<artifactId>springfox-swagger2</artifactId>
			<version>2.6.1</version>
		</dependency>

		<dependency>
			<groupId>io.springfox</groupId>
			<artifactId>springfox-swagger-ui</artifactId>
			<version>2.6.1</version>
		</dependency>

		<!-- uncomment below dependency if want to build a war -->
		<!-- <dependency>
		  <groupId>org.springframework.boot</groupId>
		  <artifactId>spring-boot-starter-tomcat</artifactId>
		  <scope>provided</scope>
		</dependency> -->

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-actuator</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-rest</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>

		<dependency>
			<groupId>eu.larkc.csparql</groupId>
			<artifactId>csparql-core</artifactId>
			<version>0.9.6</version>
		</dependency>

		<dependency>
		  <groupId>com.squareup.okhttp</groupId>
		  <artifactId>okhttp</artifactId>
		  <version>2.7.5</version>
		</dependency>

		<dependency>
		  <groupId>com.apicatalog</groupId>
		  <artifactId>titanium-json-ld</artifactId>
		  <version>1.0.0</version>
		</dependency>

		<dependency>
	    <groupId>org.glassfish</groupId>
	    <artifactId>jakarta.json</artifactId>
	    <version>2.0.0</version>
```

```
        </dependency>

    </dependencies>

    <repositories>
        <repository>
            <id>maven Repo1</id>
            <name>maven Repo1</name>
            <url>http://repo1.maven.org/maven2</url>
        </repository>
        <repository>
            <id>Sonatype repository</id>
            <name>Sonatype's Maven repository</name>
            <url>http://oss.sonatype.org/content/groups/public</url>
        </repository>
        <repository>
            <id>streamreasoning_repository</id>
            <name>streamreasoning repository</name>
            <url>http://streamreasoning.org/maven/</url>
            <layout>default</layout>
        </repository>

    </repositories>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>


</project>
```

5.4.3 Docker File

Table 41: Docker File

Example RDF Data
FROM openjdk:8-jdk-alpine RUN addgroup -S spring && adduser -S spring -G spring USER spring:spring ARG JAR_FILE=target/sesame-engine-0.0.1-SNAPSHOT.jar ADD ${JAR_FILE} app.jar ENTRYPOINT ["java","-jar","/app.jar"]

6 Conclusions

The Semantic Stream Analytics Engine (SeSA-ME) is an extension of the Super Stream Collider (SSC) tool, which provides a set of web-based interfaces and tools for building data mashups combining semantically annotated Linked Stream and Linked Data sources into easy-to-use data mashups for applications.

The SeSA-ME component/system includes static data and dynamic data (streams) processing tools along with a visual SPARQL query editor using Swagger APIs and visualization tools for novice users while supporting full access and control over the data mashups for expert users.

The development and deployment of the INFNITECH Graph Data Model which enables the support for both the design and deployment of stream-based web applications in a very simple and intuitive way and the analytics services using stream-based applications and services is tied with the development of the SeSA-ME platform is t.

The financial and insurance sector have not yet an adopted/accepted unified way of accessing & querying vast amounts of structured, unstructured, and semi-structured data. The INIFNITECH graph data model is made accessible online using machine readable files and also for human understanding and manipulation.

The main ontologies which are going to be used as baselines are FIBO, FIGI and LKIF, because they focused on both financial sector and financial operations containing the baseline for the metadata that represent, cross-domain and intra domain, financial transactions, and operations with an attached effort towards standardisation..

The INFINTECH Core ontology is an extension generated in the project that describes cross-domain vocabularies that are used in multi-domains within the INFINITECH project domain areas, it is meant to be complemented by other domain specific vocabularies. For this reason, and according to the initial requirements of the INFINITECH project, other vocabularies specifically related to security and payments are presented

INFINITECH graph data model facilitates a semantic overlay that is much easier to process and at the same time minimise the risk on processing data directly. This semantic layer approach constitutes also the first step of the INFINITECH pipeline, i.e., gathering semantically annotated data from provided and/or available datasets or data streams. In this deliverable, we have described how INFINITECH project would benefit from semantic technologies like Linked Data and ontologies as the best practices in the semantic interoperability building process.

Following semantic best practices, we have design and implemented the Semantic Stream Analytics Middleware-Engine (SeSA-ME) and analysed the already existing ontologies that are related to the finance and insurance sectors that can be reused for our purposes in the INFINITECH project.

The Prototype provided and documented is a reference implementation that was improved following general requirements coming from the study and purposes at INFINITECH pilot level following stakeholder’s requirements and also from particular domains where the use of a semantic engine for mashup building with semantic interoperability capabilities

7 References

[Boots 2017] Botts, M., Percivall, G., Reed, C. and Davidson. J. *OGC Sensor Web Enablement: Overview and High Level Architecture*. Technical report, OGC, December 2007.

[Compton et al 2012] Compton, M., Barnaghi, P., Bermudez, L., Castro, R. G., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., Huang, V., Janowicz, K. Kelsey, W. D., Phuoc, D. L., Lefort, L., Leggieri, M., Neuhaus, H., Nikolov, A., Page, K., Passant, A., Sheath, A. and Taylor, K. *The SSN Ontology of the Semantic Sensor Networks Incubator Group*. Journal of Web Semantics: Science, Services and Agents on the World Wide Web, ISSN 1570-8268, Elsevier, 2012.

[DOI] Document Object Identifier
Accessible here: http://www.doi.org/handbook_2000/DOIHandbook-v4-4.1.pdf

[EPCGlobal-RPS 2006] EPCglobal: Reader Protocol Standard, Version 1.1, 3 Ratified Standard, 4 June 21, 2006

[EPCGlobal-ALE 2009] EPCglobal: The Application Level Events (ALE) Specification, Version 1.1.1 Part I: Core Specification, EPCglobal Ratified Standard, 13 March 2009

[EPCGlobal-A 2007] EPCglobal: The EPCglobal Architecture Framework, EPCglobal Final Version 1.2 Approved 10 September 2007

[EPCGlobal-EPC 2007] EPCglobal: EPC Information Services (EPCIS) Version 1.0.1 Specification Approved September 21, 2007

[EPCGlobal-RMS 2007] EPCglobal: Reader Management Standard 1.0.1, 3 May 31, 2007

[GENE-Ontology] GENE Ontology - bioinformatics initiative
Accessible here: http://www.geneontology.org

[Heitman 2009] Heitmann, B., Kinsella, S., Hayes, C. and Decker, S. *Implementing Semantic Web Applications: Reference Architecture and Challenges*. In International Workshop on Semantic Web enabled Software Engineering, collocated with the 8th International Semantic Web Conference (ISWC2009), 2009.

[Henson 2009] Henson, C. A., Pschorr, J. K., Sheth, A. P. and Thirunarayan, K. *SemSOS: Semantic sensor observation service*. Collaborative Technologies and Systems, International Symposium on, 0:44–53, 2009.

[Jacobs 2004] Jacobs, I. and Walsh, N. *Architecture of the World Wide Web*, Volume One, World Wide Web Consortium, Recommendation REC-webarch-20041215, 2004.

[Le-Phuoc et al. 2011a] Le-Phuoc, D., Dao-Tran, M. Parreira, J. X. and Hauswirth, M. *A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data*. Proceedings of the 10th International Conference on The Semantic Web (ISWC'11), Springer, 2011

[Le-Phuoc et al. 2011b] Le-Phuoc, D., Nguyen Mau, H., Parreira, J. X. and Hauswirth, M.. *The Linked Sensor Middleware – Connecting the Real World and the Semantic Web*. Proceedings of the 10th International Conference on The Semantic Web (ISWC'11), Springer, 2011

[Le-Phuoc et al. 2009] Le-Phuoc, D. and Hauswirth, M. *Linked open data in sensor data mashups*. Proceedings of the 2nd International Workshop on Semantic Sensor Networks (SSN09) in conjunction with ISWC 2009

[LOD-Project] World Wide Web Consortium - Linked Open Data, Accessible here: http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

[Priest 2007] Priest, A. Na, M., Niedzwiadek, H. and Davidson, J. Sensor observation service. Technical Report OGC 06-009r6, October 2007.

[Ruta 2007] Ruta, M. , Noia, T. Di, Scioscia, F., Di Sciascio E. Semantic-enhanced EPCglobal Radio-Frequency IDentification. SWAP 2007

[Salehi 2007] Salehi, A., Aberer, K. «GSN, Quick and Simple Sensor Network Deployment», European conference on Wireless Sensor Networks (EWSN), Netherlands, 2007

[Scherp 2009] Scherp, A., Franz, T. Saatho, S. Staab. *F–a Model of Events Based on the Foundational Ontology DOLCE+DnS Ultralight*. In: International Conference on Knowledge Capturing (K-CAP), Redondo Beach, CA, USA., 2009.

[Sheth 2008] Sheth, A. Henson, C., Sahoo. S. *Semantic Sensor Web*. IEEE Internet Computing 12 (4), 2008.

[Tsiatsis 2010] Tsiatsis, V., Gluhak, A., Bauge, T., Montagut, F., Bernat, J., Bauer, M., Villalonga, C., Barnaghi, P.M., Krco, S. The SENSEI Real World Internet Architecture. Future Internet Assembly, IOS Press, 2010.

[UMLS] Unified Medical Language System
Accessible here: http://www.nlm.nih.gov/ research/umls/index.html

[IETF-RFC2141] IETF - Uniform Resource Names
Accessible here: http://tools.ietf.org/html/rfc2141

[W3C-RDF] World Wide Web Consortium - Resource Description Framework,
Accessible here: http://www.w3.org/TR/rdf-syntax-grammar/

[W3C-RDFSchema] World Wide Web Consortium - Resource Description Framework Schema
Accessible here: http:// www.w3.org/ TR/ rdf-schema

[W3C-Turtle] World Wide Web Consortium - Turtle Serialisation Specification
Accessible here: http://www.w3.org/TeamSubmission/turtle/

[W3C-N-Triples] World Wide Web Consortium - N-Triples format specification
Accessible here: http://www.w3.org/TR/rdf-testcases/#ntriples

W3C-OWL] World Wide Web Consortium – Ontology Web language
Accessible here: http:// www.w3.org/ TR/ owl-ref

[W3C RDFa] World Wide Web Consortium - Resource Description Framework in Attributes
Accessible here: http://www.w3.org/TR/xhtml-rdfa-primer/

[W3C-SPARQL] SPARQL Query Language for RDF Implementation
Accessible here: http://www.w3.org/TR/rdf-sparql-query/