

SemSorGrid4Env

FP7-223913



Deliverable

D4.1

Design of the SemSorGrid4Env ontology-based data integration model

Jean-Paul Calbimonte and Oscar Corcho

Universidad Politécnica de Madrid

Alasdair J G Gray

University of Manchester

August 28, 2009

Status: Final Version

Scheduled Delivery Date: August 31, 2009

Executive Summary

Semantic technologies and ontologies have been successfully used in the latest years for data integration, the process of providing *unified* and *transparent* data access to multiple and heterogeneous data sources [Len02]. The problem of integrating these sources involves not only granting access to information but also providing mechanisms for interpreting and processing the data consistently, overcoming syntactical and semantic heterogeneity.

The main of these data sources in the context of the *Semantic Sensor Grid Rapid Application Development for Environmental Management* project (SemSorGrid4Env) are stream data sources coming from *sensor networks*. Sensor networks consist of multiple interconnected nodes that are able to sense and capture different kinds of data, accessed through query processors that use declarative continuous query languages. Managing streaming data differs significantly from classical static data. Streaming data is potentially infinite and transient, with tuples being constantly added. Therefore queries on this data are also continuous, i.e. their results are updated regularly as time passes [TGN+92].

The goal of the SemSorGrid4Env project is to provide a platform that supports the rapid development of applications that make use of existing heterogeneous data sources using semantic web technologies [GGF+09]. In this deliverable we propose the design of a *Semantic Integrator Service* that deals with the above-mentioned challenges, using ontology-based data integration and mapping techniques, for both streaming and stored data.

In order to produce this design we have reviewed work on ontology-based data access, ontology-based data integration, streaming data access, distributed query processing and quality of service notions. We have also reviewed relevant technologies and approaches in all these areas, which we will reuse in our work in this WP of the SemSorGrid4Env project.

We have also proposed a walk-through of a typical use-case that shows how the user requirements expressed in the SemSorGrid4Env use-cases can be handled from the semantic integration point of view. We do not only show how requirements are fulfilled but also how we can implement the service functionalities outlined in the architecture of SemSorGrid4Env [GGF+09]. We have designed and described the components that we will develop and provide, outlining the main extensions to existing work that we have to accomplish. Finally we propose the main lines of our future work in this WP, which will be revised throughout the duration of the project and will be the focus of the following deliverables: R₂O extensions for streams, extended SPARQL queries for streams, Translation of SPARQL for streams to streaming languages, Distributed Query Processing of integrated streaming data queries and Quality-of-Service optimizations.



Note on Sources and Original Contributions

The SemSorGrid4Env consortium is an inter-disciplinary team, and in order to make deliverables self-contained and comprehensible to all partners, some deliverables thus necessarily include state-of-the-art surveys and associated critical assessment. Where there is no advantage to recreating such materials from first principles, partners follow standard scientific practice and occasionally make use of their own pre-existing intellectual property in such sections. In the interests of transparency, we here identify the main sources of such pre-existing materials in this deliverable:

- Chapter 2 contains material adapted from [GGF+09]
- Chapter 3 contains material adapted from [GGF+09, GGF+09a]
- Sections 4.2.1 and 4.2.2 contain material adapted from [Bar06]



Document Information

Contract Number	FP7-223913		Acronym	SemSorGrid4Env	
Full title	SemSorGrid4Env: Semantic Sensor Grids for Rapid Application Development for Environmental Management				
Project URL	www.semsorgrid4env.eu				
Document URL	http://www.semsorgrid4env.eu/home.jsp?content=/sew/viewTerm&content=instance.jsp&sew_var_name=instance&sew_instance=D4.1&sew_instance_set=SemSorGrid4Env&origin=%2Fhome.jsp				
EU Project officer	Gaëlle Le Gars				
Deliverable	Number	D4.1	Name	Design of the SemSorGrid4Env ontology-based data integration model	
Task	Number	T4.1	Name	Design an ontology-based integration model	
Work package	Number	WP4			
Date of delivery	Contractual	31 August 2009	Actual	31 August 2009	
Code name	D4.1		Status	draft <input type="checkbox"/>	final <input checked="" type="checkbox"/>
Nature	Prototype <input type="checkbox"/> Report <input checked="" type="checkbox"/> Specification <input type="checkbox"/> Tool <input type="checkbox"/> Other <input type="checkbox"/>				
Distribution Type	Public <input checked="" type="checkbox"/> Restricted <input type="checkbox"/> Consortium <input type="checkbox"/>				
Authoring Partner	UPM				
QA Partner	UNIMAN				
Contact Person	Jean-Paul Calbimonte				
	Email	jpcalbimonte@delicias.dia.fi.upm.es	Phone	+34913363670	Fax +34913524819
Abstract (for dissemination)	<p>Semantic technologies and ontologies have been successfully used in the latest years for data integration, the process of providing <i>unified</i> and <i>transparent</i> data access to multiple and heterogeneous data sources. The problem of integrating these sources involves not only granting access to information but also providing mechanisms for interpreting and processing the data consistently, overcoming syntactical and semantic heterogeneity.</p> <p>The main of these data sources in the context of the <i>Semantic Sensor Grid Rapid Application Development for Environmental Management</i> project (SemSorGrid4Env) are stream data sources coming from <i>sensor networks</i>. Sensor networks consist of multiple interconnected nodes that are able to sense and capture different kinds of data, accessed through query processors that use declarative continuous query languages. Managing streaming data differs significantly from classical static data. Streaming data is potentially infinite and transient, with tuples being constantly added. Therefore queries on this data are also continuous, i.e. their results are updated regularly as time passes.</p> <p>The goal of the SemSorGrid4Env project is to provide a platform that supports the rapid development of applications that make use of existing heterogeneous data sources using semantic web technologies. In this deliverable we propose the design of a <i>Semantic Integrator Service</i> that deals with the above-mentioned challenges, using ontology-based data integration and mapping techniques, for both streaming and stored data.</p> <p>In order to produce this design we have reviewed work on ontology-based data access, ontology-based data integration, streaming data access, distributed query processing and quality of service notions. We have also reviewed relevant technologies and approaches in all these areas, which we will reuse in our work in this WP of the SemSorGrid4Env project.</p> <p>We have also proposed a walk-through of a typical use-case that shows how the user requirements expressed in the SemSorGrid4Env use-cases can be handled from the semantic integration point of view. We do not only show how requirements are fulfilled but also how we can implement the service functionalities outlined in the architecture of SemSorGrid4Env. We have designed and described the components that we will develop and provide, outlining the main extensions to existing work that we have to accomplish. Finally we propose the main lines of our future work in this WP, which will be revised throughout the duration of the project and will be the focus of the following deliverables.</p>				
Keywords	Data Integration. Semantic integration				
Version log/Date	Change		Author		
0.1 / 22 May 2009	Initial Draft		J-P. Calbimonte		
0.15 / 30 July 2009	Table of contents, restructured		J-P. Calbimonte & O. Corcho		
0.2 / 15 August 2009	Revised proposal, major edits		J-P. Calbimonte & O. Corcho		
0.3 / 22 August 2009	Comments & corrections		J-P. Calbimonte & O. Corcho		
0.4 / 25 August 2009	Corrections from QA		J-P. Calbimonte & O. Corcho & A. Gray		
0.5 / 28 August 2009	Final revision		J-P. Calbimonte & O. Corcho & A. Gray		

Project Information

This document is part of a research project funded by the IST Programme of the Commission of the European Communities as project number FP7-223913. The Beneficiaries in this project are:








Partner	Acronym	Contact
Universidad Politécnica de Madrid (Coordinator)	UPM 	Prof. Dr. Asunción Gómez-Pérez Facultad de Informática Departamento de Inteligencia Artificial Campus de Montegancedo, sn Boadilla del Monte 28660 Spain #@ asun@fi.upm.es #t +34-91 336-7439, #f +34-91 352-4819
The University of Manchester	UNIMAN  The University of Manchester	Prof. Carole Goble Department of Computer Science The University of Manchester Oxford Road Manchester, M13 9PL, United Kingdom #@ carole@cs.man.ac.uk #t +44-161-275 61 95, #f +44-161-275 62 04
National and Kapodistrian University of Athens	NKUA  National and Kapodistrian University of Athens	Prof. Manolis Koubarakis University Campus, Ilissia Athina GR-15784 Greece #@ koubarak@di.uoa.gr #t +30 210 7275213, #f +30 210 7275214
University of Southampton	SOTON 	Prof. David De Roure University Road Southampton SO17 1BJ United Kingdom #@ dder@ecs.soton.ac.uk #t +44 23 80592418, #f +44 23 80595499
Deimos Space, S.L.	DMS 	Mr. Agustín Izquierdo Ronda de Poniente 19, Edif. Fiteni VI, P 2, 2º Tres Cantos, Madrid – 28760 Spain #@ agustin.izquierdo@deimos-space.com #t +34-91-8063450, #f +34-91-806-34-51
EMU Limited	EMU 	Dr. Bruce Tomlinson Mill Court, The Sawmills, Durley number 1 Southampton, SO32 2EJ – United Kingdom #@ bruce.tomlinson@emulimited.com #t +44 1489 860050, #f +44 1489 860051
TechIdeas Asesores Tecnológicos, S.L.	TI 	Mr. Jesús E. Gabaldón C/ Marie Curie 8-14 08042 Barcelona, Spain #@ jesus.gabaldon@techideas.es #t +34.93.291.77.27, #f ++34.93.291.76.00



Table of Contents

1	Introduction	1
1.1	Glossary.....	2
2	Architecture	4
2.1	Overview	4
2.2	Semantic Integration and Query Service.....	5
2.2.1	Integration Interface.....	7
2.2.2	Query Interface	8
3	Requirements	11
4	Background.....	13
4.1	Fundamentals	13
4.1.1	Ontology-based Data Access	13
4.1.2	Ontology-based Integration	15
4.1.3	Streaming Data Access	18
4.1.4	Distributed Query Processing	22
4.1.5	Quality of Service	23
4.2	Related Technologies	24
4.2.1	R ₂ O.....	24
4.2.2	ODEMapster	26
4.2.3	SNEEqI	29
4.2.4	SPARQL for RDF Streams.....	30
4.2.5	OGSA-DQP	32



5	Semantic Integrator Proposal.....	34
5.1	Integration of Data Sources: Walk-through	34
5.1.1	Mapping to the Ontology	34
5.1.2	Querying over the Ontology	37
5.1.3	Query Execution	38
5.2	Service Implementation.....	40
5.2.1	Integration Interface Implementation	40
5.2.2	Query Interface Implementation	41
5.3	Future Work	42
5.3.1	R ₂ O Extensions for Streams.....	43
5.3.2	SPARQL Support.....	43
5.3.3	Support for R ₂ O Extensions.....	43
5.3.4	Extended SPARQL for Streams.....	44
5.3.5	Query Translation	44
5.3.6	Distributed Query Processing	45
5.3.7	QoS Optimisations	45
6	Bibliography	46
7	Annexes	55
	Linked Stream Data: A Position Paper	55



Index of Figures

Figure 2.1: SemSorGrid4Env Architecture.....	4
Figure 2.2: Interfaces exposed by the Semantic Integration and Query Service.....	6
Figure 4.1: Architecture of a distributed stream processing engine [ALM+04].....	21
Figure 4.2: Phases of query processing [Kos00].	22
Figure 4.3: ODEMapster Architecture [Bar06].	26
Figure 4.4: ODEMapster virtualised execution mode [Bar06].	27
Figure 4.5: ODEMapster materialised execution mode [Bar06].	27
Figure 4.6: ODEMapster execution steps.	28
Figure 4.7: OGSA-DQP Architecture [LMH+09].	33
Figure 5.1: Sample Fire Ontology.	36
Figure 5.2: Sample mapping from stream-to-ontology.....	36
Figure 5.3: Semantic Integrator modules.....	38



Index of Tables

Table 2.1: Types used in the Integration and Query Interfaces.	6
Table 2.2: Faults used in the Integration and Query Interfaces.	7
Table 2.3: The IntegrateAs operation.	7
Table 2.4: The GetDataResourcePropertyDocument operation.	8
Table 2.5: The DestroyDataResource operation.	8
Table 2.6: The <GenericQuery> operation.	9
Table 2.7: The <GenericQueryFactory> operation.	10



1 Introduction

The goal of the *Semantic Sensor Grid Rapid Application Development for Environmental Management* project (SemSorGrid4Env) is to provide a platform that supports the rapid development of applications which make use of existing heterogeneous data sources including databases and sensor networks, using semantic web technologies [GGF+09].

The SemSorGrid4Env project activities are divided in separate work packages, and this deliverable is part of Work Package 4 (WP4). The objective of WP4 is to design and implement a *Semantic Integration Service* for the SemSorGrid4Env project and specify a suite of sensor network ontologies that will be used for describing sensors and related data.

This deliverable focuses on the issues and challenges of data integration for heterogeneous sources; and the design of the Semantic Integrator that will be in charge of combining these diverse sources.

Latest advances in wireless communications and sensor technologies have opened the way for deploying networks of interconnected sensing devices capable of ubiquitous data capture, collection and processing. Sensor networks deployments are expected to increase significantly in the upcoming years because of their advantages and unique features. Tiny sensors can be installed virtually anywhere and still be reachable thanks to wireless communications. Moreover, these devices are inexpensive and can be used for a wide range of applications such as security surveillance, traffic control, environmental monitoring, healthcare provision, industrial monitoring, etc.

However, sensor networks characteristics also raise several challenges for the research community, related to inherent constraints such as the low power resources of the devices, their restricted computation capabilities and limited storage. Another key challenge is related to the integration of the data collected by sensor networks. Applications such as the mashups in the SemSorGrid4Env project require a suitable platform to access and combine heterogeneous data coming from different sensor networks and other data sources such as databases. Furthermore, these applications require accessing this data in terms of a uniform schema that hides the diverse and internal data representations of each source.

Semantic technologies and ontologies have been successfully used in the latest years for data integration solutions, particularly in the area of databases. The integration solution that we propose in this deliverable will be built using ontology-based data access and mapping to data sources, which may be streaming sensor networks or relational databases.

The main contributions of this deliverable are:



- A study of the related work in the area of semantic data integration and ontology based access to relational data.
- A study of semantic querying over streaming data and identification of limitations on current formalisms regarding useful streaming query operators such as time windows, etc.
- A proposal of a high level ontology-based integration model for heterogeneous streaming and relational data sources.

The document is organised as follows:

In Chapter 2, we set the context of this work package, by introducing the SemSorGrid4Env architecture and its service tiers and interfaces. We focus in particular on the Semantic Integrator Service, which is the core component for the integration solution proposed in WP4.

In Chapter 3, we identify the requirements of WP4 based on the use-cases of the SemSorGrid4Env project.

In Chapter 4, we provide details about the previous works on the areas of ontology-based data access, ontology-based data integration, stream data access and integration, distributed query processing and quality of service concepts. Then we provide details about the technologies that will be used as basis for the remainder of the work in WP4.

In Chapter 5, we introduce our proposal for the design and internal details of the ontology-based data integration solution for the SemSorGrid4Env project.

An Annex (see Chapter 7) is included in this deliverable about linked stream data. This annex deals with the requirements and a proposal for generation of unique URIs for sensor network data, which may be useful for identifications creation during translation from stream to ontological models.

1.1 Glossary

API Application Programmer Interface. A set of defined method calls.

CQL Continuous Query Language

C-SPARQL Continuous SPARQL

DBMS Database Management System.

DQP Distributed Query Processor.

DSMS Data Stream Management System.

GAV Global-as-View. A mechanism for mapping a global schema to the data source schemas.



GLAV Global-Local-as-View. A mechanism for mapping a global schema to the source schemas.

HTTP Hypertext Transfer Protocol.

LAV Local-as-View. A mechanism for mapping a global schema to the data source schemas.

OGSA-DAI Open Grid Services Architecture Database Access and Integration Services.

OGSA-DQP Open Grid Services Architecture Distributed Query Processing

OWL The Web Ontology Language. A W3C standard for specifying ontologies.

QEP Query Execution Plan.

QoS Quality of Service.

R₂O Relational-to-Ontology mapping language.

RDBMS Relational Database Management System.

RDF The Resource Description Framework. A W3C standard for modeling Web resources metadata.

REST Representational State Transfer.

SNEE Sensor Network Engine. A query compiler for the SNEEqL language.

SNEEqL Sensor Network Evaluation Query Language for querying streams of data.

SPARQL SPARQL Protocol and RDF Query Language for querying RDF documents.

SQL Structured Query Language for querying relational databases.

STREAM The Stanford data stream management system.

UNIMAN University of Manchester.

UPM Universidad Politécnica de Madrid.

URI Uniform Resource Identifier.

WP Work Package.

WS-* The group of Web service standards.

XML Extensible Markup Language

2 Architecture

In this chapter we present an overview of the SemSorGrid4Env architecture and its tiers (see Section 2.1). Then we focus on the Semantic Integrator Service, the component charge with the task of integrating heterogeneous data sources using semantic technology (see Section 2.2). Then we provide details about the interfaces exposed by this service.

2.1 Overview

The SemSorGrid4Env architecture, as defined in [GGF+09] comprises of a set of web services that interact by calling each other, as depicted in Figure 2.1. The services can be categorised into three well-defined tiers: *application tier*, *middleware tier* and *data tier*.

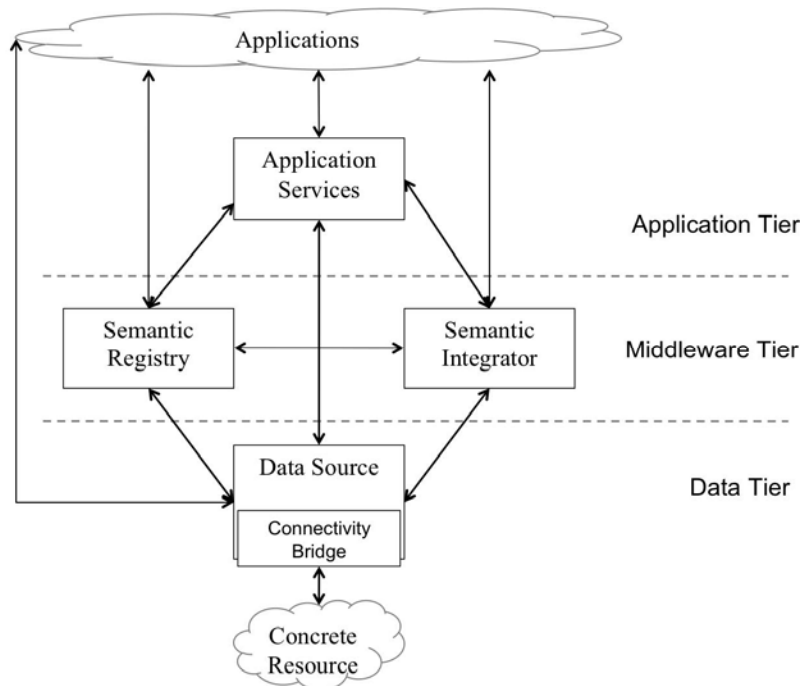


Figure 2.1: SemSorGrid4Env Architecture.

Data Tier: Services of this tier are able to publish and provide access to classical database data or streaming data. In Figure 2.1 a single Data Source service is depicted. However for the SemSorGrid4Env project two different types of data services have been defined: *Stored Data Service* and *Streaming Data Service*, for databases and streams respectively [GGF+09a].

Middleware Tier: Two main services are available in this Tier: *Semantic Registry* and *Semantic Integrator Service*. The Semantic Registry [KKK09] allows discovering available data sources and services according to some criteria. The Semantic Integrator provides a mechanism for accessing multiple data sources as a single *virtual source*. The virtual data source will present a global ontological view and queries over this view will be translated to queries over the schemas of the original data sources.

Application Tier: This tier provides domain specific functionality to consumers so that mash-up applications can directly use them instead of dealing with the complexities of the middleware and data tiers.

WP4 focuses on one of the key classes of services of the middleware tier: the Semantic Integration and Query Service or Semantic Integrator Service.

2.2 Semantic Integration and Query Service

The Semantic Integration and Query Service or Semantic Integrator provides access to multiple data sources through a single mediated ontological view, as if it was a virtual data source. All SemSorGrid4Env services define a set of interfaces that will be available and will be exposed to consumers. For each interface there is a set of operations, the types they use for inputs and outputs and a set of faults that may be generated in case of errors.

The Semantic Integration and Query Service requires the following functionality from other services [GGF+09]:

- Retrieving data sources functional properties and metadata as defined by the Service Interface.
- Retrieving data sources data description and querying the data using the Query Interface.
- Receiving data streams using a pull-based mechanism using the Data Access Interface.
- Receiving data streams using a push-based mechanism using the Subscription Interface.
- Security mechanisms for data access authorization.

As detailed in the architecture document [GGF+09] and depicted in Figure 2.2, the Semantic Integration and Query Service exposes the following interfaces:

- *Service Interface*
- *Integration Interface*
- *Query Interface*
- *Data Access Interface* (optional)
- *Subscription Interface* (optional)
- *Subscription Manager Interface* (optional)
- *Notification Interface* (optional)

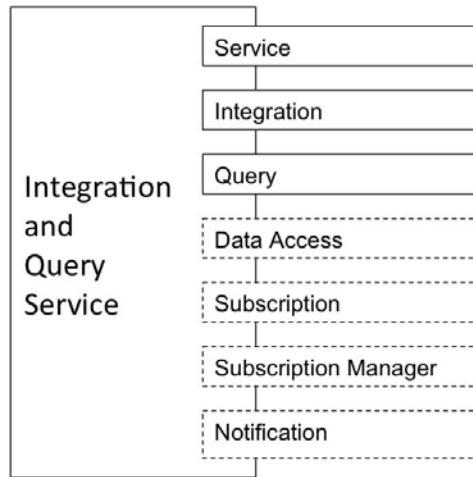


Figure 2.2: Interfaces exposed by the Semantic Integration and Query Service.

If the Integration Interface supports pull-based retrieval of streams, it must expose the Data Access Interface. If it supports push-based delivery of streams then it must expose the Subscription and the Subscription Manager interfaces. If the Integration Interface requires receiving a push-based delivery stream from a data source it must expose the Notification Interface.

The Integration and Query interfaces are detailed in the next sections. For each interface, a set of exposed operations is listed. Each operation has a set of input and output parameters, and a set of possible Faults. Notice the notation conventions: elements marked with ‘+’ may appear one or multiple times, elements marked with ‘?’ may appear once or may not appear.

The types listed in Table 2.1 and faults in Table 2.2 are valid for the subsequent sub-sections.

EndpointReference	An address conforming to WS-Addressing [GHR06] that specifies the location of a resource as accessed through a Web service.
QName	A String that denotes an XML global element definition.
DataResourceAbstractName	The abstract name associated with the data resource(s) represented as a URI.
MappingDocument	An XML document that describes the mappings between the data sources and the resulting global schema.
PropertyDocument	An XML document conforming to a defined XML schema to describe the properties of the service.
ConfigurationDocument	An XML document containing the initial parameters for the indirect access resource.
DatasetFormatURI	The URI of a dataset format.
RequestDocument	An XML document that contains the request expression.
ResponseDocument	A document consisting of a dataset format URI and the encoded data.
QueryExpressionDocument	An XML document that contains the query expression.
DataResourceAddressList	A list of data resource addresses.

Table 2.1: Types used in the Integration and Query Interfaces.

InvalidDataResourceNameFault	The data resource specified is unknown to the service.
DataResourceUnavailableFault	The data resource that is the target of the message is currently not available.
InvalidMappingDocumentFault	The mapping document specified is not in a valid format or cannot be processed by the service.
NotAuthorizedFault	The consumer is not authorized to perform the requested operation or is not authorized to perform the requested operation at this time.
ServiceBusyFault	The service is already processing a message and concurrent operations are not supported.
InvalidDatasetFormatFault	The dataset format URI specified is not known to the service.
InvalidExpressionFault	The expression given as part of the request contains errors.
InvalidLanguageFault	The input dataset has an unrecognized language element.
InvalidConfigurationDocumentFault	The configuration document specified is not valid.

Table 2.2: Faults used in the Integration and Query Interfaces.

2.2.1 Integration Interface

The Integration Interface provides integrated data access to multiple data resources through a unified virtual source that can be later managed and queried.

Operations

IntegrateAs: Creates a data resource which presents the global view over a set of data sources (see Table 2.3).

Inputs	resourceNames+: DataResourceAbstractName mappings?: MappingDocument
Output	integratedDataResource: EndpointReference
Faults	InvalidDataResourceNameFault DataResourceUnavailableFault InvalidMappingDocumentFault NotAuthorizedFault ServiceBusyFault

Table 2.3: The IntegrateAs operation.

This operation receives the names of the data resources and the mapping as an input. The data resource names are of type `DataResourceAbstractName`, represented by an URI. The mapping document will indicate how to transform the elements from the stored or streaming data to elements of the global schema. The output is a reference to the created *virtual data source*, and can later be used in order to query the information provided by the underlying original sources.

The implementation of this interface operation will need to use a mapping language expressible enough to represent complex mappings from the heterogeneous models to the global one, as described in Section 4.1.1.

The data source referenced by the result of this operation (*integratedDataResource*) will have to be able to be queried in terms of the global view. And the queries may include data that has been extracted from any of the local sources. Therefore the global view will need to cover the data that can be expressed in the local data source schemas.

2.2.2 Query Interface

The Query Interface defines operations for data access using a formal query language to specify the needed information. Two modes for retrieving the data are available in the Query Interface: *direct* and *indirect*. In the direct mode the results are thrown in the response message of the query request. In the indirect mode the response message includes a reference to a data resource that can be queried using either the Data Access Interface or the Subscription and Subscription Manager Interfaces.

Operations

GetDataResourcePropertyDocument: Returns the core property document values associated with the service implementing this message (see Table 2.4).

Inputs	resourceName: DataResourceAbstractName
Output	properties: PropertyDocument
Faults	InvalidDataResourceNameFault DataResourceUnavailableFault NotAuthorizedFault ServiceBusyFault

Table 2.4: The GetDataResourcePropertyDocument operation.

This operation returns a document containing metadata of the data source identified by the specified *DataResourceAbstractName*. It is expected that the resulting *PropertyDocument* will be an XML document complying with a defined schema.

DestroyDataResource: Destroys the named data resource, future messages directed at the resource must yield an *InvalidResourceNameFault* (see Table 2.5).

Inputs	resourceName: DataResourceAbstractName
Output	
Faults	InvalidDataResourceNameFault DataResourceUnavailableFault NotAuthorizedFault ServiceBusyFault

Table 2.5: The DestroyDataResource operation.

This is a data source management operation that will typically be used when the data source is no longer in use.

<GenericQuery>: Directs a query document to a data resource (see Table 2.6).

Inputs	resourceName: DataResourceAbstractName format?: DatasetFormatURI queryExpression: RequestDocument
Output	queryResponse: ResponseDocument
Faults	InvalidDataResourceNameFault DataResourceUnavailableFault InvalidDatasetFormatFault InvalidExpressionFault InvalidLanguageFault NotAuthorizedFault ServiceBusyFault

Table 2.6: The <GenericQuery> operation.

This is the central operation of the Query Interface, as it allows retrieving results given a declarative query expression. The resourceName parameter will indicate which data source is to be queried, for instance a stored or streaming data source, or an integrated data source as the one thrown by the IntegrateAs operation of the Integration Interface. The query itself is contained in the queryExpression input parameter of type RequestDocument.

The query expression must be written in a declarative query language expressive enough to represent the different kinds of information requests typical of stored or streaming data access. Therefore these queries must provide support for joins between relations and streams, aggregates, selections and projections, stream-to-relation operators, relation-to stream operators, etc, as it will be seen in Section 4.1.3. In the case of an integrated data source, it is expected that the queries are specified in terms of the global ontological view, rather than the local schemas. Therefore a query language capable of querying ontological elements is required for such situations.

In this case the query results are directly outputted in the queryResponse parameter. The format of this response will depend on the optional format parameter of type DatasetFormatURI. It is also noteworthy that in case of querying an integrated data source in terms of a global ontological view, the results are also expected to be in terms of that view.

<GenericQueryFactory>: Creates a relationship between a data resource representing the result of a query and the data access service by which it will be accessed (see Table 2.7).

Inputs	resourceName: DataResourceAbstractName responseInterfaceType?: QName configurationDocument?: ConfigurationDocument preferredTargetService?: EndPointReference requestDocument: QueryExpressionDocument
Output	resourceList: DataResourceAddressList<EndPointReference>

Faults	InvalidDataResourceNameFault DataResourceUnavailableFault InvalidPortTypeQNameFault InvalidCon gurationDocumentFault InvalidExpressionFault InvalidLanguageFault NotAuthorizedFault ServiceBusyFault
--------	---

Table 2.7: The <GenericQueryFactory> operation.

In this operation the results are not directly thrown in the output message. Instead a list of data sources is provided as output, and these sources can be consulted to get the response data sets either with a pull-based mechanism using the Data Access Interface; or push-based using the Subscription and Subscription Manager interfaces.

3 Requirements

The SensorGrid4Env project considers two concrete use-case scenarios in order to validate the proposed platform. The first is a fire risk monitoring and warning system in the Region of Castilla y León in Spain, and is under the scope of WP6 [LDI09]. The second is a coastal and estuarine flood warning system in the Southern region of the United Kingdom, and is the subject of WP7 [CHSR09].

This chapter summarises the requirements for WP4 gathered from the use-cases presented in the requirements specifications of WP6 and WP7. These requirements have also been analysed in [GGF+09] and [GGF+09a] and we include them here for completeness and consistency with the work done in WP2.

- *Stored data sources* must be accessible, mainly for querying and analysing archived information such as historic satellite or sensor network data.
- *Stream data sources* must be accessible, mainly for real-time queries and analysis such as continuous queries over a sensor network.
- *Integrating different stored data sources* through a unified view must be possible, for instance for retrieving and combining historic satellite and sensor network data.
- *Integrating different streaming data sources* through a unified view must be possible, for instance for combining data from different sensor networks.
- *Integrating both stored and streaming data* through a unified view must be possible, for instance in order to retrieve and combine current and past sensor network data.
- *Posing declarative queries* over the integrated view must be possible, independently of the stored or streaming data sources.
- *Subscribing to events* in the integrated data source must be possible.
- Results of the queries can be delivered either *directly* in the query response or *indirectly* through another data source.
- Streaming results can be accessed by *explicit requests* of the consumer or *without any request*. In the latter case the streams are generated and sent to consumers at a potentially unknown rate.
- The queries posed to the integrated data source must consider *temporal interval* filters and *temporal correlation* over streaming data.



- The queries posed to the integrated data source must consider *spatial correlation*.
- The integrated data source must consider *Quality-of-Service* parameters for optimizations in the query processing task.

In summary the Integration and Query Service is required to be able to combine different stream or stored data sources under a unified view. This integrated data source can later be queried as if it was a single data source.

4 Background

In this chapter we present a survey about previous work related to semantic data access and integration, data streams management and querying, distributed query processing and Quality-of-Service notions. This chapter is divided in two sections: Fundamentals (see Section 4.1), which describes the state of the art in the selected relevant topics, and Related Technologies (see Section 4.2), which describes the main technological products and assets that will be used for this WP in the context of the SemSorGrid4Env project.

4.1 Fundamentals

The following sections describe the state of the art in ontology-based data access and integration, streaming data access and integration, distributed query processing, and also quality of service representation for data services.

First in Section 4.1.1 we provide a short overview of the research and available solutions for *ontology-based data access*. Then we present the challenges of *data integration* (see Section 4.1.2) and the relevant works on the subject that consider the problem of semantic interoperability. In this section we focus primarily on ontology-based integration approaches and mapping from relational sources to ontological models.

Next in Section 4.1.3 we introduce the concepts of *streams* and how they differ from classic relational data. We include references to systems and approaches for managing and querying streaming data, and also for combining it with static relational data. Afterwards we identify the current limitations on semantic technologies to deal with stream information, particularly concerning time notions. Then we review other works dealing with semantic queries over streams and the specific extensions to currently available query languages.

Afterwards Section 4.1.4 is devoted to *Distributed Query Processing*, as it is expected that data integration of heterogeneous sources will require techniques from this area.

Finally in Section 4.1.5 we introduce some concepts about *Quality of Service*, relevant in the context of query processing and data quality evaluation.

4.1.1 Ontology-based Data Access

The realisation of the *Semantic Web* vision, where data is available, understandable and processable by computers, has launched several initiatives that aim at providing semantic access to traditional data sources. Most stored data is currently preserved in relational databases and it

has therefore become a need to generate Semantic Web content from them [SHH+08]. In this context there is a considerably large amount of research in the community, with the goal of exposing data in terms of ontologies that formally express the domain of interest [PLC+08]. This is the goal of *Ontology-based data access* (OBDA).

Most of the approaches attempt to provide some kind of *mapping* from a relational concept to a concept in an ontological model (such as OWL [BvHH+04] or RDF [MM04]). Many problems arise then, as they have to deal with model mismatch issues, query interpretation, semantic reasoning, etc.

Some of the proposed systems use their own languages [BCG05, BC07] to define the mappings between ontology concepts and roles and their corresponding tables in the relational models. Others rely on SPARQL [PS08] extensions and SQL [SQL92] expressions to define these mappings [PLC+08, SSW07, EM07, PC06].

OBDA systems use these kinds of mappings between the global ontology and the different data sources. These mappings allow constructing queries over the ontology (e.g. in SPARQL) which are then re-written to the data sources query language (typically SQL). Then the results are converted to RDF which is the result that is returned to the user [SHH+08]. We briefly describe the main works on this area that surfaced in the latest years.

One of the common approaches is to first generate a *syntactical translation* of the database schema to an ontological representation. Although the resulting ontology has no real semantics behind, it may be argued that it is a first step through an ontology model and that ontology alignment could be used later to map it to a real domain ontology [LT09, MBR01].

In SquirrelRDF [SSW07] they take this simple approach. A rough mapping of the relational database schema is built in RDF. There is no mapping to a mediated ontology. SPARQL queries can be executed and results are return in RDF.

An additional step is taken in RDBToOnto [Cer08], where the ontology generation process does not only take the database schema into account, but also the data. For instance it is able to discover subsumption relationships by finding categorisation columns in the database tables. Even after this resulting ontology is produced, RDBToOnto allows users to create custom constraints. However this work focuses on the ontology generation and does not provide a querying mechanism to the database data.

Relational.OWL [PC06] builds an ontology based on the relational schema and then maps it to the mediated ontology through a RDF query language. The first phase -transforming the database schema to the Relational.OWL ontology- produces a syntactical representation without real semantics. It is thus necessary to proceed with the second step, mapping the Relational.OWL representation to the domain ontology. This mapping can be done with an RDF query language like SPARQL and it is therefore not necessary to create a new mapping language.

The SPASQL implementation [Pru07] is an extension for MySQL to support SPARQL queries. It is thus technologically restricted to MySQL as it uses its query engine, although similar extensions could be built for other DBMSs. This extension is able to parse SPARQL queries and

compile them and directly execute them in the MySQL engine. The mapping is limited (e.g. no multi-field keys allowed) and it is not formalized.

Although automatic generation of ontologies and mappings can be useful in simple scenarios, for complex ones it is a limited approach. User expert knowledge may become necessary for complex mapping definitions, but it is also necessary to provide *well defined languages* that express those mappings.

The Virtuoso [EM07] declarative meta-schema language allows mapping relational schemas to RDF ontologies. SPARQL queries posed to the system are mapped to relational databases. The meta-schema is based on Quad Map Patterns that define transformations from relational columns into triples that match a SPARQL pattern. Notice that Virtuoso includes extensions to SPARQL for aggregates (COUNT, MIN, MAX, AVG, SUM) and it also allows SPARQL sub-queries.

In the D2RQ platform [BC07], a specific mapping language is introduced (D2RQ language) and formally defined by an RDF schema. The mapping defines relationships between an ontology and a relational database schema. The D2RQ Engine is implemented as a Jena graph. It provides a Jena or Sesame API plug-in for querying and reasoning with RDF. This plug-in rewrites the API calls to SQL queries which are executed in the database server, using the mappings. Results are returned as RDF triples.

In [PLC+08] they propose MASTRO, a DL reasoner for ontologies whose data is accessed through mappings in an external source, i.e. an ODBA-enabled reasoner. The reasoner works over the *DL-Lite_A* language, a fragment of OWL-DL. The mappings are specified through assertions that include SQL queries over the database. The language in this work is well formalized and the query answering complexity is well explored. The expressivity of the queries is limited to conjunctive queries (CQ). An implementation of a plug-in for data access to relational databases through SPARQL queries has also been built using this approach [PRR08].

The R₂O language [BCG05] provides formal mappings from relational databases to ontologies. The ODEMapster System uses this language to execute queries on the ontology that are translated to SQL to get results from the relational source. A complete description of this approach is detailed in Section 4.2.1.

4.1.2 Ontology-based Integration

Data integration can be defined as the process of providing *unified* and *transparent* data access to multiple and heterogeneous data sources [Len02]. The problem of integrating heterogeneous data sources involves not only providing access to information but also providing means of interpreting and processing the data in a consistent manner. When dealing with different sources it is often complicated to establish the meaning of the incoming data. And if the meaning cannot be clearly identified, then it is impossible to pair information for the different sources [RAY+00]. Problems of this kind are mainly due to confusion in term meanings and naming conflicts. Added to syntactical conflicts, the semantic integration challenge is hard to tackle.

We can identify the main issues of data integration as [WVV+01]:

- Incompatibility in communication and protocols, across different data management systems.
- *Syntactical heterogeneity*: differences in data model representation and structure, incompatibility of data values.
- *Semantic heterogeneity*: differences in naming and abstraction level, homonyms and synonyms in schemas.

Ontologies provide a means of explicitly specifying the meaning of the information that will be interchanged between systems. In that way it can help achieve semantic interoperability: regardless of the source's data syntax or semantics, we can map it to a known ontology and access the data in terms of the mediated ontological view [WVV+01]. As a consequence semantic declarative queries can be written in terms of the mediated schema.

It has been acknowledged that ontologies may be useful to solve this semantic heterogeneity problem as it is described in [WVV+01, DH05]. We can mention several systems and prototypes based on ontologies for data integration support: OBSERVER [MIK+00], SIMS [ACH+93], Carnot [HJK+93], DWQ [CDL+98], PICSEL [GLR00], MOMIS [BBV+07].

The main role and purpose of ontologies in an integration system is to explicitly state the semantics of the data sources so that it can be possible to identify and establish semantic relationships between these sources. For instance, if a data source stores person information and another source stores student information, the ontology may represent both concepts, with person subsuming student. This information can later be exploited by query integrators to coherently perform joins or unions over these sources.

Data sources can be relational databases, data files, xml files or any other storing media. Most of the work on data integration has been centred on relational databases, as it is the most widespread way of storing large volumes of information. In any case the ultimate goal is to allow querying the different sources through a uniform interface, hiding the underlying heterogeneous schemas. Therefore the applications querying the integrated system must only focus on the information they wish to obtain, leaving to the integration system the work of searching for the information in the different sources, clearly simplifying the development of upper layer applications on top of the integration infrastructure.

Research on the subject has produced many approaches to building such integration systems. We mention two main alternatives: the *virtualisation* and *materialisation* approaches [IKK05].

In the virtualisation approach queries are posed over the mediated schema and a mediator component identifies the sources that will be needed to produce the answer. Then a series of appropriate sub-queries are automatically written for these sources and executed. The results of each source are retrieved, post-processed and transformed to the mediated schema and returned to the caller application.

This alternative allows answering queries on demand, even if the update rate of data is high and the queries are potentially arbitrary. As the queries are rewritten dynamically for each of the sources, it is possible to retrieve any portion of data exposed through the mediated schema. On the other hand the transformations on the queries and the processing of the data from the sources

to the mediated schema may be complex and incur in performance issues. Another potential problem is data source availability, as the queries are performed live; if the source is unreachable then the queries may not be able to be completed. Even if the source is available, latency caused by networking or connection problems may increase the query response time.

The materialisation approach differs from the virtualised one in that it extracts and transforms the data from the sources periodically, instead of performing on-the-fly conversions each time a query is being executed. Instead, relevant data is first identified, and extracted from the multiple data sources in a batch process operation. The extracted data is then stored in a data warehouse where it can be accessed by the external applications. The data warehouse can then be updated periodically depending on the requirements. Under this approach, the time consumed in retrieving the data from the sources is moved to the off-line phase of updating the centralized data warehouse. Therefore when the external applications query the integration system, results can be accessed almost immediately. However the data may be stale in occasions, and it may not be possible to access all possible pieces of data, as the warehouse only stores selected materialised views of the original sources.

In the case of the virtualised approach, a mediator component is introduced as a major feature of the integration architecture. This mediator is in charge of transforming the original query into sub-queries for the sources and then transforming the results back to the mediated schema. This process of transformation is performed thanks to mapping definitions that establish relationships between the mediated and source schemas. There are three main alternatives for defining these mappings [Len02, IKK05]: *Global-as-view* (GAV), *Local-as-view* (LAV) and a combination of both, GLAV.

In the LAV approach, each of the source schemas is represented as a view in terms of the global schema. If our system I is represented as:

$$I = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$$

Where \mathcal{G} is the global schema, \mathcal{S} is a source schema and \mathcal{M} is the mapping between \mathcal{G} and \mathcal{S} , then the mapping assertions in \mathcal{M} will be a set of elements of the form:

$$s \rightarrow q_{\mathcal{G}}$$

Where s is an element of \mathcal{S} and $q_{\mathcal{G}}$ is a query over the global schema \mathcal{G} .

This approach is useful if the global schema is well established and if the sources suffer modifications constantly. Notice that changes in the sources do not affect the global schema. However, the query processing is not obvious, as it is not explicitly stated how to obtain the data in the mapping definition. Query rewriting techniques such as query answering using views can be used in this approach [Hal01]. Examples of LAV based works are DWQ [CDL+98], InfoMaster [GKD97], Information Manifold [Lev98], PICSEL [GLR00].

The other approach, GAV, defines the mappings in the opposite way. The global schema elements are represented as views over the source schemas. The mapping definition explicitly

defines how to query the sources to obtain the desired information in terms of the global schema. Following the system representation $I = \langle G, S, M \rangle$, in the GAV approach the mapping assertions are elements of the form:

$$g \rightarrow q_s$$

Where g is an element of the global schema G and is expressed as a view q_s , a query on the source schema. The advantage is that the mapping itself already indicates how to query the sources to obtain the data, so the processor can directly use this information to perform the query rewriting. The main disadvantage is that in case of changes or additions on the sources -e.g. a new source is added- then the global schema may suffer changes and this may affect other mapping definitions. Examples of GAV based works are SIMS [ACH+93], TSIMMIS [GHI+97], Carnot [HJK+93], Gestalt [RS98], MOMIS [BBV+07], IBIS [CCDG+03], DIS@DIS [CDL+03].

For the sake of completeness we can mention the GLAV (*Global-Local-as-view*) approach, which is a generalization of the previous two. This formalism allows more expressive mappings than LAV and GAV combined, and it has been shown to reach the limits of tractability of these description languages [FLM99].

4.1.3 Streaming Data Access

Managing streaming data differs significantly from classical static data. Streaming data is potentially infinite and its items are constantly added. Therefore queries on this data are also continuous, i.e. their results are updated regularly as time passes [TGN+92]. On the other side classical database systems deal with mostly static data, with lower insert rates and queries that retrieve the state of the data at the current time.

While in database systems datasets are queried several times, stream data is transient and usually the latest registered information is used for queries, and the old items are less relevant. These items are either discarded or archived. Sliding windows provide a suitable way of selecting the stream data that a continuous query will handle [BBD+02]. For example, a sliding window selecting tuples received in the last 30 minutes can be used to process only a desired subset of the incoming data. Otherwise it would be impossible to compute aggregates such as averages, maximums and minimums, as the streaming data is potentially infinite.

Although the continuous queries on data streams operate over the newest items, it is also useful to take into account older information. Several approaches exist to deal with this issue. For example, significant samples of past received data can be stored in static tables or aggregates (e.g. average) can also be stored in some repository [BBD+02]. Storage and processing of all the archived data is also a possibility but it is impractical in many scenarios, due to the high volumes of collected data.

One of the requirements of the SemSorGrid4Env project for WP4 (see Chapter 3) precisely indicates the need for accessing historic stored data, and combining it with other stored or

streaming data sources. Therefore we emphasize on the importance of querying systems providing the means to integrate information from both relational and streaming sources.

Streaming Data Queries

Several Data Stream Management Systems (DSMS) or systems managing continuous data, have been designed and built in the past years, such as Tapestry [TGN+92], Tribeca [Sul96], TelegraphCQ [CCD+03], Aurora [ACC+03, CCC+02], STREAM [ABB+04], Borealis [AAB+05], NiagaraCQ [CDT+00], Gigascope [CJS+02], CAPE [RDS+04], TinyDB [MFH+05], Cougar [YG02] and SNEE [GBJ+08].

Some of them have their own stream query language although all are based somehow on SQL. Although CQL (Continuous Query Language) [ABW06] has been regarded as the most prominent and best known of these languages and some standardization attempts have surfaced [JMS+08], there is still no common language for stream queries.

Formal models for stream management systems like [ABB+04] define both *streams* and *relations* and the operators that can be used on them. A stream [ABB+04, ABW06] S can be seen as a possibly infinite bag of elements (s,t) consisting of a tuple s and a timestamp t . A relation R is a finite bag of tuples at any instant t . This model has been used in various implementations and others have used it as a basis for extensions.

Stream systems require operators to limit streams to finite bounded structures in order to process only a smaller subset of data. Consider for example that we want to obtain the latest temperature values measured at the sea level. This is a subset of data that must be obtained from a stream of continuously appended temperature measurements. Windows are widely used to perform such transformations from streams to a bounded extents such relations [BGF+08]. The most common stream-to-relation operators are *time windows*. A time window usually takes a time interval as parameter and returns a relation containing tuples of the stream whose timestamp falls in that specified time interval. It can be said that a time window returns a finite snapshot of the stream in the window interval.

Time intervals in windows usually take the upper bound of the interval as the current timestamp. For instance in CQL the following statement:

S [RANGE T]

denotes the relation consisting of the elements of S whose timestamp falls in the latest T time units. For example S [RANGE 30 seconds] would result in a window consisting of the elements issued in the latest 30 seconds. More complex time windows and special cases (e.g. NOW, UNBOUNDED) are also considered in most streaming languages.

Sliding parameters are another usual feature of time-based windows. The slide parameter specifies how often the window is produced [ABW06, BGF+08]. For instance in CQL the following statement defines a window that slides every L time units:

S [RANGE T SLIDE L]

Even though time based windows are the most widespread stream-to-relation operator, other kinds of windows have also been studied, most notably *tuple-based* windows. A tuple window contains the last N tuples of a stream, where N is an integer. For instance S [Rows 4] will represent a relation with the last 4 elements of the stream S . A slide parameter can also be specified for tuple-based windows. Other sliding windows exist in previous works, such as partitioned windows, fixed windows, tumbling windows, etc [ABW06].

Apart from performing stream-to-relation transformations it is often required to transform relations to streams. Consider the following query for stream S with attributes a_1 and a_2 :

```
SELECT  $a_1, a_2$  FROM  $S$ [RANGE 10 MIN SLIDE 10 MIN]
```

The result of this query is a window containing the a_1 and a_2 attributes of S produced in the latest 10 minutes. The SLIDE parameter indicates that a new window will be created every 10 minutes. It may be however desired to build a single stream out of those windows that contains all their tuples. For those and other situations, relation-to-stream operators have been studied. The main operators available in most querying languages are RSTREAM, ISTREAM and DSTREAM [ABW06].

The ISTREAM operator applied to a relation R at time t , produces a stream that contains all tuples of R that are in R at time t but not in $t-1$. This behaviour informally specifies that the produced stream contains only the inserted elements in a relation at each time. The DSTREAM operator applied to a relation R at time t produces a stream that contains all tuples of R that are not in R at time t but were there in time $t-1$. Therefore the resulting stream contains the deleted elements of R at each time. Finally the RSTREAM operator applied to a relation R at a time t produces a stream that contains all tuples of R at time t .

Sensor Networks Query Processing

Sensor Networks are one of the main sources of continuous streaming data [ALM+04]. Sensor networks consist of multiple interconnected nodes that are able to sense and capture different kinds of data. It is evidently necessary to provide means to query data collected by these networks. In order to solve this requirement, research has produced Sensor Networks Query Processing engines such as TinyDB [MFH+05], Cougar [YG02] and SNEE [GBJ+08] (see Section 4.2.3). These processors use declarative query languages for continuous data like the ones we described earlier in this section. These declarative queries describe logically the set of information that is to be collected but leaves to the engine to determine the algorithms and plans that are needed to get the data from the different nodes of the sensor network.

These engines must also consider several optimization techniques in order to efficiently gather the information from the sensor nodes. Sensor networks are characterized by strong resource limitations such as *energy consumption*, *computing power* and *storage capabilities*. Architectures for query optimization in these constrained scenarios have surfaced [GBJ+09, MFH+05], showing that even with such limitations it is still possible to use rich and expressive declarative query languages.

Streaming Data Integration

As we have seen, in scenarios like sensor networks, there is an in-network integration among the participating nodes. This is a low level integration in the query processors that considers the different nodes where the requested data is being sensed [GBJ+09].

However the distributed inner nature of sensor networks is not the only case of distributed execution. In many stream management systems such as Borealis [AAB+05], D-CAPE [SLJ+05] or Harmonica [KW07] the architecture has been extended in order to work as a distributed stream processor. The continuous queries issued to the system are distributed and processed across different sites. Each of these sites could even be a wrapper of a Sensor Network or whatever stream data source.

In the Borealis System for instance, each node has a Query Processor, Optimizer, Monitor and Communication module. Then all queries can be distributed to all nodes of the system, with each processor performing part of the execution.

As it is described in [ALM+04], the integration of sensor networks to each of the Query Processor nodes is possible through proxy wrappers. These wrappers make each of these integrated sensor networks appear as if was another node of the distributed streaming system. Consequently when a query is issued, some part of the plan can be delegated to this node, and the wrapper will be in charge of transforming the plan operators to some implementation in the sensor network and possibly perform optimizations. Figure 4.1 shows this architectural approach:

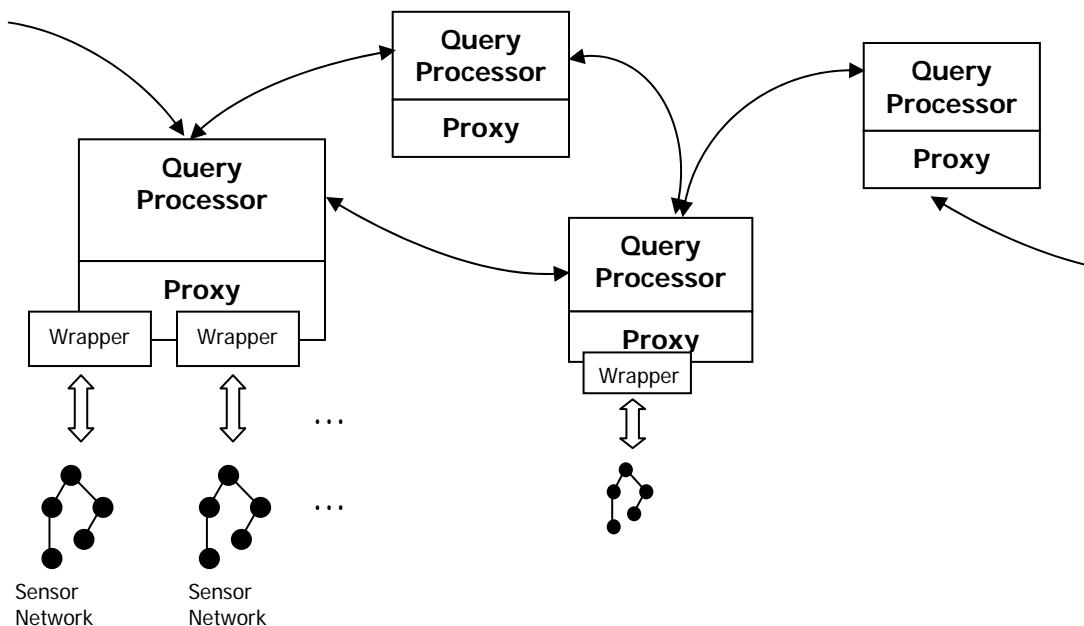


Figure 4.1: Architecture of a distributed stream processing engine [ALM+04].

A similar approach of a mediated architecture and wrappers is also applied in [IKK05]. As it can be seen, in distributed scenarios it can be advisable to use distributed processing techniques and approaches that are detailed in the next section.

4.1.4 Distributed Query Processing

Data integration using *Distributed Query Processing* arises as a requirement because of several factors. In large-scale systems it is impractical and unfeasible to expect building a centralized data repository. From the hardware point of view large systems can scale up by adding processing nodes to it. This option is cheaper and simpler than upgrading a single centralized server. Moreover the physical distribution of servers can also have geographical and institutional requirements. Another key point is data integration across heterogeneous systems, including legacy systems. Each individual system is responsible of executing its own part of the processing and there is an integrator component charged with the orchestration of the execution and aggregation of results.

Even if there are several use cases and applications for distributed query processing, a common basic architecture has been defined in [HFL+89, Kos00] and it is summarised below. In the most general sense a declarative query is received by the processor, it is translated to an internal representation and then optimized. An executable query plan is generated and then results are obtained and returned to the caller. We can identify the following components in a query processor [Kos00, HFL+89]: *Parser*, *Query Rewriter*, *Optimizer*, *Code generator*, *Query Executor* (See Figure 4.2).

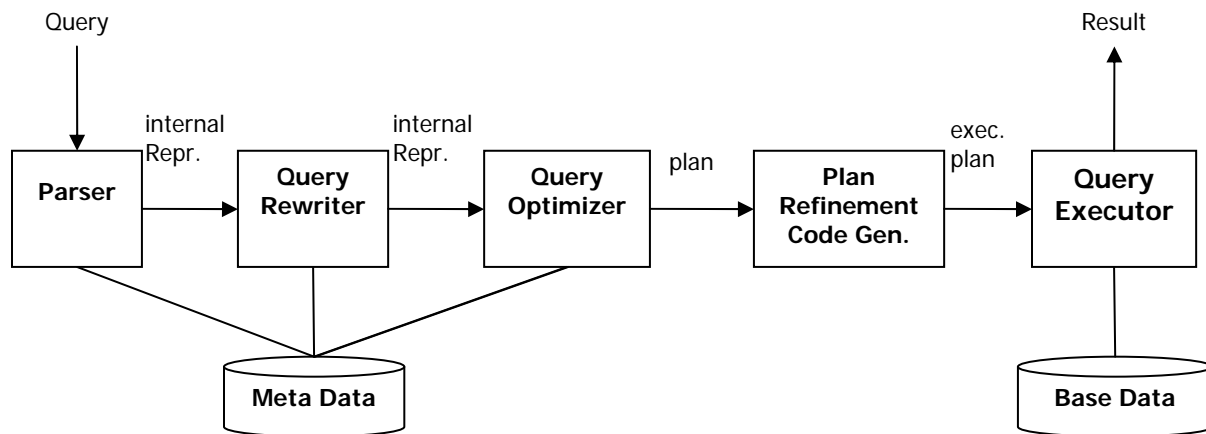


Figure 4.2: Phases of query processing [Kos00].

The *parser* is in charge of translating the received query (e.g. SQL query) into an internal representation. The parser will check for syntax correctness and the resulting internal representation, a parse tree, will be used later by the other components of the processor. The parser will perform this validation and transformation based on the formal grammar of the query language and will also check for semantic correctness at some extent. It will identify keywords

and literals, relation names, aliases and attributes. Then it will check for existence of relations and attributes, compatibility of operators and types, etc.

Then the *query rewriter* performs transformations and optimizations using, for instance, a rule engine. These optimizations consist in discarding redundancies in the query formulation and simplifying expressions by the use of equivalences. Un-nesting and decomposing queries for better performance can also be considered during query rewriting. Removal of sub-queries and replacing them with other constructs can also be performed at this stage. Identification of empty sets and tautologies and other simplifications can also be expected.

Next the *query optimizer* will be in charge of generating a query plan. In order to do so the optimizer will decide about the right indexes to use, the internal methods for executing the query operations and the order of these operations (scan, hash, sort, join, etc). It will also decide which operations will be executed in which nodes of the distributed system. Many plans may be possible for a given query, so the optimizer will have to choose the best one according to cost and query response time criteria. Dynamic programming algorithms have been used to solve the problem of enumerating and choosing the best query plan in relational databases.

After the optimization phase, the query plan is transformed into an executable query plan, where the algorithms for each operator are specified and the code for the complete plan is generated. This executable plan is taken by the *query execution engine*, which provides implementations for the operators specified in the plan. Query execution in a distributed processor will include dealing with communication between the nodes. Sending and receiving tuples in blocks with send and receive operators is one of the usual techniques for this problem. Communication must also consider optimizing the paths to send messages among processing nodes. The query executor can also consider possible multithreading execution of query plans. Although it can be useful and faster to parallelise the execution of operations, this is not always the case, for instance if communication is expensive anyway. Therefore algorithms for deciding when to run parallel threads are necessary.

As we can see, although the general architecture for query processing can be applied to a distributed scenario, there are certain issues that require particular attention. In the case of integration of heterogeneous sources, a more specific architecture variant can be specified, based on a mediator and wrappers [Wie92, Kos00].

4.1.5 Quality of Service

Quality-of-Service (QoS) refers to the set of non-functional properties and attributes of a delivered service [KP06]. QoS includes not only network-related properties such as bandwidth or latency but also other parameters that may affect the quality of the delivered service, such as security, availability, reliability, performance, etc. QoS parameters can be very useful in Service-oriented environments, for instance in service design and service description. QoS specifications can be used to discriminate services providing similar or equivalent functionality [KP06, KP07]. Moreover service consumers can use QoS parameters to interact with the services in appropriate ways. For instance in terms of trust or uncertainty of results, data can be labelled as untrusted but still be usable by a consumer. Or if a service is known for its low availability, it may be excluded

from certain interactions. In that way, during service discovery process, QoS may have a great impact on matchmaking and ranking. QoS can also be used in data integration and query processing [ACC+03, CCR+03, GBJ+09, NLF99, YKB99]. Algorithms can use QoS information to decide which sources are suitable for a particular distributed query in a data integration system.

In order to specify QoS requirements, QoS metrics must be established. Then the QoS specification will be defined as a set of constraints on those metrics [KP07], with ranges or sets of possible and expected values. Describing these parameters, requirements, metrics and values needs an underlying expressive model capable of representing them. Lot of effort has been put into QoS Management and QoS models have surfaced, including ontology-based models [KP07, JBM08, DLS05].

In the context of query processing, QoS has also been used especially for optimization in distributed scenarios. In Aurora [CCR+03] the system models QoS as a multidimensional function of attributes including response times, tuple drops and values produced. Aurora tries to maximize the QoS for the query operations it executes. For example in case of overload Aurora performs QoS guided load shedding. In [GBJ+09], the SNEE compiler/optimization stack uses QoS expectation parameters -namely acquisition rate and delivery time- for the execution scheduling algorithm.

4.2 Related Technologies

In this section we will explore in more detail the technologies that will be used, extended or used as a basis in the ontology-based data integration solution of the SemSorGrid4Env project. First we provide details about R₂O, a mapping language designed to establish relationships between relational and ontological elements for ontology-based data access (see Section 4.2.1). Then we present ODEMapster, a system that uses the R₂O language to transform ontological queries into relational queries and return results in RDF (see Section 4.2.2). Next we describe the SNEE processor for sensor networks and its continuous data query language SNEEql (see Section 4.2.3). In Section 4.2.4 we introduce SPARQL extensions for streaming data. Finally we present OGSA-DQP, a distributed query processing system (see Section 4.2.5).

4.2.1 R₂O

R₂O (*Relational to Ontology*) [BCG05] is a mapping definition language that defines a relationship between an ontology and local relational schemas. It follows the GAV approach and therefore each element in the global ontology is characterized in terms of a query over the relational sources. The R₂O language is XML-based and independent of any specific DBMS. It allows complex mapping expressions between ontology and relational elements.

R₂O mapping assertions can be created either manually or with the help of a mapping tool. The mapping definition could also be used to perform validation of the database integrity by checking it against the ontology restrictions and axioms.

It is important to notice that R₂O does not define mappings in both senses, i.e. it does not provide a mapping definition from the ontology to the relational schema. The mapping assertions are designed to be used by applications and middleware systems, not by end users.

R₂O specifically considers Concepts, Attributes and Relations in an ontology. They are described in terms of selections and transformations over database tables and columns. In order to be able to describe these mappings, R₂O defines the ontology elements and also the database elements using its own XML notation.

R₂O covers a wide set of mapping cases common in relational to ontology situations. The domain covered by the ontology and the database more than often differ. In some occasions they coincide on some concepts, so the domains intersect. In other occasions the ontology domain includes the relational database domain or vice versa. Depending on the scenario, finding correspondences between ontological elements and relational elements can be straightforward or complex. R₂O is designed to cope with the cases detailed below.

- In the simplest case a database table maps one concept in the ontology. Then the table columns map to attributes or relations of the concept. For each row in the table a corresponding instance in the ontology will be generated, with its attribute values filled with the columns data.
- In a second case, a single database table is mapped to more than one concept in the ontology, and for each row a single instance of each concept is generated. In this case some columns will be mapped to a concept attributes while other columns will be mapped to another concept attributes. In some way each record is split into two ontology instances of different concepts.
- In a third case, a single database table is mapped to more than one concept in the ontology, and multiple instances can be generated for each concept. It is a more general case than the previous one; multiple instances of the same ontology concept can be generated from a single database record.

Mapping tables and columns to concepts and attributes often requires performing some operations on the relational sources. Several cases are handled by R₂O and detailed below.

- **Direct Mapping.** When the relational table maps an ontology concept and the column values are used to fill the attribute values of the ontology instances. Each table record will generate a concept instance in the ontology.
- **Join/Union.** In some occasions a single table does not correspond alone to a concept, but it has to be combined with other table or tables to match the ontology concept. The result of the join or union of the tables will generate the corresponding ontology instances.
- **Projection.** Sometimes not all the columns are required for the mapping. The unnecessary columns can simply be ignored. In order to do so, a projection on the needed columns can be performed (e.g. a SELECT).
- **Selection.** In some situations not all the records of a table correspond to instances of the mapped ontology concept. Then a subset of the records must be extracted. To do so, selection conditions can be applied to choose the right subset for the mapping.

It is of course possible to combine joins, unions, projections and selections for more complex mapping definitions.

Values from the database records can be copied as-is to the attributes of instances in the ontology. However in many situations it is necessary to perform some transformations on the values using some function. R₂O allows the use of defined functions for this purpose, e.g. concatenation, sub-strings, arithmetic functions, etc.

4.2.2 ODEMapster

The ODEMapster [BG06] system is an ontology-based data processor that provides access to data in a relational database through declarative queries over an ontology. The ODEMapster System uses the R₂O mapping language to define how the ontology elements must be accessed in terms of the relational tables. The query language used by ODEMapster over the ontology is ODEMQ, an XML-based language that is at least as expressive as SPARQL for conjunctive queries [Fus08]. The ODEMapster architecture is depicted in Figure 4.3.

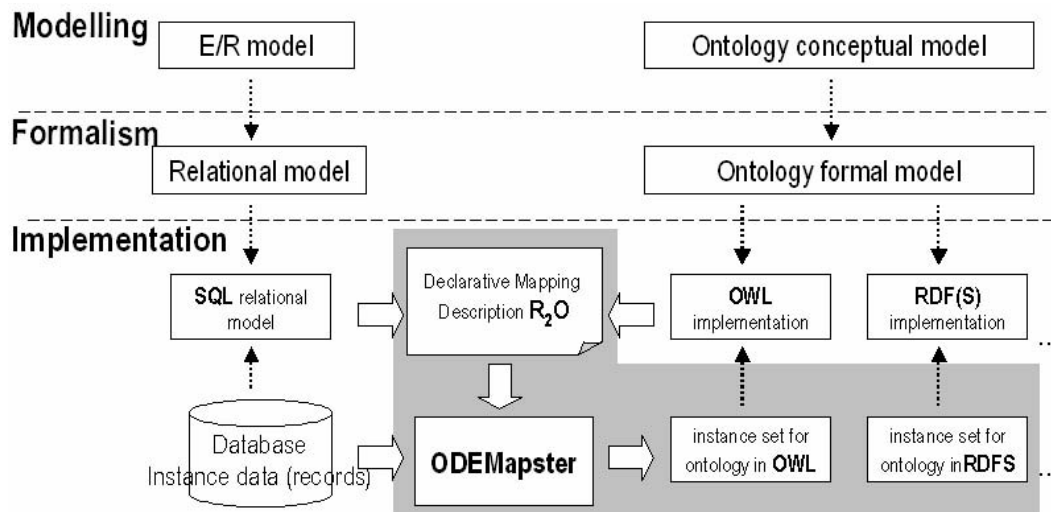


Figure 4.3: ODEMapster Architecture [Bar06].

The ODEMapster System may work on one of two operation modes: virtualised and materialised. The virtualised operation mode, as shown in Figure 4.4, acts as a query translator of the ODEMQ queries into SQL queries over the relational tables. The queries are immediately executed in the sources and the results returned as instances in terms of the ontology.

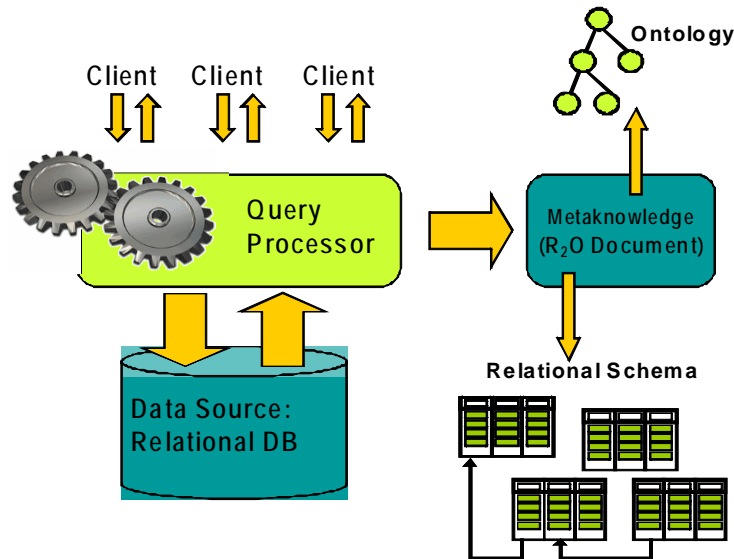


Figure 4.4: ODEMapster virtualised execution mode [Bar06].

The materialised mode, as shown in Figure 4.5, executes the transformation as a batch process, creating a RDF repository taking the entire relational database as input.

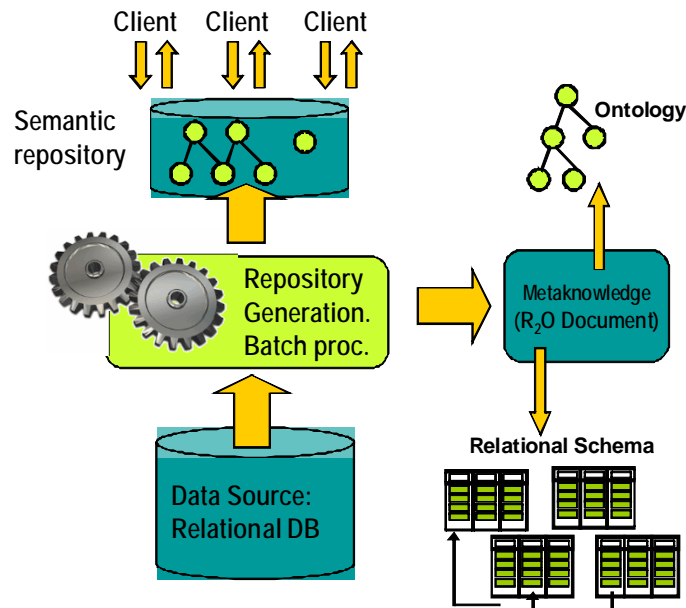


Figure 4.5: ODEMapster materialised execution mode [Bar06].

The execution of an ODEMapster query follows the following steps, depicted in Figure 4.6:

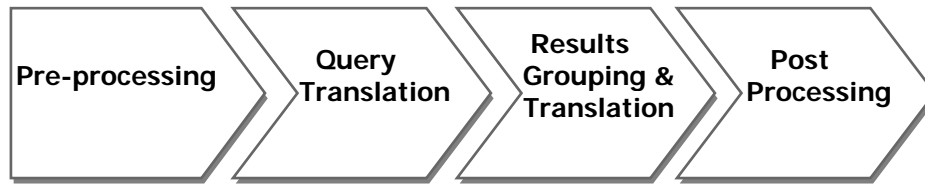


Figure 4.6: ODEMapster execution steps.

Pre-processing

First the query and the mapping definition are parsed. This phase verifies the integrity of the database schema and the ontology, and then it verifies that the resulting elements from the query are already defined in the mapping. It also validates the data types of all the involved elements of the ontology and database schema.

Query Translation

In this phase the engine proceeds to generate the corresponding SQL queries to be executed in the data sources. This includes the following steps:

1. First it checks whether it is possible to translate the query or not. This checking process is performed by identifying if a corresponding maximum query equivalent exists or not.
2. If a corresponding maximum equivalent query exists, then for every element in this query (concept, attribute, or relation) the explicitation of that element is generated.
3. The explicitations obtained in the previous step are transformed into relational algebra expressions.
4. The algebra expressions from the previous step are transformed into SQL queries that are ready to be executed.

Results Grouping & Translation

The final result of the translation phase is a set of SQL queries that are ready to be executed. Once these queries have been executed and the relation DBMS returns the results, these are translated back to instances of the ontology.

Post-processing

The final phase is to deliver the final result which is an RDF document containing ontology instances. In this phase, the non delegable expression is evaluated by ODEMapster. A non delegable expression is an expression that is specific for every DBMS. As ODEMapster is DBMS independent, the non delegable expression cannot be translated in the Query Translation phase. So it has to be evaluated by ODEMapster, which maps these non-delegable expressions to the DBMS specific constructs.

ODEMapster has continued to evolve. A tool for facilitating the creation of R₂O mapping documents and evaluation of queries has been developed, the ODEMapster plug-in for NeOn Toolkit [HLS+08].

An extension of R₂O for defining mappings from an ontology to multiple databases has been created. The corresponding extension to ODEMapster for accessing multiple databases has been implemented as well [Pri09]

4.2.3 SNEEq1

In the SemsorGrid4Env project, we will focus on the SNEEq1 [BGF+08] language for querying streaming data sources. Although it is based on the widely known CQL [ABW06], it provides greater expressivity in queries, including streams and relations, time and tuple windows, blocking and non-blocking operators, and pull and pushed based streams. In [BGF+08], the SNEEq1 data model presents compound types: tuples and tagged tuples. A tuple is a set of typed attributes, e.g.:

$$a_1:t_1, a_2:t_2, \dots, a_n:t_n$$

Where $a_1..a_n$ are the attribute names and $t_1..t_n$ are types. A tagged tuple is a tuple that includes two named attributes: tick and index. The tick attribute is a timestamp that indicates when the tuple was created. This is similar to stream elements in CQL where each tuple has a corresponding timestamp t . This attribute is essential to define the semantics of stream operators in these languages. The index named attribute of a typed tuple is a position of the tuple in a sequence. SNEEq1 defines two collection types: windows and streams. A window is a pair consisting of a tick and a bag of tuples. A stream is a potentially infinite sequence of tagged tuples or windows.

SNEEq1 defines stream queries and window queries. A stream query is of the form:

```
SELECT  $a_1 \dots a_n$  FROM  $s$  WHERE  $p$ 
```

where $a_1 \dots a_n$ is a project list, s denotes a stream of tagged tuples (i.e., either the name of a sensed or pushed extent, or a sub-query of type stream), and p is a predicate [BGF+08]. The result of the execution of a stream query is a stream of tagged tuples.

Window queries are of the form:

```
SELECT  $a_1 \dots a_n$  FROM  $w_1 \dots w_m$  WHERE  $p$ 
```

where $a_1 \dots a_n$ is a projection list of attributes of the stream, $w_1 \dots w_m$ is a list of window definitions, and p is a predicate. [BGF+08]

The result of the execution of a window query is a stream of windows. A window on as stream can de specified as follows:

```
 $s$  [FROM  $t1$  TO  $t2$  SLIDE int unit]
```

where FROM $t1$ TO $t2$ indicates a time interval, e.g. NOW-10 denotes the current tick minus 10 time units. The slide parameter indicates the frequency of the window creation in time units or rows.

A window on a table can be specified as follows:

t [SCAN int timeUnit]

where t is a table. The scan parameter indicates the frequency of the table scan and its corresponding windows creation.

The SNEEqL stream-to-window operator previously described follows the ideas of CQL, and other languages have similar constructs as well, mainly regarding windows. Notice that windows are not only time dependant, but also may be tuple (row) dependant.

Window-to-stream operators are also required for stream query languages. Based on the CQL [ABW06] operators, SNEEqL provides the ISTREAM, DSTREAM and RSTREAM operators. The ISTREAM operator appends tuples that were not on the previous window, to the output stream. DSTREAM appends tuples that were deleted from the previous window. RSTREAM appends all tuples from the previous window to the output stream.

Queries over streams usually require aggregation functions. Querying for sums, averages, maximums and minimums is quite common. Streaming query languages such as CQL and SNEEqL provide support for this kind of operators, for either streams or relations.

Another feature of some query languages is the possibility of specifying quality of service requirements. SNEEqL allows for instance specifying the desired acquisition rate and delivery time:

{ACQUISITION RATE = 3s ; DELIVERY TIME = 5s}

These parameters can be used by the query engine to perform optimizations on the query plans and these can be extremely useful in the context of sensor networks because of the limited processing and power resources.

4.2.4 SPARQL for RDF Streams

As it is described in Section 4.1.3 query languages for data streams have been proposed and implemented in the last years. These languages borrow much of the relational query languages such as SQL. In ontology-based integration solutions, queries are expected to be posed in terms of an ontological view. In order to do so, it is necessary to have a stream query language that natively supports semantic models [DCB+09, GGK+07].

SPARQL [PS08] is the W3C Recommendation for a query language over RDF. SPARQL has been used to query RDF triples annotated with time constructs such as timestamps [BBC+09,

BGJ08] can be used to represent data coming from streaming sources. However SPARQL currently lacks the necessary operators to effectively query streaming data.

In [BGJ08] extensions for SPARQL are provided, so that the resulting language is able to handle RDF based data streams. The semantics of this extension is also provided as well as the algorithm to map the language extension to the extended algebra. They call this extended language Streaming SPARQL.

The extended grammar of Streaming SPARQL basically consists in adding the capability of defining windows over streams which are explicitly stated using the `STREAM` keyword. The windows are defined in the `FROM` section, using the `WINDOW` keyword. Both time-based and tuple based windows are supported. For time-based windows the `RANGE` keyword allows specifying the window size in time units. A `SLIDE` parameter can be specified to indicate the frequency of the window creation. For tuple-base windows the `ELEMS` keyword is used instead of `RANGE`. Here's an example of a Streaming SPARQL query, it obtains the sensor temperature values sensed in the latest 30 minutes, every minute:

```
PREFIX fire:<http://www.semsorgrid4env.eu/ontologies/fireDetection#>
SELECT ?sensor ?temperature
FROM STREAM <http://.../Sensors.rdf>
WINDOW RANGE 30 MINUTE SLIDE
WHERE { ?sensor fire:hasTemperatureMeasurement ?temperature . }
```

Another extension to SPARQL for continuous queries is C-SPARQL (Continuous SPARQL) [BBC+09]. C-SPARQL also works over RDF Streams, sequences of triples annotated with non-decreasing timestamps. As in Streaming SPARQL, it defines sliding windows, time or tuple-based. C-SPARQL offers aggregates, such as `COUNT`, `SUM`, `AVG`, `MIN` and `MAX`. It also allows combining static and streaming knowledge and also combining multiple streams. Here's an example of a C-SPARQL query, it obtains the temperature average of the values sensed in the last 10 minutes:

```
REGISTER QUERY AvergaeTempreature AS
PREFIX fire:<http://www.semsorgrid4env.eu/ontologies/fireDetection#>
SELECT DISTINCT ?sensor ?average
FROM STREAM <www.uc.eu/tollgates.trdf> [RANGE 10 MIN STEP 1 MIN]
WHERE { ?sensor fire:hasTemperatureMeasurement ?temperature . }
AGGREGATE {{?average, AVG, {?temperature}}}
```

Notice that neither language supports time windows with other upper bound different than the current timestamp (now). Relation-to-stream operators are also missing in these specifications. Only Streaming SPARQL provides an extended algebra for these streaming features. C-SPARQL still lacks any formal description of the semantics of the language.

4.2.5 OGSA-DQP

As we have seen in Section 4.1.4 distributed query processing capabilities are needed to achieve data integration of heterogeneous sources. OGSA-DQP (Open Grid Services Architecture - Distributed Query Processing) [AMG+04, LMH+09] is a service-based data integration and orchestration framework that is capable of evaluating declarative queries over several existing services on a grid infrastructure. It relies on OGSA-DAI (Data Access and Integration) [OGS09] implementation to access the structured data sources and manage the service instances.

OGSA-DQP consists of two main components: Coordinator Service and Evaluator Service, as depicted in Figure 4.7 [LMH+09]. The Coordinator is an extended OGSA-DAI data service that operates in two phases: setup and query evaluation. During the setup phase the data sources to be used are identified and their schemas and metadata are retrieved. Metadata is later used during optimisation to build the query execution plans. After the schema has been successfully imported, a DQP data resource is exposed through a created and initialised data resource accessor, to which queries can be sent later.

In the evaluation phase a SQL query is received from the client, and the DQP parser is the module in charge of creating a tree from that query. The SQL expression parser supports a subset of SQL 92 [SQL92] plus user defined functions (UDF) which were introduced as an extension point for the DQP query language. UDFs allow developers to add support for arbitrary functionality within the DQP. Then an SQL parser is generated from a grammar. The parser produces an abstract syntax tree (an internal representation of the SQL expression) which is then passed to the query plan builder.

The query plan builder takes the abstract syntax tree generated by the SQL parser and produces a logical query plan. A logical query plan is a directed graph whose nodes are relational operators. To this logical query plan several optimisers are applied. The optimisations are heuristic based optimisations, cost based optimisations or try to push the select clauses as next to the OGSA-DAI data resource as possible. The optimisers are the following:

- Query Normaliser
- Select Push Down Optimiser
- Rename Pull Up Optimiser
- Project Pull Up Optimiser
- Join Ordering Optimiser
- Join Annotation
- Partitioning Optimiser
- Table Scan Implosion Optimiser
- Filtered Table Scan Optimiser

The next step in the process is to partition the query plan in the nodes it is going to be executed. OGSA-DQP provides an algorithm in charge of partitioning the logical query plan according to the data nodes from which the datasets originate. If a query plan contains a join or product of two data streams that are located on different data nodes the partitioning algorithm detects this and transforms the query plan by inserting exchange operators that represent data transfers between

two remote data nodes. The output from the partitioner is a set of partitions and the query plan in which every operator is annotated with the partition to which it belongs.

Once the query plan has been created, optimised and distributed across the nodes it is time for executing it. From this planning a set of OGSA-DAI workflows is created. Each of these workflows represents a partition of the logical query plan created and these workflows are connected through their inputs and outputs to produce the result of the query. Once the workflow has been created the generated remote requests and local sub-workflows are executed and the results collected and returned to the client. A complete description of the Coordinator-Evaluator interactions in OGSA-DQP can be found at [LMH+09].

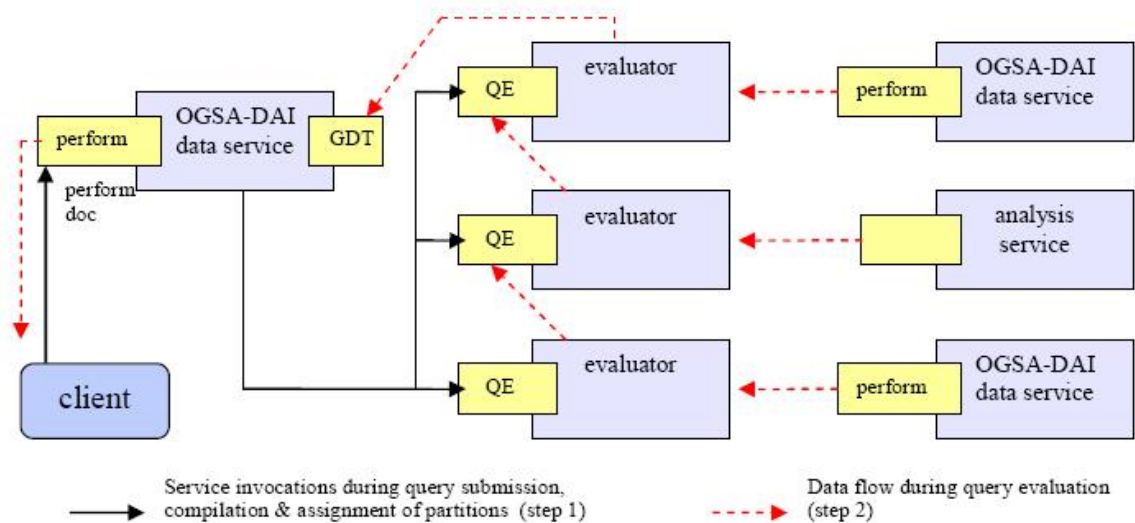


Figure 4.7: OGSA-DQP Architecture [LMH+09].

5 Semantic Integrator Proposal

This chapter describes the identified challenges and design proposal for the Semantic Integrator Service. The integration encompasses two main stages: the integration of data sources itself and the query processing. First we will present a walk-through of the proposed integration process in Section 5.1. Then in Section 5.2 we will explain how this proposal implements the interfaces described in Chapter 2 and fulfil the requirements of Chapter 3. Finally we point out the future works for WP4 in Section 5.3.

5.1 Integration of Data Sources: Walk-through

In this section we will describe the different phases of the integration of heterogeneous streaming and stored data sources.

Before actually accessing heterogeneous streaming and relational data sources, applications are expected to specify which sources they want to access, and proceed to create a reference to an integrated data source with a global schema. Each source is identified by an abstract name, previously registered in the system. The sources are typically data streams or relational databases through Streaming Data Services and Stored Data Services respectively (see Section 2.1). In either case these services provide the data sources metadata, including connection information, schema and quality of service parameters.

5.1.1 Mapping to the Ontology

Apart from the reference to the data sources, the Semantic Integrator requires a mapping document that describes how to transform the data source elements to ontology elements. The mapping document is an extended version of R₂O (Section 4.2.1).

As it was explained in Section 4.2.1, R₂O includes a component in the mapping document that describes the database tables and columns, `dbschema-desc`. In order to support streams, R₂O will be extended to also describe the data stream schema. A new component called `streamschema-desc` will be created, as in the following example:

```
streamschema-desc  
name CoastalSensors  
has-stream  
  name SensorWaves  
  streamType pushed  
  documentation "Wave measurements"
```

keycol-desc**name** measurementID**columnType** integer**timestamp-desc****name** time**columnType** timestamp**nonkeycol-desc****name** mes_height**columnType** float**nonkeycol-desc****name** mes_temperature**columnType** float

The description of the stream is similar to a table's. An additional attribute `streamType` has been added, it denotes the kind of stream in terms of data acquisition. It can be a sensed stream, i.e. pull based arriving at some acquisition rate. Or it can be pushed, arriving at some variable potentially unknown rate. Relations can also be specified just like tables in R_2O . Just as key and non key attributes are defined, a new `timestamp-desc` element has been added to provide support for decalring the stream timestamp attribute.

For the concept and attribute mapping, the R_2O existent definitions can be used for stream schemas just as it was for relational schemas. This is specified in the `conceptmap-def` element:

conceptmap-def**name** Wave**identified-by** SensorWaves.measurementID**uri-as**

<transformation>

applies-if

<cond-expr>

described-by**attributemap-def****name** height**selector****aftertransform****constant****arg-restriction****on-param** const-val**has-column** SensorWaves.mes_height

In addition, although they are not explicitly mapped, the timestamp attribute of stream tuples could be used in some of the mapping definitions, for instance in the URI construction (`uri-as` element). See the Annex (Linked Stream Data: A Position Paper) for guidelines for creating URIs for streams.

As it has been seen, R_2O requires some changes to support creating mapping documents for stream schemas. It is expected that common and simple mappings in the R_2O language can be defined using the ODEMapster tool, enhanced with streaming data support.

As an example, consider these concepts and relations in an ontology, where Sensors may measure wind speed and direction (see Figure 5.1):

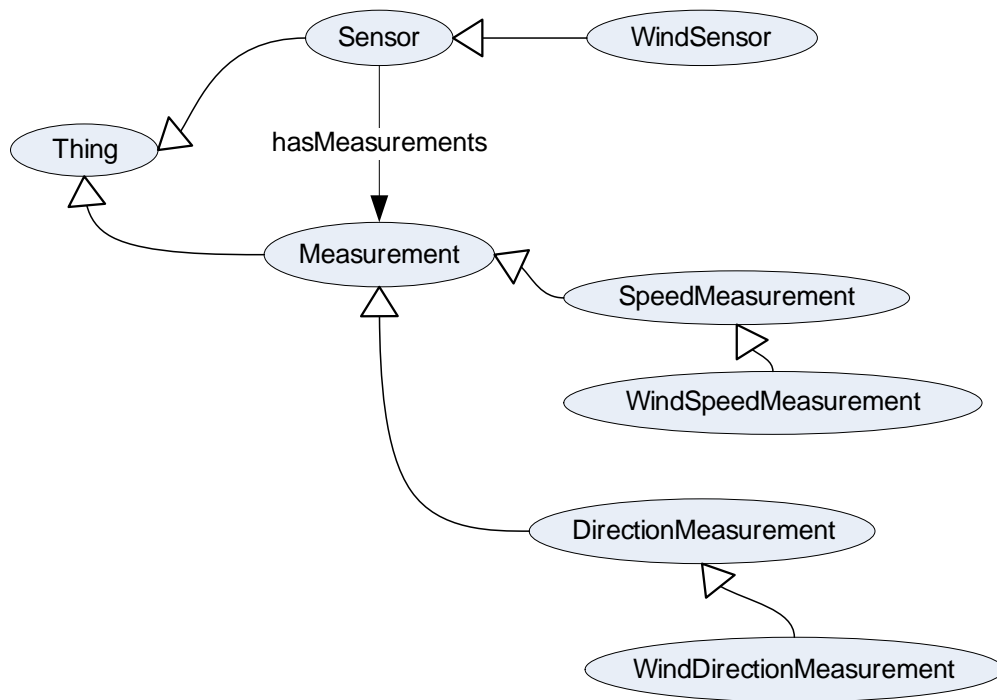


Figure 5.1: Sample Fire Ontology.

Now consider a stream Winds that collects wind speed and direction values. A simple mapping can be represented in Figure 5.2:

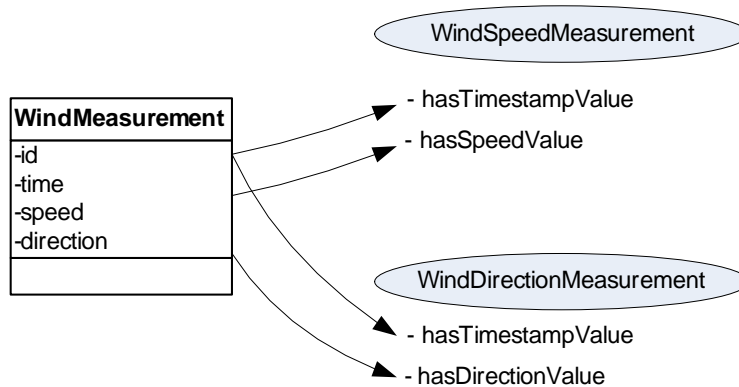


Figure 5.2: Sample mapping from stream-to-ontology.

Notice that in this simple example a single stream is split into two concepts. Therefore for each tuple in the stream there will be two different ontology instances. If a query aims at recovering only the speed measurements, then a corresponding projection operation over the stream must be executed.

5.1.2 Querying over the Ontology

Once the sources have been mapped to the ontology, it must be possible to write queries over the unified ontological view. ODEMapster originally used its own query language, ODEMQ. In [Fus08] it has been shown that it is feasible to use SPARQL conjunctive queries instead. The Semantic Integrator will support an extended version of SPARQL, similar to Streaming SPARQL [BGJ08] and C-SPARQL [BBC+09] query languages.

We consider the following features offered by both Streaming SPARQL and C-SPARQL (Section 4.2.4), taking into account the requirements stated in Chapter 3:

- Time based windows
- Tuple based windows
- Window Slide parameters
- Support for stored data (only in C-SPARQL)
- Aggregates: COUNT, SUM, AVG, MIN, MAX (only in C-SPARQL)

And we aim at providing the following additions:

- Upper bound time window specification, FROM – TO windows construct.
- Relation-to-stream operators, RSTREAM, ISTREAM, DSTREAM.

Given the limitations of both languages we will propose our extension of SPARQL for continuous queries. However this decision will depend on a deeper analysis of the semantics of these languages. Using this extended SPARQL language we will be able to write declarative queries over the ontological global schema. Consider the simple ontology presented in Figure 5.1. The following extended SPARQL query collects the latest wind speed and direction measurements sensed. This query can be sent to the Semantic Integrator as a query on the ontological integrated view. The syntax is taken from Streaming SPARQL but is given only for illustrative purposes.

```
PREFIX fire: <http://www.semsorgrid4env.eu/ontologies/fireDetection>
SELECT ?sensor ?speed ?direction
FROM STREAM <http://.../SensorReadings.rdf> WINDOW RANGE 10 MIN
WHERE {
  ?sensor a fire:WindSensor;
    fire:hasMeasurements ?WindSpeed, ?WindDirection;
  ?WindSpeed a fire:WindSpeedMeasurement;
    fire:hasSpeedValue ?speed;
    fire:hasTimestampValue ?wsTime.
  ?WindDirection a fire:WindDirectionMeasurement;
    fire:hasDirectionValue ?direction;
    fire:hasTimestampValue ?dirTime.
}
```

5.1.3 Query Execution

The queries written in the global streaming query language can be submitted to the Semantic Integrator so that it rewrites them to either SQL or SNEEq and returns the results. This process is divided in phases managed by the following components of the query processor: *Parser*, *Rewriter*, *Plan Builder & Partitioner*, *Evaluator* and *Integrator* (See Figure 5.3).

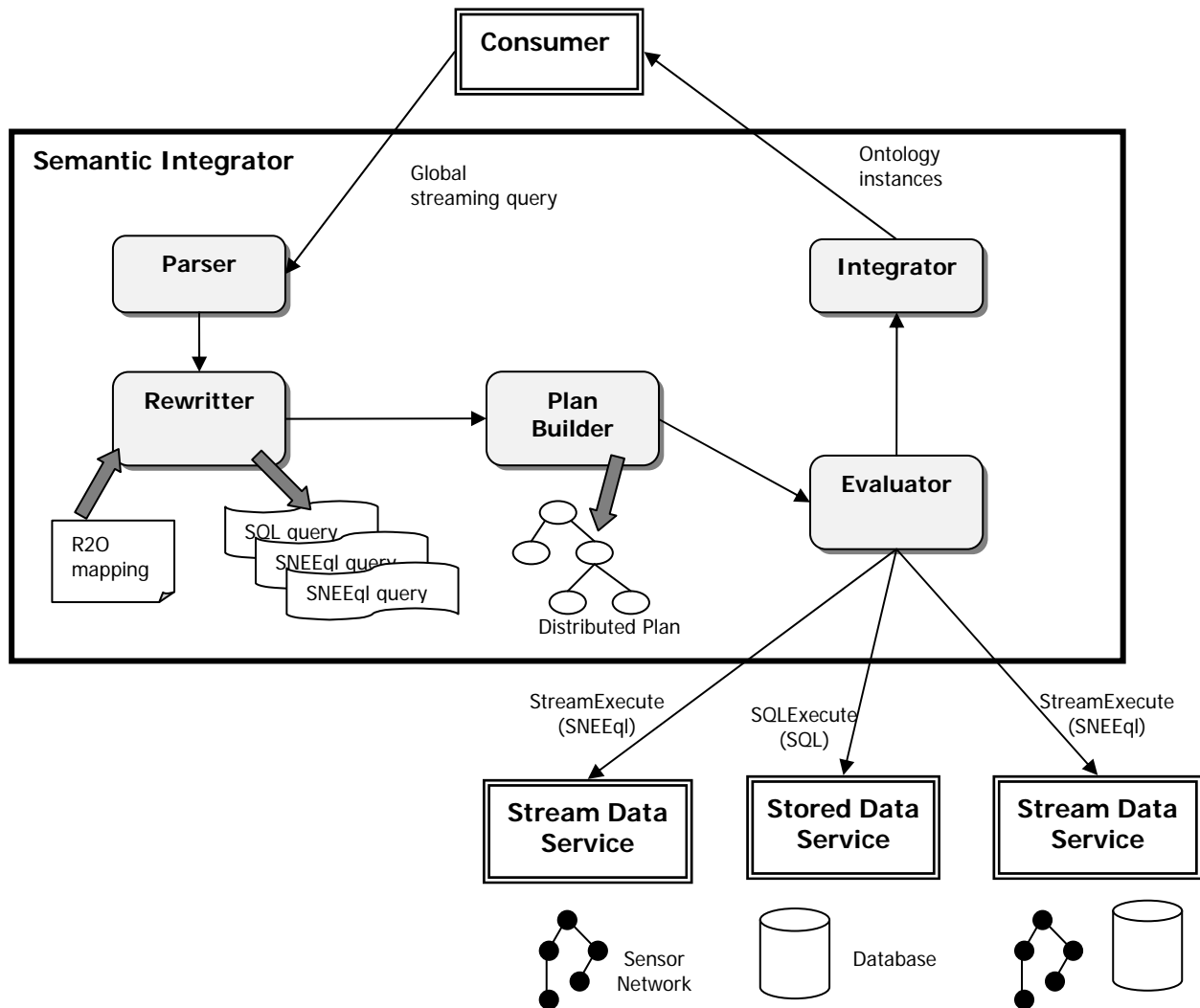


Figure 5.3: Semantic Integrator modules.

Parser

The Parser takes the global streaming query expression, parses and transforms it into an internal representation tree. The validation during the parsing process will check for syntactic correctness and coherence of the expression. In case of failure the process will be stopped and an invalid expression fault will be raised.

The engine is capable of processing Conjunctive Queries of the form [Bar06]:

$$Q(x_1, \dots, x_k): \bigwedge_{i=1 \dots k} P_i(x_1, \dots, x_n, y_1, \dots, y_m, c_1 \dots c_l)$$

Where Q is the query and the x_i terms are the query variables. The query answer will be an instantiation of those variables. The P_i terms can be either concepts, attributes or relations of the global ontology for which explicitations exist in the mapping document. The y_i terms are existentially qualified variables that restrict the x_i variables. The c_i terms are constants used to specify concrete values of attributes or individuals.

Rewriter

Then the Rewriter will translate this representation into queries for the different data sources using the R₂O mapping document. The conceptmap-definition elements of the R₂O document will provide the necessary information for the rewriter to generate the set of SNEEqL queries that will be sent to the different data sources. This includes the following steps.

1. The global streaming query is analyzed in order to determine if it can be rewritten, by calculating an equivalent maximum query. This process is described in [Bar06].
2. The explicitation of the equivalent maximum query is generated, by expanding all its conjunctive elements.
3. The explicitation obtained in the previous step is transformed to an equivalent relational or streaming algebra expression and then to the implementation, i.e. SQL or SNEEqL.
4. Non-delegable expressions like transformations on attribute values are implementation-specific and may have a different syntax or name depending on the vendor. These operators (e.g. concat, sub-string, etc.) are generated in this last step.

During this phase the processor will also use the semantic information of the ontology to enrich the query. For instance it may use the subsumption assertions to automatically determine the inclusion of a selection clause in the query to find not only instances of a given concept but also instances of sub-concepts. Other semantic reasoning can be applied to semantically enhance the query results. In this stage, Quality of Service parameters must also be sent to the query execution engine (SNEE) so that it can perform the expected optimizations. The rewriter will also have to generate the necessary queries that actually integrate the distributed sources, through joins or other operators.

For instance in the query presented in Section 5.1.2 and considering the mapping in Figure 5.2, the resulting query would be a SNEEqL expression:

```
SELECT speed, direction FROM WindMeasurement [FROM NOW TO NOW-10 MINUTES]
```

Plan Builder

The Plan Builder will then proceed to generate a distributed query plan, including the set of queries to execute, and any post processing operations to be executed after getting the results from each source. Then it will be in charge of splitting the plan for every data source involved. It will specify also the communication exchange operators between the execution nodes if needed.

Evaluator

Afterwards the Evaluator will be in charge of executing the individual queries by invoking the respective data sources and will get the results back, either continuously or not depending on the nature of the query. The Evaluator must contact the Stored or Stream Data Services using the appropriate interface operations. Typically for a stored data services it will call the ExecuteSQL operation passing the SQL expression generated by the Rewriter as a query parameter. Otherwise if it is a Stream Data Service the StreamExecute operation will typically be invoked, with a SNEEqL query as parameter.

Integrator

Finally the Integrator will be in charge of receiving the results from the evaluator and performing the post processing integration tasks of information, unions and joins according to the execution plan. It will be in charge of generating the resulting RDF tuples of the Response Document. Notice that continuous query results can also be generated and accessed indirectly as described in the Query Interface (see Section 2.2.2).

The Plan Builder, Evaluator and Integrator components will for the distributed query processor. This query processor may be built based on an extension of the SNEE processor, taking advantage of the already available capabilities such as joining streaming and stored data, and upcoming functionality such as query capabilities over non-acquisitional streams, middleware SNEE processing, among others.

5.2 Service Implementation

As we have seen in Section 2.2, WP4 is focused on the Semantic Integrator Service and its interfaces. In the previous section we have broadly described how the challenges of the semantic integration will be tackled. In this section we provide a sketch design of the implementation of the interfaces based on the existing technologies and the extensions and proposals of the previous section.

5.2.1 Integration Interface Implementation

The IntegrateAs operation of the Integration Interface, as seen in Section 2.2.1, receives as input a set of source names to integrate and a mapping document. The source names are standard URIs

and will be stored and managed in the SemSorGrid4Env Semantic Registry [KKK09]. These sources can be either Stored Data services with SQL endpoints or Stream Data services with SNEEq endpoints. If the resource name is invalid, an `InvalidResourceNameFault` will be thrown. If the data resource is not available, a `DataResourceUnavailableFault` will be thrown.

The mapping document will be a R₂O document extended as described in Section 5.1.1. These extensions are mainly the multiple data source support [Pri09], and stream data source support (Section 5.1.1).

The Integration implementation will then validate the R₂O document and retrieve the sources metadata, including QoS properties.

The validation of the extended R₂O document includes the following steps:

1. Verification of references to ontology, stored and stream data elements. The elements in the `dbschema-desc` and `streamschema-desc` must correspond to existent, valid and appropriate element references.
2. Verification of attribute, concepts and relations explicitations. In the `conceptmap-def` section all explicitation references must point to existent concepts, relations and attributes.
3. Verification of types. Operation types specified in the mapping must correspond to the declared ontology attribute types.

A validation anomaly will produce an `InvalidMappingDocumentFault`.

Finally if no faults are thrown, the virtual source reference will be produced and sent as a result.

5.2.2 Query Interface Implementation

The Query Interface defined in Section 2.2.2 defines the query operations over stored or stream data sources. The query operation `<GenericQuery>` returns the results directly in the response message. This generic operation can be realised by a concrete one, depending on the case. For queries over Stored Data, the `SQLExecute` operation [ACK+06] will be used and for querying Stream Data the `StreamExecute` operation [GGFP09] will be used. In this section we will describe the case of an integrated data source created by the `IntegrateAs` operation.

The `<GenericQuery>` receives the data resource name, a dataset format URI and the query expression. The expression is an extended streaming SPARQL query as indicated in Section 5.1.2. This query is written in terms of the global ontology referenced by the mapping document used in the `IntegrateAs` operation. The query execution process will follow the path described in Section 5.1.3.

- If the data resource name is not found or unavailable, a `DataResourceUnavailableFault` will be raised. If the data resource name is not valid, an `InvalidResourceNameFault` will be raised.

- Upon receiving the query, the query processor *Parser* validates its correctness against the global ontology referenced by the R₂O mapping document, checking also its compliance to the global streaming language syntax. If the expression has errors, the `InvalidExpressionFault` will be raised.
- Then the *Rewriter* uses the R₂O mapping information to transform the global streaming language queries into either SQL or SNEEqL queries. The rewriting follows four stages as shown in Section 5.1.3.
- After the queries have been generated for the data sources, the *Plan Builder* will produce a plan that will indicate which queries will be executed in which source, and the post-processing operators that must be used to integrate the incoming results.
- The *Evaluator* will then send the queries to the Stored Data Services or Stream Data Services, fronted by SQL or SNEEqL endpoints. For a SNEEqL endpoint, the `StreamExecute` operation will be called and for the SQL endpoint the `SQLExecute` operation will be called instead.
- The results of these queries will be received by the *Integrator* module that will perform the operations specified by the plan and later produce a `ResponseDocument` with the ontology instances that satisfy the query.

Generic Query Factory

In the case of most continuous queries, the result of the query execution cannot just be thrown as a one-shot `ResponseDocument` result. This would only be possible for queries asking for finite result-sets such as relations or one-shot windows. Otherwise the `<GenericQueryFactory>` operation must be used. For streams the concrete `StreamExecuteFactory` operation will be called. As in the `<GenericQuery>`, it receives the resource name and the query expression. In addition it receives a `configurationDocument` parameter. This configuration XML document contains the parameters that indicate how the response will be generated and accessed, e.g. using a pull or push-based mechanism, the interface type, etc.

For a pull-based retrieval of data, the execution will follow similar steps to those described above, but instead of generating a resulting `ResponseDocument` it will generate a reference to a data source where the results can be obtained. In a pull-based schema the data can be recovered using Data Access Interface operations [GGF+09a], namely concrete implementations of the generic `<GetResponseItem>`. For example, the `GetStreamNewestItem` retrieves a specified number (or time-based) of items from the result set.

5.3 Future Work

As it has been described there are still many open points that need to be answered, solved and implemented. These are discussed below.

5.3.1 R₂O Extensions for Streams

The R₂O extensions for streaming data need to be added, basically for stream data declaration, following the discussions of Section 5.1.1. This includes:

- The necessary additions to define stream sources, with all the metadata that the query rewriter will need to generate the SQL or SNEEqL queries. This information includes the stream schema, key and non-key attributes, timestamp attributes, tuple generation information, etc.
- The language concept-mapping elements must also be modified so that they support stream element just like they currently do with relation elements.
- These additions must be formalized in the language specification so that the processor (i.e. extended ODEMapster) will be able to interpret and use them. The syntax must be defined and written in the XML schema.
- Validate the language extensions against the requirements.

This task will end with the consolidation of a formal mapping language capable of expressing stream-to-ontology mappings, or stream-and-relation-to-ontology mappings. This will enable providing a unified ontological view for the different streaming and stored schemas.

5.3.2 SPARQL Support

The ODEMapster engine must be modified to fully support SPARQL queries. The ODEMapster engine will evolve to constitute the Semantic Integrator translator processor. This task includes:

- Analyzing the current ODEMQ L implementation and the possibility of porting this code for SPARQL support. Currently the ODEMapster processor works over ODEMQ L queries. Although it has been proven that it can perfectly work over SPARQL for conjunctive queries [Fus08],
- Modifying the ODEMapster processor for SPARQL support.
- The processor must be able to parse the language and produce the SQL queries based on the R₂O mapping document, as it is described in Section 4.2.2.
- Then this implementation will be evaluated and validated against the previous ODEMQ L version.

5.3.3 Support for R₂O Extensions

The Semantic Integrator engine must be modified to support the extended R₂O language for stream-to-ontology mappings. This task includes:

- Modifying the processor to read the new constructs that will be available in the extended R₂O, mainly the stream definitions including the attributes, timestamps and other metadata.

- The processor will be modified to be able to connect to a stream data source, in this case SNEE.
- The processor will be modified so that the pre-processing phase the validation takes into account the R₂O extensions.
- The implementation will focus on the case of the SNEEqL query language, and it will be used to validate this new support.

This task depends on the formalization of the R₂O extensions.

5.3.4 Extended SPARQL for Streams

The global streaming SPARQL extensions must be formally defined and the parser must be implemented. In Section 5.1.2 we mentioned the expected language features that we plan to support based on our requirements. This task will include:

- Evaluating the semantics of previous languages (see Section 4.2.4).
- Proposing a set of required features for the language.
- Proposing a formal syntax for the proposed language.
- The development of a parser for the new language.
- The specification of the semantics of this language.
- Provide a comparison and guidelines for mapping this language into SNEEqL.
- Evaluate and validate the query language.

5.3.5 Query Translation

The Semantic Integrator engine must be modified to support the Streaming SPARQL queries and be able to transform them into SNEEqL. This task includes:

- Modifying the Semantic integrator processor so that the translator phase is able to convert the global streaming query language into queries in SNEEqL.
- Implementation of translation of stream attributes, time windows, slide parameters, tuple-based windows, aggregates, joins between streams, joins between streams and relations, streaming operators.
- Evaluation and validation of the produced queries.

The processor will have to be already modified in order to understand the new stream extensions of the R₂O language. The processor will already be able to transform classic SPARQL queries. The global streaming query language based on SPARQL extensions will need to be already well defined.

5.3.6 Distributed Query Processing

The distributed query processor of the Semantic Integrator must be designed and built as an extension of the SNEE processor. This task includes:

- Design and implementation of modifications to SNEE. SNEE is capable of joining streaming and stored data, processing acquisitional data streams. It must be modified to perform distributed query execution.
- It must be extended to support the distributed scenario, with multiple sources. The processor must be able to produce a distributed query plan.
- It must evaluate the query fragments over the distributed sources.
- It must integrate the different results according to the plan.

The implementation will start as a SNEE processor extension, in collaboration with WP2. The Semantic Integrator processor will have to be already modified to support translation of the global streaming query language to SNEEqL,

5.3.7 QoS Optimisations

QoS parameters must be defined and included in the metadata so they can later be used for optimization purposes. A QoS model must be defined so that it is suitable to represent the desired information. This model must be understandable by the translator and query processor of the Semantic Integrator, so that this information can be used for optimization purposes. The query optimization oriented QoS parameters already considered in SNEEqL can be taken as a starting point. Other QoS expectation parameters can be analyzed for the future.

6 Bibliography

- [AAB+05] Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Çetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag S. Maskey, Alexander Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stan Zdonik. The design of the Borealis stream processing engine. In *Proceedings of 2nd Biennial Conference on Innovative Data Systems Research (CIDR'05)*, Asilomar, CA, USA, January 2005.
- [ABB+04] Arvind Arasu, Brian Babcock, Shivnath Babu, John Cieslewicz, Mayur Datar, Keith Ito, Rajeev Motwani, Utkarsh Srivastava, and Jennifer Widom. STREAM: The Stanford Data Stream Management System. In *Data Stream Management: Processing High-Speed Data Streams*. 2004. Springer-Verlag.
- [ABW06] Arvind Arasu and Shivnath Babu and Jennifer Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *The VLDB Journal - The International Journal on Very Large Data Bases* 15(2), pages: 121 – 142. 2006. Springer-Verlag New York, Inc.
- [ACC+03] Daniel J. Abadi, Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, Stan Zdonik. Aurora: a new model and architecture for data stream management. *The VLDB Journal – The International Journal on Very Large Data Bases* 12(2), pages 120-139, August 2003. Springer-Verlag New York, Inc.
- [ACH+93] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, 2(2), pages 127–158, 1993.
- [ACK+06] Mario Antonioletti, Brian Collins, Amy Krause, Simon Laws, James Magowan, Susan Malaika, and Norman W. Paton. Web services data access and integration - the relational realisation (WS-DAIR) specification, version 1.0. Recommendation GFD.76, Open Grid Forum, 20 July 2006.
<http://www.ogf.org/documents/GFD.76.pdf>.
- [ALM+04] Daniel J. Abadi, Wolfgang Lindner, Samuel Madden, Jörg Schuler. An Integration Framework for Sensor Networks and Data Stream Management Systems. In *Proc. VLDB*, 2004.

- [AMG+04] M. Nedim Alpdemir, Arijit Mukherjee, Anastasios Gounaris, Norman W. Paton, Paul Watson, Alvaro A.A. Fernandes, Jim Smith. OGSA-DQP: A Service-Based Distributed Query Processor for the Grid. In *Proc. EDBT 2004, volume 2992 of LNCS*. 2004
- [Bar06] Jesús Barrasa. Modelo para la definición automática de correspondencias semánticas entre ontologías y modelos relacionales. PhD Thesis. Departamento de Inteligencia Artificial. Facultad de Informática. UPM. December 2006.
- [BBD+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 1-16, Madison, Wisconsin, USA, June 2002.
- [BBC+09] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, Michael Grossniklaus. C-SPARQL: SPARQL for Continuous Querying. In *Proceedings of the 18th international conference on World wide web*, pages 1061-1062, 2009.
- [BBV+07] Domenico Beneventano, Sonia Bergamaschi, Maurizio Vincini, Mirko Orsini, Carlos Nana Rodriguez. Query Translation on Heterogeneous Sources in MOMIS Data Transformation Systems. *InterDB Third International Workshop on Database Interoperability*, 2007.
- [BC07] Christian Bizer, Richard Cyganiak. D2RQ — Lessons Learned. Position paper for the W3C Workshop on RDF Access to Relational Databases, Cambridge, USA, 25-26 October 2007. <http://www.w3.org/2007/03/RdfRDB/papers/d2rq-positionpaper/>
- [BCG05] Jesús Barrasa, Óscar Corcho, Asunción Gómez-Pérez. R2O, an Extensible and Semantically Based Database-to-ontology Mapping Language. In: *Bussler, C., Tannen, V., Fundulaki, I. (eds.) SWDB 2004 LNCS*, vol. 3372, 2005. Springer.
- [BG06] Jesús Barrasa, Asunción Gómez-Pérez. Upgrading Relational Legacy Data to the Semantic Web. In *Proceedings of the 15th international conference on World Wide Web*, pages 1069 – 1070, 2006.
- [BGF+08] Christian Y.A. Brenninkmeijer, Ixent Galpin, Alvaro A.A. Fernandes, and Norman W. Paton. A Semantics for a Query Language over Sensors, Streams and Relations. In *Proceedings of the 25th British national conference on Databases: Sharing Data, Information and Knowledge*, pages 87 – 99, Cardiff, Wales, UK. 2008. Springer-Verlag.
- [BGJ08] Andre Bolles, Marco Grawunder and Jonas Jacobi. Streaming SPARQL - Extending SPARQL to process data streams. *The Semantic Web: Research and Applications*, pages 448-462, 2008.

- [BvHH+04] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. OWL web ontology language reference. Recommendation, W3C, 10 February 2004. <http://www.w3.org/TR/owl-ref/>.
- [CCC+02] Don Carney, Uğur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, Stan Zdonik. Monitoring Streams - A New Class of Data Management Applications. In *Proceedings of the 28th International Conference on Very Large Data Bases*, pages 215-226, Hong Kong, China, August 2002. VLDB Endowment.
- [CCD+03] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Sam Madden, Vijayshankar Raman, Fred Reiss, and Mehul Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proceedings of the 1st Conference on Innovative Data Systems Research*, pages 269-280, Asilomar, CA, USA, January 2003.
- [CCDG+03] Andrea Cali, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Paolo Naggari and Fabio Vernacotola. IBIS: Semantic Data Integration at Work Contact Information. *Advanced Information Systems Engineering*, page 1033. January 2003. Springer.
- [CCR+03] Don Carney, Uğur Çetintemel, Alex Rasin, Stan Zdonik, Mitch Cherniack, Mike Stonebraker. Operator Scheduling in a Data Stream Manager. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 838-849, Berlin, Germany, September 2003. VLDB Endowment.
- [CDL+98] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, Riccardo Rosati. Description Logic Framework for Information Integration. In *Proceedings of the 6th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'98)*, pages 2-13, 1998.
- [CDL+03] Andrea Cali, Saverio De Nigris, Domenico Lembo, Gabriele Messineo, Riccardo Rosati, Marco Ruzzi. DIS@DIS: a system for semantic data integration under integrity constraints. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, pages 335 – 338, December 2003.
- [CDT+00] Jianjun Chen, David J. DeWitt, Feng Tian, Yuan Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In *ACM SIGMOD Record Volume 29, Issue 2*, pages 379 – 390, June 2000.
- [Cer08] Farid Cerbah. Learning Highly Structured Semantic Repositories from Relational Databases The RDBToOnto Tool. In *Proceedings of the 5th European Semantic Web Conference (ESWC 2008)*, Tenerife, Spain, June, 2008.



- [CHSR09] Mike Clark, Craig Hutton, Jason Sadler, and Samantha Roe. Wp 7.1 Flood user requirements specification. Deliverable D7.1, SemSorGrid4Env, March 2009.
- [CJS+02] Chuck Cranor, Theodore Johnson, Oliver Spatschnek, Vladislav Shkapenyuk. Gigascope: A Stream Database for Network Applications. In *Proc. ACM SIGMOD*, page 262, 2002.
- [DCB+09] Emanuele Della Valle, Stefano Ceri, Daniele Braga, Irene Celino, Dieter Frensel, Frank van Harmelem, Gulay Unel. Research Chapters in the Area of Stream Reasoning. In *Proceedings of the 1st International Workshop on Stream Reasoning*, 2009.
- [DH05] AnHai Doan, Alon Y. Halevy. Semantic Integration Research in the Database Community: A Brief Survey. *AI Magazine Volume 26, Issue 1 (March 2005)*, pages 83 – 94, 2005.
- [DLS05] Glen Dobson, Russell Lock, Ian Sommerville. Quality of Service Requirements Specification Using an Ontology. In *Proc. Service-Oriented Computing: Consequences for Engineering Requirements (SOCCER 05) at 13th International Requirements Engineering Conference (RE 05)*. 2005
- [EM07] Orri Erling, Ivan Mikhailov. RDF support in the virtuoso dbms. In *Proceedings of the 1st Conference on Social Semantic Web (CSSW)*, volume 113 of LNI, pages 59–68, Leipzig, Germany, September 2007. Gesellschaft Für Informatik.
- [FLM99] Marc Friedman, Alon Levy, Todd Millstein. Navigational Plans For Data Integration (1999). In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1999.
- [Fus08] Francesco Fusco. SPARQL for Ontology-based data access. Master thesis. Department of Computer Science. Università degli Studi di Roma *La Sapienza*. 2008
- [GBJ+08] Ixent Galpin, Christian Y.A. Brenninkmeijer, Farhana Jabeen, Alvaro A.A. Fernandes, Norman W. Paton. An Architecture for Query Optimization in Sensor Networks. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 1439-1441, Cancún, México, April 2008. IEEE.
- [GBJ+09] Ixent Galpin, Christian Y.A. Brenninkmeijer, Farhana Jabeen, Alvaro A.A. Fernandes, and Norman W. Paton Comprehensive Optimization of Declarative Sensor Network Queries. In *Proceedings of the 21st International Conference on Scientific and Statistical Database Management*, pages 339 – 360, New Orleans, LA, USA, June 2009. Springer.
- [GGF+09] Alasdair J. G. Gray, Ixent Galpin, Alvaro A. A. Fernandes, Norman W. Paton, Kevin Page, Jason Sadler, Manolis Koubarakis, Kostis Kyzirakos, Jean-Paul Calbimonte, Oscar Corcho, Raúl García, Víctor Manuel Díaz, and Israel Liebana.

SemSorGrid4Env architecture phase I. Deliverable. D1.3v1, SemSorGrid4Env, August 2009.

- [GGF+09a] Ixent Galpin, Alasdair J G Gray, Alvaro A A Fernandes, Norman W. Paton, Alexis Kotsifakos, Dimitris Kotsakos, and Dimitrios Gunopulos. Data Requirements, Data Management and Analysis Issues, and Query-Based Functionalities. Deliverable D2.1
- [GGFP09] Alasdair J. G. Gray, Ixent Galpin, Alvaro A. A. Fernandes, and Norman W. Paton. Web services data access and integration - the data stream realisation (WS-DAIStreaming) specification. Technical report, University of Manchester, August 2009.
- [GGK+07] Sven Groppe, Jinghua Groppe, Dirk Kukulenz and Volker Linnemann. A SPARQL Engine for Streaming RDF Data. In *Proceedings of the 2007 Third International IEEE Conference on Signal-Image Technologies and Internet-Based System*, pages 167-174. 2007.
- [GHI+97] Héctor Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information System*, 1997.
- [GHR06] Martin Gudgin, Marc Hadley, and Tony Rogers (Eds). Web services addressing 1.0 core. Recommendation, W3C, 9 May 2006. <http://www.w3.org/TR/ws-addr-core/>.
- [GKD97] Michael R. Genesereth, Arthur M. Keller, Oliver M. Duschka. Infomaster: An Information Integration System. In *Proceedings of 1997 ACM SIGMOD Conference*, pages 39—542, 1997.
- [GLR00] François Goasdoué, Véronique Lattes, Marie-Christine Rousset. The Use of CARIN Language and Algorithms for Information Integration: The PICSEL Project. *International Journal of Cooperative Information Systems (IJCIS)* 9(4), pages 383–401, 2000.
- [Hal01] Alon Y. Halevy. Answering queries using views: A survey. In *VLDB Journal*, 10(4), pages 270–294, December 2001.
- [HFL+89] Laura Haas, Johann Christoph Freytag, Guy Lohman, and Hamid Pirahesh. Extensible query processing in starburst. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 377–388, Portland, OR, USA. 1989.
- [HJK+93]. Michael N. Huhns, Nigel Jacobs, Tomasz Ksiezzyk, Wei-Min Shen, Munindar P. Singh, and Philip E. Cannata. Integrating enterprise information models in Carnot. In *International Conference on Intelligent and Cooperative Information Systems (CoopIS)*, May 1993.



- [HLS+08] Peter Haase, Holger Lewen, Rudi Studer, and Mathieu d'Aquin. The NeOn Ontology Engineering Toolkit. In *WWW2008 - The 17th International World Wide Web Conference - Developers' Track*, Beijing, China. 2008.
- [IKK05] Ismail Khalil Ibrahim, Reinhard Kronsteiner, Gabriele Kotsis. A semantic solution for data integration in mixed sensor networks. *Computer Communications* 28(13), pages 1564-1574, 2005.
- [JBM08] Lei Jiang, Alex Borgida, and John Mylopoulos. Towards a Compositional Semantic Account of Data Quality Attributes. In *Proceedings of the 27th International Conference on Conceptual Modeling*, Barcelona, Spain, pages 55 – 68, 2008. Springer-Verlag.
- [JMS+08] Namit Jain, Shailendra Mishra, Anand Srinivasan, Johannes Gehrke, Jennifer Widom, Hari Balakrishnan, Mitch Cherniack, Ugur Cetintemel, Richard Tibbetts, and Stan Zdonik. Towards a Streaming SQL Standard. *Proceedings of the 34th International Conference on Very Large Data Bases (industrial track)*, Auckland, New Zealand, August 2008.
- [KKK09] Kostis Kyzirakos, Manolis Koubarakis, and Zoi Kaoudi. Data models and languages for registries in SemsorGrid4Env. Deliverable D3.1 Version 1.0, SemSorGrid4Env, August 2009.
- [Kos00] Donald Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys Volume 32, Issue 4*, pages 422-469, December 2000.
- [KP06] Kyriakos Kritikos and Dimitris Plexousakis. Semantic QoS Metric Matching. In *Proceedings of the European Conference on Web Services*, pages 265-274, December 2006.
- [KP07] Kyriakos Kritikos and Dimitris Plexousakis. OWL-Q for Semantic QoS-based Web Service Description and Discovery. In *Proceedings of the SMR 2007 Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMRR 2007) co-located with ISWC 2007 + ASWC 2007*, Busan, South Korea, November 11, 2007.
- [KW07] Hiroyuki Kitagawa, Yousuke Watanabe. Stream Data Management Based on Integration of a Stream Processing Engine and Databases. In *Proceedings of the 2007 IFIP International Conference on Network and Parallel Computing Workshops*, pages 18-22, 2007.
- [Len02] Maurizio Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233 – 246, Madison, Wisconsin, 2002. ACM.
- [Lev98] Alon Y. Levy. The Information Manifold Approach to Data Integration. In: *IEEE Intelligent Systems*, 13, pages 12-16, August 19 1998.

- [LDI09] Israel Liébana, Víctor Manuel Díaz, and Agustín Izquierdo. Fire risk monitoring and warning in Castilla y León - requirements specification. Deliverable D6.1, SemSorGrid4Env, March 2009.
- [LMH+09] Steven Lynden, Arijit Mukherjee, Alastair C. Hume, Alvaro A. A. Fernandes, Norman W. Paton, Rizos Sakellariou, and Paul Watson. The design and implementation of OGSA-DQP: A service-based distributed query processor. *Future Generation Computer Systems*, 25(3), pages 224–236, March 2009.
- [LT09] Lina Lubyte and Sergio Tessaris. Supporting the Development of Data Wrapping Ontologies. Free University of Bozen-Bolzano, 2009.
- [MBR01] Jayant Madhavan, Philip Bernstein, Erhard Rahm. Generic Schema Matching with Cupid. In *Proceedings of the 27th VLDB Conference*. Roma, Italy, 2001.
- [MFH+05] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: An Acquisitional Query Processing System for Sensor Networks. In *ACM Trans.Database Syst.*, 30(1), pages 122-173, 2005.
- [MIK+00] Eduardo Mena, Arantza Illarramendi, Vipul Kashyap, Amit Sheth. OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies. *International Journal on Distributed and Parallel DBs* 8(2), pages 223–271, 2000.
- [MM04] Frank Manola and Eric Miller (eds). RDF primer. Recommendation, W3C, 10 February 2004. <http://www.w3.org/TR/rdf-primer/>.
- [NLF99] Felix Naumann, Ulf Leser, Johann Christoph Freytag. Quality-driven Integration of Heterogeneous Information Systems. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages: 447 - 458, Edinburgh, Scotland, UK, September 1999. Morgan Kaufmann Publishers Inc.
- [OGS09] OGSA-DAI open grid services architecture database access and integration services project, 2005-2009. <http://www.ogsadai.org.uk/> Accessed 3 August 2009.
- [PC06] Cristian Pérez de Laborda and Stefan Conrad. Database to Semantic Web Mapping using RDF Query Languages. In *Conceptual Modeling - ER 2006, 25th International Conference on Conceptual Modeling*, pages 241-254, Tucson, Arizona, November 2006.
- [PLC+08] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking Data to Ontologies. In *J. on Data Semantics. Volume X*, pages 133-173, 2008.
- [Pri09] Freddy Priyatna. Multi-database ontology-based mapping and data access. Master Thesis. Department of Artificial Intelligence. School of Computer Science. Universidad Politécnica de Madrid. 2009.



- [PRR08] Antonella Poggi, Mariano Rodriguez-Muro, Marco Ruzzi. Ontology-based database access with DIG-MASTRO and the OBDA Plugin for Protégé. *In Proc. of the 4th Int. Work. on OWL: Experiences and Directions (OWLED 2008 DC)*, 2008.
- [Pru07] Eric Prud'hommeaux, SPASQL <http://www.w3.org/2005/05/22-SPARQL-MySQL/XTech.W3C.June.2007>.
- [PS08] Eric Prud'hommeaux and Andy Seaborne (eds). SPARQL query language for RDF. Recommendation, W3C, 15 January 2008. <http://www.w3.org/TR/rdf-sparql-query/>.
- [RAY+00] Naphtali Rishe, Rukshan I. Athauda, Jun Yuan, Shu-Ching Chen. Semantic Relations: The key to integrating and query processing in heterogeneous databases. In *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, Vol. 7, Computer Science and Engineering: Part I*, pages 717-722, 2000.
- [RDS+04] Elke A. Rundensteiner, Luping Ding, Timothy Sutherland, Yali Zhu, Brad Pielech, Nishant Mehta. CAPE: continuous query engine with heterogeneous-grained adaptivity. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, pages 1353 – 1356, Toronto, Canada, 2004.
- [RS98] Raghu Ramakrishnan and Avi Silberschatz. Scalable Integration of Data Collection on the Web. Technical report, University of Wisconsin-Madison, 1998.
- [SHH+08] Satya S. Sahoo, Wolfgang Halb, Sebastian Hellmann, Kingsley Idehen, Ted Thibodeau Jr, Sören Auer, Juan Sequeda, Ahmed Ezzat. A Survey of Current Approaches for Mapping of Relational Databases to RDF. W3C RDB2RDF Incubator Group, January 8, 2009. http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf
- [SLJ+05] Timothy M. Sutherland, Bin Liu, Mariana Jbantova, Elke A. Rundensteiner. D-CAPE: distributed and self-tuned continuous query processing. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, Bremen, pages: 217 – 218, Germany, 2005.
- [SQL92] Information technolog database language SQL. Standard ISO/IEC 9075:1992, ISO, 1992.
- [SSW07] Andy Seaborne, Damian Steer, Stuart Williams. SQL-RDF. In *W3C Workshop on RDF Access to Relational Databases*, Cambridge, MA, USA, October 2007.
- [Sul96] Mark Sullivan. Tribeca: A Stream Database Manager for Network Traffic Analysis. In *Proceedings of the 22nd International Conference on Very Large Data Bases*, page 594, San Francisco, CA, USA, September 1996. Morgan Kaufmann Publishers Inc.



- [TGN+92] Douglas Terry, David Goldberg, David Nichols and Brian Oki. Continuous Queries over Append-Only Databases. In *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, pages 321-330, San Diego, CA, USA, June 1992. ACM.
- [YG02] Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. In *SIGMOD Rec.*, 31(3), pages 9-18, 2002.
- [YKB99] Haiwei Ye, Brigitte Kerhervé, Gregor v. Bochmann. QoS-Aware Distributed Query Processing. In *Proceedings of the 10th International Workshop on Database & Expert Systems Applications*, page: 923, Florence, Italy, September 1999.
- [Wie92] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3), pages 38–49, March 1992.
- [WVV+01] Holger Wache, Thomas Vögele, Ubbo Visser, Heiner Stuckenschmidt, Gerhard Schuster, Holger Neumann and Sebastian Hübner. Ontology-Based Integration of Information - A Survey of Existing Approaches. In *IJCAI--01 Workshop: Ontologies and Information Sharing (2001)*, pages 108-117. 2001.

7 Annexes

Linked Stream Data: A Position Paper

Juan F. Sequeda¹, Oscar Corcho²

¹ Department of Computer Sciences. University of Texas at Austin

² Ontology Engineering Group. Departamento de Inteligencia Artificial. Universidad Politécnica de Madrid, Spain

jsequeda@cs.utexas.edu, ocorcho@fi.upm.es

Abstract. The amount of sensors publishing data on the Web is increasing as a result of the online availability of Sensor Web platforms that provide support for this task. With such increase in sensor data publication, new challenges arise for the identification, discovery and access to this data. Following the set of best practices to publish and link structured data on the web proposed by the Linked Data community, in this paper we introduce the concept of Linked Stream Data, a way in which the Linked Data principles can be applied to stream data and be part of the Web of Linked Data.

1 Introduction

The cost of deploying sensor networks has been falling in the last years, while their capacity has been increasing steadily. As a result, more sensor networks are being deployed in many different environments (roads, forests, agricultural lands, people, homes, etc.), and the information coming from these sensor networks is being used more often for better situation assessment and decision making.

The amount of information being generated by deployed sensor networks is extremely large. For example temperature sensors can emit their readings every 30 min, while heart rate sensors can send their data to a repository every minute. Having this data available not only internally to legacy applications but also available on the web will provide a new source of knowledge for scientists, decision-makers and other types of users. We can then talk about the worldwide sensor web [1].

However, the availability of this data on the web poses new challenges related to how this data can be discovered, identified and exploited in a range of applications. In other words, there needs to be a way to identify it and describe it consistently and to access it easily.

We believe that there is a good opportunity to apply to sensor data the same principles that have been used for the publication of other types of (more static) data on the (Semantic) Web, in the

context of the Linked Data initiative. Basically, Linked Data refers to a set of best practices to be followed in order to publish and link data on the Web, using the following basic principles:

- Use URIs as names for things.
- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information, using appropriate standards (RDF, SPARQL)
- Include links to other URIs, so that more things can be discovered.

In the application of these principles, there are several features that clearly differentiate the type of data that is normally published in the Linked Data world from data that is originated from sensor networks. First, sensor-based data is normally related to physical measurements and observations, hence predominantly numerical. Second, time and space considerations have to be addressed: sensors and sensor networks are located in specific geographical positions and these positions are usually important for decision making and for further processing, and the measurements provided are commonly tagged with timestamps (what allows for fragments of sensor-based data to be identified by time windows). Third, data accuracy and uncertainty is another factor to be taken into account. Although these are characteristics of most types of sensor-based data, this is not exclusive from this type of data and could also be the case for other types of data.

In summary, the objective of this paper is to discuss how to apply linked data principles to sensor-based data, in what we call "Linked Stream Data" (LSD). We propose a URI-based mechanism to identify and access Stream Data coming from sensor networks, detailing the main requirements for their creation and the main reasons for taking decisions on their design. Finally, we discuss the open challenges for future research.

2. A Selection of Use Cases

In this section we present a series of use cases where information coming from sensors is used and which may benefit from the availability of this information as linked data.

Linear Road Benchmark [2]. It is a well established benchmark for Data Stream Management Systems. This benchmark specifies a variable tolling system by determining changing factors of car congestion on a highway. Each car on the highway is equipped with a sensor that emits the vehicle's exact location and speed every 30 seconds. The data emitted by the sensors is sent as streams to a central system where statistics are generated about traffic conditions on the highways. This tolling system is designed to discourage drivers to use already congested roads because they have an increased toll. Consequently, it would encourage drivers to use less congested roads because they would have decreased tolls. Although this use case was created in order to test and compare different characteristics of existing data stream management systems, the domain in which it is applied is one that may clearly benefit from its availability as Linked Data.

Heart Sensors. Patients with heart problems can have sensors that monitor their heart rate and current location. The data emitted by the heart sensor can be sent as stream data to the patient's hospital where it is monitored. The hospital can detect if the patient suffers from any heart

abnormality in real-time. Furthermore, if a patient is having a heart attack, the hospital can immediately send an ambulance to the patient's exact location. Using common Web protocols for publishing and accessing this data, while preserving security and privacy, can ease the development of such type of emergency management applications.

Environmental Sensors. Environmental researchers need to track a large number of aspects of specific regions. For example, one specific application may be monitoring the temperature and humidity of a region. Usually static sensors are used in this type of applications, because they are monitoring fixed areas, therefore space is not an issue in these types of sensors. However, the accuracy of the measurements made may be relevant.

3. Requirements

In this section we present some of the requirements that can be extracted from the previous use cases and that we consider that need to be satisfied by our proposal for linked stream data.

3.1 Identification and Querying

Resources on the Web are commonly identified by means of URIs. As mentioned in the Linked Data principles, URIs act as unique names (identifiers) for such resources on the web, but can also be extended to objects in the real world. As a consequence, we can propose the use of URIs to identify sensors that are deployed in the real world. Furthermore, URIs may also be used to identify data that is emitted by sensors.

The Linked Data principles also stipulate that these should be HTTP URIs and that once de-referenced, useful information should be provided. Therefore, by getting data back once a URI is de-referenced, the URI acts like a query interface or a RESTful service. Hence, the data returned from a sensor URI should be the metadata about the sensor and from the stream data URI should be the observations of the sensor.

From this discussion we can derive the following set of requirements that will inform our decisions for the creation of Linked Data Streams:

- Req 1: Sensors should be identified by URIs.
- Req 2: Stream Data emitted by sensors should be identified by URIs.
- Req 3: The information returned by a sensor URI should be its metadata.
- Req 4: The information returned by a stream data URI should be the observations of the sensor.

3.2 Time Dimension

Data streams provided by a specific sensor or group of sensors can be identified by a specific moment in time or by a time window. For example, consider a sensor that emits the heart rate of a person. One could identify the exact heart rate of a person at a specific time. Furthermore, one could also identify the series of heart rates emitted by that sensor in a specific window of time.

From this discussion we can derive the following set of requirements:

- Req 5: Stream data should be identifiable at specific moments in time.
- Req 6: Stream data should be identifiable in specific time windows.
- Req 7: Time used to describe time points or time windows should be expressed in a given unit of time (milliseconds, seconds, minutes, etc).

3.3 Space Dimension

Data Streams can also be identified given their spatial context. For example, consider a mobile sensor that emits the heart rate of a person. One could identify the exact heart rate of a person at a specific location. In the case of the linear road benchmark, we may be interested only in data coming from vehicles in a specific segment. Similar to the time dimension case, identifying space in sensors can be done by a specific location (or coordinate in this case), or by a bounding area (similar to the time windows but for space). A bounding area, given a center point, can be a radio, square or polygon.

- Req 8: Stream data should be identifiable at a specific location.
- Req 9: Stream data should be identifiable in a bounding area.
- Req 10: Bounding areas can be defined by a radio, square or polygon.

3.4 Combined Time and Space Dimensions

The notions of time and space are particularly interesting in the case of mobile sensors. In the case of the heart rate sensor, one could identify the exact heart rate of a person at a given time and location. In this case, we consider that the requirements for Time and Space should both be satisfied when identifying stream data from mobile sensors.

- Req 11: Stream data should be identifiable at a specific moment in time and specific location
- Req 12: Stream data should be identifiable in a time window and at a specific location.
- Req 13: Stream data should be identifiable at a specific moment in time and in a bounding area.
- Req 14: Stream data should be identifiable in a time window and in a bounding area.

4. A Proposal for Linked Stream Data

In the previous section, we have presented several requirements in order to identify sensors and stream data coming from sensors on the web. Per our motivation, we believe Stream Data can become part of the Web of Data by defining a URI-based mechanism, following the linked data principles, to identify sensors and stream data, optionally given their spatial and temporal context, which can be provided in many different forms. In this section, we present a proposal of human-friendly URIs¹ to identify sensors and stream data.

¹ It is important to note that the fact of using a human-friendly URI that encodes implicitly its semantics in the URI itself does not mean that the explicit semantics are necessarily provided for them. However, in our proposal we aim at combining human readability and the provision of explicit semantics when the URI is dereferenced. This last one is the only one that will allow automation of tasks based on semantics. The first one is only for human consumption and should be always considered like that by any application that uses them.

4.1 URI for Sensors

Sensors are real objects that can be identified by a URI. The data that is returned after de-referencing the URI is the sensor or sensor network metadata (type of sensor, type of measurements made, etc.).

For example, for the sensor that streams the Heart Rate for a Patient 1, we could use the following URI:

```
http://www.linkeddatastreams.org/sensor/heartrate/1
```

Following the guidelines on "cool URIs" for real-world objects², these URIs have to be de-referenceable on the Web and be unambiguous, which is the case in our proposal.

For example, de-referencing the URI that identifies Heart Rate Sensor for Person 1 would lead us to the following RDF document³:

```
@prefix owl: <http://www.w3.org/2002/07/owl#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix hr:
<http://www.linkeddatastreams.org/sensors/ontology/HeartRateMonitor.owl#>
@prefix hrsensor: <http://www.linkeddatastreams.org/sensor/heartrate/>
@prefix sensor: <http://www.csiro.au/Ontologies/2009/SensorOntology.owl#>

hrsensor:1          rdf:type sensor:Sensor ;
                    sensor:measures _measurement .
_measurement        rdf:type hr:HeartRateMonitor .
```

4.2 URIs for Time

Sensors normally emit data at a certain frequency. In the Heart Rate case, data is streamed every couple seconds. In the Linear Road Benchmark, data is streamed every 30 seconds. Hence it is advisable to have a URI scheme that identifies the observations that the sensor emits at a given time, such as the following:

```
http://www.linkeddatastreams.org/sensor/name/%time%
```

Furthermore, in many situations it would be also useful to represent time windows (intervals of time), as it happens with most data stream and sensor network query languages (e.g., CQL [3]). Therefore we propose the following URI scheme:

² <http://www.w3.org/TR/cooloris>

³ Please note that the type of metadata that a sensor can provide is not standardised, and this is only a simple example of how the URI would be dereferenced. The W3C incubator group on Semantic Sensor Networks is currently working on different types of sensor ontologies, which include sensor types, observations and measures, etc.



`http://www.linkeddatastreams.org/sensor/name/%start time%,%end time%`

For example, the following URI identifies the observations from the Heart Rate sensor of Person 1 on July 15, 2009 at 17:00:

`http://www.linkeddatastreams.org/sensor/hearttrate/1/2009-07-15 17:00:00`

De-referencing this URI may lead us to an RDF document like the following:

```
hrsensor:1  sensor:measures
<http://www.linkeddatastreams.org/sensor/hearttrate/1/2009-07-15 17:00:00>.

<http://www.linkeddatastreams.org/sensor/hearttrate/1/2009-07-15 17:00:00>
  rdf:type  hr:HeartRateMonitor;
  hr:heartRate  "74";
  hr:timestamp  "2009-07-15 17:00:00"^^xsd:dateTime .
```

And the following URI identifies the observations from the Heart Rate sensor of Person 1 between July 15, 2009 at 17:00 and July 15, 2009 at 18:00:

`http://www.linkeddatastreams.org/sensor/hearttrate/1/2009-07-15 17:00:00,
2009-07-15 18:00:00`

For example, de-referencing this URI may lead us to the following RDF document⁴:

```
hrsensor:1  sensor:measures
<http://www.linkeddatastreams.org/sensor/hearttrate/1/2009-07-15 17:00:00>.
hrsensor:1  sensor:measures
<http://www.linkeddatastreams.org/sensor/hearttrate/1/2009-07-15 17:00:05>.
hrsensor:1  sensor:measures
<http://www.linkeddatastreams.org/sensor/hearttrate/1/2009-07-15 17:00:10>.
```

Note that if `%start time%` and `%end time%` are the same, then the time window is 0, hence the URI would be the same as the URI with one given time. For example:

```
<http://www.linkeddatastreams.org/sensor/hearttrate/1/2009-07-15 17:00:00,
2009-07-15 17:00:00>
owl:sameAs
<http://www.linkeddatastreams.org/sensor/hearttrate/1/2009-07-15 17:00:00>
```

4.3 URIs for Space

A sensor may also emit its current location together with the rest of data, if it is mobile, or may have as part of its metadata information about the location where it is placed, in the case of static

⁴ The three measures hereby shown as an example may also come bundled in an RDF list.

ones. In the Heart Rate and the Linear Road cases, sensors are mobile, hence their coordinates depend on the sensor location.

The Linked GeoData project has already proposed a way of representing spatial dimensions as Linked Data. Therefore we follow their guidelines. A URI to identify spatial dimension from a sensor would be:

```
http://www.linkeddatastreams.org/sensor/name/%latitude%,%longitude%/%radius%
```

For example, the following URI identifies the car position emitted from Car 1 in a 1 meter radius from the coordinates 50.60242, -2.5225.

```
http://www.linkeddatastreams.org/sensor/car/1/50.60242,-2.5225/1
```

For example, de-referencing this URI would lead us to the following RDF document:

```
@prefix carsensor: <http://www.linkeddatastreams.org/sensor/car/>
carsensor:1  sensor:measures
<http://www.linkeddatastreams.org/sensor/car/1/50.7, -2.6>.
carsensor:1  sensor:measures
<http://www.linkeddatastreams.org/sensor/car/1/51.2, -3.3>.
...
```

This leads us to have URIs that identify observations from sensors are specific locations. If this type of URI would be de-referenced, it would lead us to the following RDF document:

```
carsensor:1  sensor:measures
<http://www.linkeddatastreams.org/sensor/car/1/50.7, -2.6>.

<http://www.linkeddatastreams.org/sensor/car/1/50.7, -2.6>
  rdf:type    lr:CarLoc;
  lr:speed    "60";
  hr:timestamp "2009-07-15 17:00:00"^^xsd:dateTime .
```

A same location can emit different observations, and this would be differentiated through time, because we assumed that two or more observations of a sensor cannot be emitted at the same time.

In these examples, we have used a radius as the bounding area. Other possibilities can be the use of a bounding box or a bounding polygon.

4.4 URIs for Time and Space

Now that we have proposed how to represent Time and Space in URIs separately, we propose a way to represent them together. As discussed before, this type of URI makes more sense for sensors that are mobile. Given that we have proposed two ways of representing time in URIs (specific time and time window), we therefore propose the following two schemes for

representing time and space in URIs (please note that temporal attributes go before the spatial attributes, however, there is no strong reason for this and it could be done in a different order):

```
http://www.linkeddatastreams.org/sensor/name/%time%/%latitude%,%longitude%/%radius%
```

```
http://www.linkeddatastreams.org/sensor/name/%start time%,%end time%/%latitude%,%longitude%/%radius%
```

The following URI identifies the observations of the Heart Rate sensor of Person 1 on July 15, 2009 at 17:00 when it was in a 1 meter radius from the coordinates 50.60242, -2.5225:

```
http://www.linkeddatastreams.org/sensor/heartrate/1/2009-07-15 17:00:00/50.60242,-2.5225/1
```

The following URI takes in account a time window and identifies the observations of the Heart Rate sensor of Person 1 between July 15, 2009 at 17:00 and July 15, 2009 at 18:00 when it was in a 1 meter radius from the coordinates 50.60242, -2.5225:

```
http://www.linkeddatastreams.org/sensor/heartrate/1/2009-07-15 17:00:00,2009-07-15 17:00:00/50.60242,-2.5225/1
```

5. Related Work

To our knowledge, there has not been an approach that applies the linked data principles to stream data. However, several other approaches take in account some of the issues that we have presented in this paper.

First, we have presented a set of human-friendly URIs that can be used to identify sensors and sensor data. However, these human-friendly URIs may be written in many other different ways. For example, Davis [4,5] proposes a different way of representing intervals in URIs. Instead of having the start and end time in the URI, one could write the start time with the duration. This approach is feasible and we consider that a user should decide what the best approach is.

The approach presented by Pfeiffer et al [6] specifies a syntax for addressing time intervals within time-based Web resources through URI queries and fragments. However, these types of URIs do not follow the Cool URIs approach used in Linked Data.

Hausenblas et al [7] presented an approach for applying the Linked Data principles to multimedia fragments. This approach focuses on addressing multimedia fragments through a URI-based mechanism. The main motivation behind it is the ability to send only the multimedia fragment that is wanted by the user, instead of sending the whole multimedia file and then having the user multimedia client find the relevant fragment, which is what it is currently done. Furthermore, this approach also describes metadata about multimedia fragments, which enables it to be part of the Linked Data cloud.

6. Conclusions, Open Challenges and Future Work

In this position paper we have described our proposal for the generation of Linked-Data-observant URIs for sensors and sensor data, so that this wealth of information could be easily included in the Linked Data cloud. We have covered the two most relevant issues that have to be dealt with when considering this type of data, time and space, and we have tried to be compliant with existing proposals that have addressed the use of time and space in URIs, including Linked Data efforts in the of geolocation-related information.

One important aspect to be considered is that by just generating human-readable URIs that encode internally time and space constraints we are not providing the explicit semantics of those URIs so that they could be adequately consumed by semantic-aware systems. This means that when dereferencing these URIs, the results obtained (e.g., in RDF) should also reflect that information as part of the information provided.

Another important consideration that has to be made is that this way of defining URIs allows for the easy creation of REST-like query interfaces to sensor data. The time and space constraints expressed in the URIs can be transformed into stream query languages that will allow performing transformations into RDF on the fly. However, architectural decisions about how to make these transformations still need to be made, and corresponding implementations will need to be done as well.

A final aspect that has not been considered in our proposal, and that is still open, is the handling of uncertainty in the data that is provided when dereferencing URIs or when expressing them. We have discussed that this is an important feature of this type of data and should be handled somehow.

Finally, it is important to note that other sources of information, such as RSS feeds, social network feeds (Twitter, Facebook, etc.) may be also considered as stream data, even if used for different types of applications. Even though in this paper we are considering stream data emitted from physical sensors, our proposal may be applied to these other types of stream data.

Acknowledgements

This work is supported by the EC project SemsorGrid4Env (<http://www.semsorgrid4env.eu/>). We appreciate the discussions held with the project members about the different requirements and design decisions taken, which have allowed us to improve our proposal.

References

1. Magdalena Balazinska, Amol Deshpande, Michael J. Franklin, Phillip B. Gibbons, Jim Gray, Mark Hansen, Michael Liebhold, Suman Nath, Alexander Szalay, Vincent Tao, "Data Management in the Worldwide Sensor Web," IEEE Pervasive Computing, vol. 6, no. 2, pp. 30-40, Apr.-June 2007
2. Arasu, A., Cherniack, M., Galvez, E., Maier, D., Maskey, A., Ryvkina, E., Stonebracker, M., Tibbetts, R. Linear Road: A Stream Data Management Benchmark. In Proceedings of Very Large Databases Conference 2004.
3. Arasu, A., Babu, S., Widom, J., The CQL continuous query language: semantic foundations and query execution. VLDB Journal 15(2). 2006
4. Davis, I. <http://placetime.com/instant/gregorian/>



5. Davis, I. <http://placetime.com/interval/gregorian/>
6. Pfeiffer, S. Parker, C. Pang, A. (2005) Specifying time intervals in URI queries and fragments of time-based Web resources. Internet Draft. <http://www.annodex.net/TR/draft-pfeiffer-temporal-fragments-03.html>
7. Hausenblas, M. Troncy, R. Buerger, T. Raimond, Y. (2009) Interlinking Multimedia: How to Apply Linked Data Principles to Multimedia Fragments. Linked Data on the Web Workshop at WWW 2009.