SemSorGrid4Env

FP7-223913

Deliverable

D4.2v2

Implementation and Deployment of the Ontology-based Data Integration Service v2

Jean-Paul Calbimonte, and Oscar Corcho
Universidad Politécnica de Madrid

Alasdair J G Gray
University of Manchester

February 23, 2011

Status: Final version
Scheduled Delivery Date: 28 February 2011

Executive Summary

The realization 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. In recent years streaming data sources have become increasingly available thanks to advances in ubiquitous data capturing technologies such as sensor networks.

The required transparent integration of heterogeneous sources of this kind has brought new challenges to the research community. In the context of the SemSorGrid4Env project and its Work Package 4 (WP4), we propose the design, implementation and deployment of a Semantic Integration Service for streaming and stored data sources. The design of this service follows an ontology-based approach that extends previous works on data access for relational databases.

This deliverable focuses on a core component of the semantic Integrator: the Ontology-based streaming data access module. To the best of our knowledge there is still no bridging solution that allows connecting ontology-based mappings and streaming data access technologies coherently in order to answer the requirements of i) establishing mappings between ontological models and streaming data source schemas, and ii) accessing streaming data sources through queries over ontology models. In summary our approach consists in creating an Ontology-based Streaming Data Access service that can receive requests over an ontological view and transforms them into queries for acquisitional or event-based stream sources or stored sources. The results of these queries can be integrated following a query plan and returned as RDF triples in terms of the global ontology. The implementation of the proposed solution is presented as part of the SemSorGrid4Env software platform.

The previously published deliverable D4.1 [CCG09] presented the design of the ontology-based integration service, and an initial prototype was presented in D4.2v1 [CC09]. This document describes the final implementation of the service, called Integration and Query Service (IQS) and how it fits in the SemSorGrid4Env architecture. It differs from the precedent version in the following aspects:

- **Section 1** The system description has been updated to reflect the current status. The IQS service specification has been added, and the component formerly known as *StreamMapster* is referred as Semantic Integrator and includes a more complete description of its functionality.
- **Section 2** The whole section has been added in order to explain the theoretical foundations of the delivered system.
- **Section 3** The code description and dependencies have been updated to reflect the current status.
- **Section 4** The data sources used have been described in detail.
- **Section 5** Full examples of how to use the IQS service have been included, i.e. registering an integrated data sources, creating queries and pulling data.
- **Section 6** The whole section containing the test strategy has been added.

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:

- Section 1 contains material from [GGF+10] and [CCG10].
- Section 2 contains material from [CCG10] and [CCG09].

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	www.semsorgrid4env.eu/files/deliverables/wp4/D4.2v2.pdf		
EU Project Officer	Antonios Barbas		

Deliverable	**Number**	D4.2v2	**Name**	Implementation and Deployment of the Ontology-based Data Integration Service v2		
Task	**Number**	T4.2	**Name**	Implement and deploy the Ontology-based integration model in the SemSorGrid4Env infrastructure		
Work package	**Number**			WP4		
Date of delivery	**Contract**	28 February 2011		**Actual**	28 February 2011	
Code name	D4.2v2			**Status**	draft ☐	final ☑
Nature	Prototype ☐ Report ☑ Specification ☐ Tool ☐ Other ☐					
Distribution Type	Public ☑ Restricted ☐ Consortium ☐					
Authoring Partner	UPM					
QA Parner	UNIMAN					
Contact Person	Jean-Paul Calbimonte					
	Email	jp.calbimonte@upm.es		**Phone**	+34913363670	**Fax** +34913524819
Abstract (for dissemination)	Thanks to the increasingly available ubiquitous data capturing technologies such as sensor networks, new requirements for streaming data access solutions have surfaced. One of these is the integration of heterogeneous streaming data sources. In the context of the SemSorGrid4Env project and its Work Package 4 (WP4), we propose the design, implementation and deployment of a Semantic Integration Service for streaming and stored data sources. The design of this service follows an ontology-based approach that extends previous works on data access for relational databases. This deliverable focuses on a core component of the semantic Integrator: the Ontology-based streaming data access module. This module, named Semantic Integrator, extends the ODEMAPSTER engine that provided ontology-based access for databases, and uses the language for Stream-to-ontology mappings S2O. An implementation of the proposed solution is presented, so that we can provide some evidence of the applicability of our approach. This final deliverable of the implementation has been completed and presented as part of the entire SemSorGrid4Env platform.					
Keywords	Semantic Integration, Query Translation, Mappings					

Version log/Date	**Change**	**Author**
0.1 / 1 December 2009	Initial Draft	J-P. Calbimonte
0.2 / 14 December 2009	Revised Text	J-P. Calbimonte
0.3 / 15 December 2009	Comments from QA	J-P. Calbimonte and O. Corcho
0.4 / 18 December 2009	Revised Text	J-P. Calbimonte
1.1 / 18 January 2011	New version v2	J-P. Calbimonte
1.2 / 8 February 2011	Added Theory Section	J-P. Calbimonte
1.3 / 14 February 2011	Added Code Section	J-P. Calbimonte
1.4 / 15 February 2011	Corrections	O. Corcho
1.5 / 16 February 2011	Remarks QA	A. Gray
1.6 / 23 February 2011	Corrections QA	J-P. Calbimonte

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	Prof. Norman Paton Department of Computer Science The University of Manchester Oxford Road Manchester, M13 9PL, United Kingdom #@ npaton@cs.man.ac.uk #t +44-161-275-69 10, # +44-161-275 62 04
National and Kapodistrian University of Athens	NKUA	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	Dr. Kirk Martinez University Road Southampton SO17 1BJ United Kingdom #@ km@ecs.soton.ac.uk #t +44 23 80594491, #f +44 23 80595499
Deimos Space, S.L.U.	DMS	Mr. Agustín Izquierdo Ronda de Poniente 19, Edif. Fiteni VI, P 2, 2° 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	Dr. 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

Contents

List of Figures

List of Tables

Glossary

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

ARQ SPARQL processor for Jena.

CCO Chanel Coastal Observatory.

D2RQ Platform for Database-to-RDF processing.

GAV Global-as-View integration.

IQS Integration and Query Service.

IRI Internationalized Resource Identifier.

NFCDD National Flood and Coastal Defence Database.

ODEMAPSTER Library for transforming relational data to RDF data..

OWL Web Ontology Language.

QName Qualified name within an XML document.

RDF The Resource Description Framework. A W3C standard originally designed for modelling metadata about Web resources.

R2RML Relational-to-RDF mapping language.

SNEE Sensor Network Evaluation Engine for processing queries over streams of data.

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

SOAP Simple Object Access Protocol.

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

SPARQL$_{\text{Stream}}$ SPARQL extended language for streaming data.

S$_2$O Stream-to-Ontology mapping language based on R2O.

SWEET Semantic Web for Earth and Environmental Terminology.

WSDL Web Services Description Language.

W3C World Wide Web Consortium.

1. Introduction

This document describes the implementation of the ontology-based semantic integration service of the SemSorGrid4Env architecture, and the technical details of its functionality.

1.1 Data Integration in SemSorGrid4Env

The main objective of the SemSorGrid4Env[1] project is to specify, design, implement, evaluate and deploy a service-oriented architecture and middleware which allows application developers to build semantic-based sensor network applications for environmental management [GGF+10]. In this novel architecture, semantic technologies are exploited in various ways, and one of which is the integration of stored and streaming data sources.

One of the objectives of Work Package 4 (WP4) is to design and implement an ontology-based data integration service for streaming and stored data. This document describes the final implementation of the service, called Integration and Query Service (IQS) and how it fits in the SemSorGrid4Env architecture.

In summary, the IQS service provides the following functionality:

- Create an integrated data resource exposed through an ontological schema, given a set of streaming data resources and a mapping document.
- Query integrated data resources using a SPARQL extension for streaming data, while the service internally accesses the underlying data sources.
- Provide data access to the continuous data queries registered for the integrated data sources.
- Register the created data resources in the SemSorGrid4Env Registry [KKK09].
- Provide metadata for the integrated data resources in the form of property documents as specified in the architecture [GGF+10].

The remainder of this deliverable document is organized as follows. We complete this section with a high level overview of the system objectives and the components delivered. In Section 2 we present the theoretical foundations of this work. Then we provide details about the source code of the components in Section 3, and about the data sources used in Section 4. We provide build and installation instructions, as well as a few use-case examples in Section 5, and finally we explain the test suites performed and planned for the delivered code.

1.2 System Objectives

The objectives of the ontology-based data integration service developed for phase II of the SemSorGrid4Env project are to:

- Provide a WSDL specification of the Integration and Query Service, including the integration, query, data access and subscription interfaces presented in the SemSorGrid4Env architecture [GGF+10].

[1] Semantic Sensor Grid for Rapid Application Development for Environmental Management, http://www.semsorgrid4env.eu.

- Provide an Implementation of the Integration and Query Service, capable of integrating streaming sources, and querying them through the SPARQL extended language $\text{SPARQL}_{\text{Stream}}$.
- Implement a query translation component capable of transforming $\text{SPARQL}_{\text{Stream}}$ queries into data stream queries using the S_2O mappings.

1.3 System Description

The system developed consists of two main components. The first is a software library that is capable of translating $\text{SPARQL}_{\text{Stream}}$ queries into data stream queries in a target language in terms of the underlying stream schemas, and then executing these queries and returning the results as SPARQL bound variables. This component is called Semantic Integrator. The other component is the Integration and Query Service (IQS), which provides a web service interface according to the SemSorGrid4Env architecture specification.

1.3.1 Semantic Integrator

Our approach to enable ontology-based access to streaming data is depicted in Figure 1.1. The service receives queries specified in terms of the classes and properties[2] of the ontology using $\text{SPARQL}_{\text{Stream}}$, an extension of SPARQL that supports operators over RDF streams (see Section 2.2.1). In order to transform the $\text{SPARQL}_{\text{Stream}}$ query, expressed in terms of the ontology, into queries in terms of the data sources, a set of mappings must be specified. These mappings are expressed in S_2O, an extension of the R_2O mapping language, which supports streaming queries and data, most notably window and stream operators (see Section 2.2.2). This transformation process is called *query translation*, and the target is the continuous query language SNEEql, which is expressive enough to deal with both streaming and stored sources.

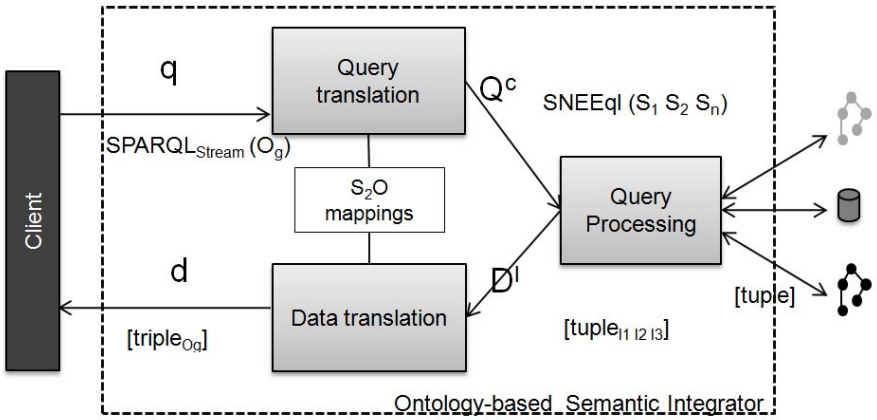

Figure 1.1: Ontology-based Semantic Integrator for streaming data

After the continuous query has been generated, the *query processing* phase starts, and the evaluator uses distributed query processing techniques [Kos00] to extract the relevant data from the sources and perform the required operations, e.g. selection, projection, and joins (See Section 2.3.1). Note that query execution in sources such as sensor networks may include in-network query processing, pull or push based delivery of data between sources, and other data source specific settings. The result of the query processing is a set of tuples that the *data translation* process transforms into ontology instances.

[2] We use the OWL nomenclature of classes, and object and datatype properties for naming ontology elements.

This approach requires several contributions and extensions to the existing technologies for continuous data querying, ontology-based data access, and SPARQL query processing. This deliverable focuses on a first stage that includes the process of transforming the $\text{SPARQL}_{\text{Stream}}$ queries into queries over the streaming data sources using SNEEql as the target language. In Section 2 we provide the syntax and semantics for the querying of streaming RDF data and the mappings between streaming sources and an ontology. In the remainder of this section we provide details about the sub-components of the Semantic Integrator delivered in D4.2v2.

Query Translation

During query translation, the following phases can be identified:

- **Query Parsing** First the incoming $\text{SPARQL}_{\text{Stream}}$ query, specified in terms of an ontological schema, is parsed using an extended version of the ARQ SPARQL parser [3]. Consequently the result of this phase is an extension of the ARQ SPARQL abstract syntax tree. This tree can be easily explored to find the elements that will later be mapped and translated.
- **Mapping Reading** The stream-to-ontology mapping is read and the relevant mappings to the $\text{SPARQL}_{\text{Stream}}$ input query are extracted. These mappings are provided as an input to the translation phase. The R_2O extended language, called S_2O is used as mapping language, although there is also support for the R2RML mapping language format[4].
- **Translation** The translation phase uses the $\text{SPARQL}_{\text{Stream}}$ syntax tree and the relevant mapping definitions to construct streaming relational algebra trees. These algebra expressions can be optimized using standard static optimization techniques, and is finally serialized into a streaming query language expression. The current implementation translates queries to the SNEEql language. The details of the translation process is explained in Section 2.3.

Query Processing

In this stage the Semantic Integrator delegates query execution to the distributed query processing service [GGF+10]. In order to be able to be used by the Query Executor sub-component, a wrapper class must be implemented for the specific engine. Currently it is a wrapper of a SemSorGrid4Env web-service implementation of the streaming data service. Alternatively, the SNEE engine is supported natively through its Java API, as it was initially implemented in D4.2v1.

Data Translation

After the results are returned by the Query Processing component, they are translated again to instances of the ontologies referenced in the original query. This process requires again the mappings of the translation phase. Currently results are returned as SPARQL bound variables.

1.3.2 Integration and Query Service

The functionality exposed by the IQS is specified by the Integration, Query and Pull Data interfaces of the SemSorGrid4Env architecture[GGFP09]. We present them in tables 1.1, 1.2 and 1.3.

The service implementation is based on the capabilities of the Semantic Integrator library described in Section 1.3.1. Examples and details of the whole process of query translation and execution is provided in Section 2 and in [CCG10].

[3] http://jena.sourceforge.net/ARQ/
[4] http://www.w3.org/2001/sw/rdb2rdf/r2rml/

Types	
DataResourceAbstractName	The abstract name associated with the data resource(s) rep resented as a URI (As defined in WS-DAI).
MappingDocument	An XML document consisting of a mapping format URI and the encoded mappings between the data sources and the resulting global schema.
Faults	
ServiceBusyFault	The service is already processing a message and concurrent operations are not supported.
NotAuthorizedFault	The consumer is not authorized to perform the requested operation or is not authorized to perform the requested operation at this time.
InvalidResourceNameFault	The data resource specified is unknown to the service.
DataResourceUnavailableFault	The data resource that is the target of the message is currently not available. This could be caused by a temporary fault or could indicate that the data resource has stopped operating permanently.
InvalidMappingDocumentFault	The mapping document specified is not in a valid format or cannot be processed by the service.
Operations	
IntegrateAs	Creates a data resource which presents the global view over a set of data sources.
Inputs	resourceNames+: DataResourceAbstractName, mappings?: MappingDocument, integratedSourceName?: String
Output	integratedDataResource: EndpointReference
Faults	InvalidDataResourceNameFault, DataResourceUnavailableFault, InvalidMappingDocumentFault, NotAuthorizedFault, ServiceBusyFault
AddSource	Add one or more data sources to the set of known data sources.
Inputs	resourceNames+: DataResourceAbstractName,
Output	
Faults	InvalidDataResourceNameFault, DataResourceUnavailableFault, NotAuthorizedFault, ServiceBusyFault
RemoveSource	Remove one or more data sources from the set of known data sources.
Inputs	resourceNames+: DataResourceAbstractName,
Output	
Faults	InvalidDataResourceNameFault, DataResourceUnavailableFault, NotAuthorizedFault, ServiceBusyFault

Table 1.1: The Integration Interface.

Types	
DataResourceAbstractName	The abstract name associated with the data resource(s) represented as a URI.
DatasetFormatURI	The URI of a dataset format.
RequestDocument	An XML document that contains the request expression, i.e. the query and its parameters.
ResponseDocument	A document consisting of a dataset format URI and the encoded data.
ConfigurationDocument	An XML document containing the initial parameters for the indirect access resource.
PropertyDocument	An XML document conforming to a defined XML schema to describe the properties of the service.
Faults	
ServiceBusyFault	The service is already processing a message and concurrent operations are not supported.
NotAuthorizedFault	The consumer is not authorized to perform the requested operation or is not authorized to perform it at this time.
InvalidResourceNameFault	The data resource specified is unknown to the service.
DataResourceUnavailableFault	The data resource that is the target of the message is currently not available.
InvalidExpressionFault	The expression given as part of the request contains errors.
InvalidLanguageFault	The input dataset (usually the expression component of an incoming request) has an unrecognized language element.
InvalidDatasetFormatFault	The dataset format URI specified is not known to the service.
InvalidConfigurationDocumentFault	The configuration document specified is not valid.
Operations	
GetSPARQLPropertyDocument	Returns the core property document values associated with the service implementing this message.
Inputs	resourceName: DataResourceAbstractName
Output	properties: SPARQLPropertyDocument
Faults	InvalidResourceName, DataResourceUnavailableFault, NotAuthorizedFault, ServiceBusyFault
DestroyDataResource	Destroy the named data resource; future messages directed at the resource MUST yield an InvalidResourceNameFault.
Inputs	resourceName: DataResourceAbstractName
Output	
Faults	InvalidResourceName, DataResourceUnavailableFault, NotAuthorizedFault, ServiceBusyFault
SPARQLExecute	Directs a query document to a data resource.
Inputs	resourceName: DataResourceAbstractName format?: DatasetFormtURI, queryExpression: RequestDocument
Output	queryResponse: ResponseDocument
Faults	InvalidResourceNameFault, DataResourceUnavailableFault, InvalidDatasetFormatFault, InvalidExpressionFault, InvalidLanguageFault, NotAuthorizedFault, ServiceBusyFault
SPARQLExecuteFactory	Creates a relationship between a data resource representing the result of a query and the data access service by which it will be accessed.
Inputs	resourceName: DataResourceAbstractName responseInterfaceType?: QName , configurationDocument?: ConfigurationDocument, preferredTargetService?: EndPointReference, requestDocument: RequestDocument
Output	queryResponse: QueryExpressionDocument
Faults	InvalidResourceNameFault, DataResourceUnavailableFault, InvalidPortTypeQnameFault, InvalidExpressionFault, InvalidConfigurationDocumentFault, InvalidLanguageFault, NotAuthorizedFault, ServiceBusyFault

Table 1.2: The Query Interface.

Types	
DataResourceAbstractName	The abstract name associated with the data resource(s) represented as a URI.
DatasetFormatURI	The URI of a dataset format.
StreamDataset	A document consisting of a dataset format URI and the encoded data.
PropertyDocument	An XML document conforming to a defined XML schema to describe the properties of the service.
Faults	
ServiceBusyFault	The service is already processing a message and concurrent operations are not supported.
NotAuthorizedFault	The consumer is not authorized to perform the requested operation or is not authorized to perform the requested operation at this time.
InvalidResourceNameFault	The data resource specified is unknown to the service.
DataResourceUnavailableFault	The data resource that is the target of the message is currently not available.
Operations	
GetDataResourcePropertyDocument	Retrieves the PropertyDocument for the pull stream response resource.
Inputs	resourceName: DataResourceAbstractName
Output	properties: PropertyDocument
Faults	InvalidResourceName, DataResourceUnavailableFault, NotAuthorizedFault ServiceBusyFault
GetStreamItem	Retrieves a specified count of stream items from the pull stream service from a specific point in the available history.
Inputs	resourceName: DataResourceAbstractName datasetFormatURI?:anyURI, position?: xsd:dateTime | xsd:int, count?: xsd:duration | xsd:int, maxTuples?: xsd:int
Output	streamDataset:StreamDataset
Faults	InvalidResourceName, DataResourceUnavailableFault, InvalidDatasetFormatFault, NotAuthorizedFault, InvalidWindowFault, MaxTuplesExceededFault, ServiceBusyFault
GetStreamNewestItem	Retrieves the most recent stream items in the pull stream response resource up to the specified count.
Inputs	resourceName: DataResourceAbstractName datasetFormatURI?:anyURI, count?: xsd:duration | xsd:int, maxTuples?: xsd:int
Output	streamDataset:StreamDataset
Faults	InvalidResourceName, DataResourceUnavailableFault, InvalidDatasetFormatFault, NotAuthorizedFault, InvalidWindowFault, MaxTuplesExceededFault, ServiceBusyFault

Table 1.3: The Pull Data Interface.

2. Theoretical Background

In this section we provide details about the theoretical foundations of the query translation mechanisms used to access streaming data sources through $\text{SPARQL}_{\text{Stream}}$ queries over ontological schemas.

2.1 Motivation

Recent advances in wireless communications and sensor technologies have opened the way for deploying networks of interconnected sensing devices capable of ubiquitous data capture, processing and delivery. Sensor network 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 variety of applications including security surveillance, healthcare provision, and environmental monitoring.

As an example, consider a web application which aids an emergency planner to detect and coordinate the response to flood risk alerts along the south east coast of England. This involves retrieving relevant data from multiple sources, e.g. meteorological forecasts from the Met Office (UK's National Weather Service)[1], real-time wave and tide data from sensor networks deployed in the region by the CCO (Channel Coastal Observatory)[2], and any other relevant sources of data such as the NFCDD (UK's Environment Agency[3] National Flood and Coastal Defence Database) providing data about coastal defences. Typically sources are managed autonomously and model their data according to the needs of the deployment. To integrate the data requires linking the sources to a common data model so that conditions that are likely to cause a flood can be detected, and presented to the user in terms of their domain, e.g. flood risk assessment. We propose that ontologies can be used as such a common model. For the scenario presented here, we use an ontology that extends ontologies from SWEET[4] and the W3C incubator group's semantic sensor network ontology[5].

The semantic web community has already addressed the problem of accessing stored data sources using mappings, such as R_2O [BCGP04] and D2RQ [BC07]. However, similar solutions for streaming data mapping and querying using ontology-based approaches have not been explored yet. The database community has investigated data stream processing where the data is viewed as an append-only sequence of tuples. Systems such as STREAM [ABB+06] and Borealis [AAB+05] have focused on query evaluation and optimization over streams with high, variable, data rates. Other systems such as SNEE [GBG+11] and TinyDB [MFHH05], have focused on data generated by sensor networks, which tends to be at a lower rate, and query processing in the sensor network where resources are more constrained and energy efficiency is the primary concern. There have also been proposals for query processing over streaming RDF data [BGJ08, BBCG10].

However there is still no bridging solution that connects these technologies coherently in order to answer the requirements of i) establishing mappings between ontological models and streaming data source schemas, and ii) accessing streaming data sources through queries over ontology models.

In this section we focus on providing ontology-based access to streaming data sources, including sensor networks, through declarative continuous queries. We build on the existing work of R_2O for enabling ontology-based access to relational data sources, and SNEE for query evaluation over streaming and stored data sources. This constitutes a first step towards a framework for the

[1] http://www.metoffice.gov.uk accessed 15 September 2010.
[2] http://www.channelcoast.org/ accessed 15 September 2010.
[3] http://www.environment-agency.gov.uk/ accessed 15 September 2010.
[4] http://sweet.jpl.nasa.gov/ accessed 15 January 2011.
[5] http://www.w3.org/2005/Incubator/ssn/wiki/Semantic_Sensor_Network_Ontology accessed 15 January 2011.

integration of distributed heterogeneous streaming and stored data sources through ontological models.

2.2 Query and Mapping Syntax

In this section we introduce the SPARQL$_{\text{Stream}}$ query language, an extension to SPARQL for streaming RDF data, which has been inspired by C-SPARQL [BBCG10] and SNEEql [BGFP08]. However, significant improvements have been made to C-SPARQL that correct the types supported and the semantics of windowing operations, which can be summarized as: (i) we only support windows defined in time, (ii) the result of a window operation is a window of triples, not a stream, over which traditional operators can be applied, as such we have added window-to-stream operators, and (iii) we have adopted the SPARQL 1.1 definition for aggregates. We also present S$_2$O for the definition of stream-to-ontology mappings.

2.2.1 SPARQL$_{\text{Stream}}$

Just as in C-SPARQL we define an RDF *stream* as a sequence of pairs (T_i, τ_i) where T_i is an RDF triple $\langle s_i, p_i, o_i \rangle$ and τ_i is a timestamp which comes from a monotonically non-decreasing sequence. An RDF stream is identified by an IRI , which provides the location of the data source[6].

Window definitions are of the form 'FROM *Start* TO *End* [STEP] [*Literal*]', where the *Start* and *End* are of the form NOW or NOW − *Literal*, and *Literal* represents some number of time unit (DAYS, HOURS, MINUTES, or SECONDS)[7]. The optional STEP indicates the gap between each successive window evaluation. Note, if the size of the step is smaller than the range of the window, then the windows will overlap, if it coincides with the size of the window then every triple will appear in one and only one window, and if the step is larger than the range then the windows *sample* the stream. Also note that the definition of a window can be completely in the past. This is useful for correlating current values on a stream with values that have previously occurred.

The result of applying a window over a stream is a timestamped bag of triples over which conjunctions between triple patterns, and other "classical" operators can be evaluated. Windows can be converted back into a stream of triples by applying one of the window-to-stream operators in the SELECT clause: ISTREAM for returning all newly inserted answers since the last window, DSTREAM for returning all deleted answers since the last window, and RSTREAM for returning all answers in the window. Listing 1 shows a complete SPARQL$_{\text{Stream}}$ query which, every minute, returns the average of the last 10 minutes of wind speed measurements for each sensor, if it is higher than the average speed from 2 to 3 hours ago.

Note, SPARQL$_{\text{Stream}}$ only supports time-based windows. Other similar languages, such as C-SPARQL also have the notion of triple-based windows. However, such windows are problematic to define in an approach where RDF triples are generated on the fly, since the number of triples required to generate an answer may be greater than the size of the triple window. For example, consider a window size of 1 triple and the graph pattern from the example query in Listing 1. Only one of the triples that form the graph pattern would be kept by the window, which provides insufficient information to compute the expected query answer.

2.2.2 S$_2$O: Defining Stream-to-Ontology Mappings

The mapping document that describes how to transform the data source elements to ontology elements is written in the S$_2$O mapping language, an extended version of R$_2$O [BCGP04]. An R$_2$O

[6] Note in our work the IRI identifies a virtual RDF stream since it is derived from the streaming data sources.
[7] Note that the parser will also accept the non-plural form of the time units and is not case sensitive.

```
PREFIX flood: <http://www.semsorgrid4env.eu#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT RSTREAM ?WindSpeedAvg
FROM STREAM <www.semsorgrid4env.eu/SensorReadings.srdf> [FROM NOW - 10 MINUTES TO NOW STEP
    1 MINUTE]
FROM STREAM <www.semsorgrid4env.eu/SensorArchiveReadings.srdf> [FROM NOW - 3 HOURS TO NOW
    -2 HOURS STEP 1 MINUTE]
WHERE {
  {
    SELECT AVG(?speed) AS ?WindSpeedAvg
    WHERE
    {
        GRAPH <www.semsorgrid4env.eu/SensorReadings.srdf> {
        ?WindSpeed a flood:WindSpeedObservation;
            flood:hasSpeed ?speed; }
    } GROUP BY ?WindSpeed
  }
  {
    SELECT AVG(?archivedSpeed) AS
       ?WindSpeedHistoryAvg
    WHERE
    {
      GRAPH <www.semsorgrid4env.eu/SensorArchiveReadings.srdf> {
        ?ArchWindSpeed a flood:WindSpeedObservation;
        flood:hasSpeed ?archivedSpeed; }
    } GROUP BY ?ArchWindSpeed
  }
  FILTER (?WindSpeedAvg > ?WindSpeedHistoryAvg)
}
```

Listing 1: An example SPARQL$_{\text{Stream}}$ query which every minute computes the average wind speed measurement for each sensor over the last 10 minutes if it is higher than the average of the last 2 to 3 hours.

mapping document includes a section that describes the database relations, dbscehma-desc. In order to support data streams, R$_2$O has been extended to also describe the data stream schema. A new component called streamschema-desc has been created, as shown in the top part of Listing 2.

The description of a relational stream is similar to a relation in terms of the R$_2$O language. An additional attribute streamType has been added, it denotes the kind of stream in terms of data acquisition, i.e. event or acquisitional, as this will have an impact on the underlying implementation. In the same way as key and non-key attributes are defined, a new timestamp-desc element has been added to provide support for declaring the stream timestamp attribute. Since S$_2$O extends R$_2$O, relations can also be specified using the existing R$_2$O mechanism. For the class and property mappings, the existing R$_2$O definitions can be used for stream schemas just as it was for relational schemas. This is specified in the conceptmap-def element and the attributemap-def elements, as shown in the bottom part of Listing 2.

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). Finally, a SPARQL$_{\text{Stream}}$ streaming query requires an RDF stream to have an IRI identifier. For this, S$_2$O creates a *virtual* RDF stream and its IRI is specified in the S$_2$O mapping using the virtualStream element. It can be specified at the conceptmap-def level or at the attributemap-def level.

```
streamschema-desc
  name MeteoSensors
  has-stream SensorWind
    streamType push
    documentation "Wind measurements"
    keycol-desc sensorId
      columnType integer
    timestamp-desc  timestamp
      columnType datetime
    nonkeycol-desc  speed
      columnType float
    nonkeycol-desc direction
      columnType float

conceptmap-def WindSpeedObservation
  virtualStream <http://semsorgrid4env.eu/SensorReadings.srdf>
  uri-as
    concat(SensorWind.sensorId)
  described-by
    attributemap-def observationResult
      virtualStream http://semsorgrid4env.eu/SensorReadings.srdf>
      operation constant
        has-column SensorWind.speed
    attributemap-def observationResultTime
      virtualStream http://semsorgrid4env.eu/SensorReadings.srdf>
      operation constant
        has-column SensorWind.timestamp
```

Listing 2: An example S2O declaration of a data stream schema and mapping from a stream schema to an ontology concept.

2.3 Semantics of the Streaming Extensions

Now that the syntax of SPARQL$_{\text{Stream}}$ and S2O have been presented, we define their semantics.

2.3.1 SPARQL$_{\text{Stream}}$ Semantics

The SPARQL extensions presented here are based on the formalization of Pérez *et al.* [PAG09]. An RDF stream S is defined as a sequence of pairs (T, τ) where T is a triple $\langle s, p, o\rangle$ and τ is a timestamp in the infinite set of timestamps $\mathbb{T}$. More formally,

$$S = \{(\langle s, p, o\rangle, \tau) \mid \langle s, p, o\rangle \in ((I \cup B) \times I \times (I \cup B \cup L)), \tau \in \mathbb{T}\},$$

where I, B and L are sets of IRIs, blank nodes and literals. Each of these pairs can be called a *tagged triple*.

We define a stream of windows as a sequence of pairs (ω, τ) where ω is a set of triples, each of the form $\langle s, p, o\rangle$, and τ is a timestamp in the infinite set of timestamps $\mathbb{T}$, and represents when the window was evaluated. More formally, we define the triples that are contained in a time-based window evaluated at time $\tau \in \mathbb{T}$, denoted ω^{τ}, as

$$\omega^{\tau}_{t_s,t_e,\delta}(S) = \{\langle s, p, o\rangle \mid (\langle s, p, o\rangle, \tau_i) \in S, t_s \leq \tau_i \leq t_e\}$$

where t_s, t_e define the start and end of the window time range respectively, and may be defined relative to the evaluation time τ. Note that the rate at which windows get evaluated is controlled by the STEP defined in the query, which is denoted by δ.

We define the three window-to-stream operators as

$$\begin{aligned}
\mathsf{RStream}_{(\omega^{\tau},\tau)} &= \{(\langle s, p, o\rangle, \tau) \mid \langle s, p, o\rangle \in \omega^{\tau}\} \\
\mathsf{IStream}_{(\omega^{\tau},\tau),(\omega^{\tau-\delta},\tau-\delta)} &= \{(\langle s, p, o\rangle, \tau) \mid \langle s, p, o\rangle \in \omega^{\tau}, \langle s, p, o\rangle \notin \omega^{\tau-\delta}\} \\
\mathsf{DStream}_{(\omega^{\tau},\tau),(\omega^{\tau-\delta},\tau-\delta)} &= \{(\langle s, p, o\rangle, \tau) \mid \langle s, p, o\rangle \notin \omega^{\tau}, \langle s, p, o\rangle \in \omega^{\tau-\delta}\}
\end{aligned}$$

where δ is the time interval between window evaluations. Note that RStream does not depend on the previous window evaluation, whereas both IStream and DStream depend on the contents of the previous window.

We have provided a brief explanation of the semantics of SPARQL$_{\text{Stream}}$. This is particularly useful in the sense that users may know what to expect when they issue a query using these new operators. However, as the actual data source is not an RDF stream but a sensor network or an event-based stream, e.g. exposed as a SNEEql endpoint, we need to transform the SPARQL$_{\text{Stream}}$ queries into SNEEql queries. The next section describes the semantics of the transformation from SPARQL$_{\text{Stream}}$ to SNEEql using the S$_2$O mappings.

2.3.2 S$_2$O Semantics

In this section we will present how we can use the S$_2$O mapping definitions to transform a set of conjunctive queries over an ontological schema, into the streaming query language SNEEql that is used to access the sources. This work is based on extensions to the ODEMAPSTER processor [BCGP04] and the formalization work of Calvanese *et al.* [CDGL$^+$05] and Poggi *et al.* [PLC$^+$08].

A conjunctive query q over an ontology $\mathcal{O}$ can be expressed as:

$$q(\vec{x}) \leftarrow \varphi(\vec{x}, \vec{y})$$

$$\varphi(\vec{x}, \vec{y}) : \bigwedge_{i=1\ldots k} P_i, \text{ with } P_i \begin{cases} C_i(x), C_i \text{ is an atomic class.} \\ R_i(x,y), R_i \text{ is an atomic property.} \\ x = y \end{cases}$$

x, y are variables in $\vec{x}, \vec{y}$ or constants.

where $\vec{x}$ is a tuple of distinct distinguished variables, and $\vec{y}$ a tuple of non-distinguished existentially quantified variables. The answer to this query consists in the instantiation of the distinguished variables [CDGL$^+$05]. For instance consider:

$$\begin{aligned} q_1(x) \leftarrow & WindSpeedObservation(x) \wedge \\ & isProducedBy(x, y) \wedge \\ & WindSensor(y) \end{aligned}$$

It requires all instances x that are wind speed measurements captured by wind sensors. In this example x is a distinguished variable and y a non-distinguished one. The query has three atoms: $WindSpeedObservation(x)$, $isProducedBy(x, y)$, and $WindSensor(y)$.

Concerning the formal definition of the query answering, let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation, where $\Delta^{\mathcal{I}}$ is the interpretation domain and $\cdot^{\mathcal{I}}$ the interpretation function that assigns an element of $\Delta^{\mathcal{I}}$ to each constant, a subset of $\Delta^{\mathcal{I}}$ to each class and a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ to each property of the ontology. Given a query $q(\vec{x}) \leftarrow \varphi(\vec{x}, \vec{y})$ the answer to q is the set $q^{\mathcal{I}}_{\vec{x}}$ of tuples $\vec{c} \in \Delta^{\mathcal{I}} \times \cdots \times \Delta^{\mathcal{I}}$ that substituted to $\vec{x}$, make the formula $\exists \vec{y}.\varphi(\vec{x}, \vec{y})$ true in $\mathcal{I}$ [CDGL$^+$05, PLC$^+$08, LT09]. Now we can introduce the definition of the mappings. Let $\mathcal{M}$ be a set of mapping assertions of the form:

$$\Psi \rightsquigarrow \Phi$$

where Ψ is a conjunctive query over the global ontology $\mathcal{O}$, formed by terms of the form $C(x)$, $R(x, y)$, $A(x, z)$, with C, R, and A being classes, object properties and datatype properties respectively in $\mathcal{O}$; x, y being object instance variables, and z being a datatype variable. Φ is a set of expressions that can be translated to queries in the target continuous language (e.g. SNEEql) over the sources.

A mapping assertion $C(f^{Id}_C(\vec{x})) \rightsquigarrow \Phi_{S_1,\ldots,S_n}(\vec{x})$ describes how to construct the concept C from the source streams (or relations) $S_1, \ldots, S_n$. The function f^{Id}_C creates an instance of the class C, given

the tuple $\vec{x}$ of variables returned by the Φ expression. More specifically this function will construct the instance identifier (URI) from a set of attributes from the streams and relations. In this case the expression Φ has a declarative representation of the form:

$$\Phi_{S_1,\ldots,S_n}(\vec{x}) = \exists \vec{y}.p^{Proj}_{S_1,\ldots,S_n}(\vec{x}) \wedge p^{Join}_{S_1,\ldots,S_n}(\vec{v}) \wedge p^{Sel}_{S_1,\ldots,S_n}(\vec{v})$$

where $\vec{v}$ is a tuple of variables in either $\vec{x}, \vec{y}$. The term p^{Join} denotes a set of join conditions over the streams and relations S_i. Similarly the term p^{Sel} represents a set of condition predicates over the variables $\vec{v}$ in the streams S_i (e.g. conditions using $<, \leq, \geq, >, =$ operators).

A mapping assertion $R(f^{Id}_{C_1}(\vec{x_1}), f^{Id}_{C_2}(\vec{x_2})) \rightsquigarrow \Phi_{S_1,\ldots,S_n}(\vec{x_1}, \vec{x_2})$ describes how to construct instances of the object property R from the source streams and relations S_i. The declarative form of Φ is:

$$\Phi_{S_1,\ldots,S_n}(\vec{x_1}, \vec{x_2}) = \exists \vec{y}.\Phi_{S_1,\ldots,S_k}(\vec{x_1}) \wedge \Phi_{S_{k+1},\ldots,S_n}(\vec{x_2}) \wedge p^{Join}_{S_1,\ldots,S_n}(\vec{v})$$

where $\Phi_{S_1,\ldots,S_k}, \Phi_{S_{k+1},\ldots,S_n}$ describe how to extract instances of C_1 and C_2 from the streams $S_1, \ldots, S_k$ and $S_{k+1}, \ldots, S_n$ respectively. The term p^{Join} is the set of predicates that denotes the join between the streams and relations $S_1, \ldots, S_n$.

Finally an expression $A(f^{Id}_C(\vec{x}), f^{Trf}_A(\vec{z})) \rightsquigarrow \Phi_{S_1,\ldots,S_n}(\vec{x}, \vec{z})$ describes how to construct instances of the datatype property A from the source streams and relations $S_1, \ldots, S_n$. The function f^{Trf}_A executes any transformation over the tuple of variables $\vec{z}$ to obtain the property value (e.g. arithmetic operations, or string operations). The declarative form of Φ in this case is:

$$\Phi_{S_1,\ldots,S_n}(\vec{x}, \vec{z}) = \exists \vec{y}.\Phi_{S_1,\ldots,S_k}(\vec{x}) \wedge \Phi_{S_{k+1},\ldots,S_n}(\vec{z}) \wedge p^{Join}_{S_1,\ldots,S_n}(\vec{v})$$

The definition follows the same idea as the previous one. The variables of $\vec{z}$ will contain the actual values that will be used to construct the datatype property value using the function f^{Trf}_A.

When a conjunctive query is issued against the global ontology, the processor first parses it and transforms it into an abstract syntax tree and then uses the expansion algorithm described in [BCGP04] (that is based on the PerfectRef algorithm of [CDGL$^+$05]) to produce an expanded conjunctive query based on the TBox of the ontology. Afterwards the rewritten query can be translated to an extended relational algebra.

A query $Q_{\mathcal{O}}(\vec{x})[t_s, t_e, \delta]$ is a conjunctive query with a window operator (where t_s, t_e are the start and end points of the window range and δ is the slide) in order to narrow the data set according to a given criteria. For a query:

$$Q_{\mathcal{O}}(\vec{x})[t_s, t_e, \delta] = (C_1(x) \wedge R(x, y) \wedge A(x, z))[t_s, t_e, \delta]$$

the translation is given by $\lambda(\Phi)$, following the mapping definition:

$$\lambda(\Phi_{S_1,\ldots,S_n}(\vec{x})[t_s,t_e,\delta]) = \pi_{p^{Proj}}(\bowtie_{p^{Join}} (\sigma_{p^{Sel}}(\omega_{t_s,t_e,\delta}S_1), \ldots, \sigma_{p^{Sel}}(\omega_{t_s,t_e,\delta}S_n)))$$

The expression denotes first a window operation $\omega_{t_s,t_e,\delta}$ over the relations or streams $S_1, \ldots, S_n$, with t_s, t_e, and δ being the time range and slide. A selection $\sigma_{p^{Sel}}$ is applied over the result, according to the conditions defined in the mapping. A multi-way join $\bowtie_{p^{Join}}$ is then applied to the selection, also based on the corresponding mapping definition. Finally a projection $\pi_{p^{Proj}}$ is applied over the results. For any conjunctive query with more atoms, the construction of the algebra expression will follow the same direct translation using the GAV approach.

2.4 Implementation and Execution: Walkthrough

The presented approach of providing ontology-based access to streaming data has been implemented as an extension to the ODEMAPSTER processor [BCGP04]. This implementation generates SNEEql queries that can be executed by the streaming query processor.

Consider the motivating example where a sensor network generates two streams `WindFolkestone` and `WindHernebay` of wind sensor measurements. The associated stored information about the sensors, e.g. location and type, are stored in a relation `Sensors`.

```
WindFolkestone: (sensorId INT PK, timestamp DATETIME PK, speed FLOAT, direction FLOAT)
WindHernebay: (sensorId INT PK, timestamp DATETIME PK, speed FLOAT, direction FLOAT)
Sensors: (sensorId INT PK, sensorName VARCHAR(45), lat FLOAT, long FLOAT)
```

Listing 3: Relational schema of the stream data source.

The schemas are presented in Listing 3. Also consider the following ontological view:

$$\begin{aligned} SpeedObservation &\sqsubseteq Observation \\ WindSpeedObservation &\sqsubseteq SpeedObservation \\ SpeedObservation &\sqsubseteq \exists observationResult \\ SpeedObservation &\sqsubseteq \exists observationResultTime \\ Observation &\sqsubseteq \exists isProducedBy.Sensor \\ Sensor &\sqsubseteq \exists hasLatitude \\ Sensor &\sqsubseteq \exists hasLongitude \end{aligned}$$

We can define an S_2O mapping that unifies the `WindFolkestone` and `WindHernebay` stream tuples into instances of a $WindSpeedObservation$ concept. Listing 4 is an extract of the S_2O mapping document concerning the $WindSpeedObservation$. The mapping extract defines how to construct the $WindSpeedObservation$ and $Sensor$ class instances from the streams `WindFolkestone` and `WindHernebay` and the `Sensors` table:

$$\Psi_{WindSpeedObservation} \rightsquigarrow \Phi^{\texttt{WindFolkestone}}_{\texttt{WindHernebay}}$$

$$\Psi_{Sensor} \rightsquigarrow \Phi_{\texttt{Sensors}}$$

It needs to perform a union operation over both streams, as specified in the mapping document, and assign an alias name to this union (`SensorWind`). In the case of the $WindSpeedObservation$ the function $f^{Id}_{WindSpeedObservation}$ produces the URI's of the instances by concatenating the `sensorId` and `timestamp` attributes.

```
conceptmap-def WindSpeedObservation
  virtualStream <http://semsorgrid4env.eu/SensorReadings.srdf>
  uri-as
      concat('http://semsorgrid4env.eu/WindSpeedObservation_', SensorWind.sensorId,
          SensorWind.timestamp)
  union
    name SensorWind
    has-stream WindFolkestone
    has-stream WindHernebay
  described-by
    attributemap-def observationResult
      operation constant has-column SensorWind.speed
    attributemap-def observationResultTime
      operation constant has-column SensorWind.timestamp
    dbrelationmap-def isProducedBy
      toConcept Sensor
      joins-via
        condition equals
          has-column Sensors.sensorId
          has-column SensorWind.sensorId

conceptmap-def Sensor
  uri-as
    concat('http://semsorgrid4env.eu/Sensor_',Sensors.sensorId)
  described-by
    attributemap-def hasLatitude
      operation constant has-column Sensors.lat
    attributemap-def hasLongitude
      operation constant has-column Sensors.lon
```

Listing 4: S_2O mapping from the data streams `WindFolkestone` and `WindHernebay` to the ontology concepts $WindSpeedObservation$.

```
PREFIX flood: <http://www.ssg4env.eu/floodObservation#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT RSTREAM ?speed ?time ?lat ?lon
FROM STREAM <www.ssg4env.eu/SensorReadings.srdf> [FROM NOW - 10 MINUTES TO NOW STEP 1
    MINUTE]
WHERE {
    ?WindSpeed a flood:WindSpeedObservation;
        flood:observationResult ?speed;
        flood:observationResultTime ?time;
        flood:isProducedBy ?Sensor.
    ?Sensor a flood:Sensor;
        flood:hasLatitute ?lat;
        flood:hasLonitude ?lon.
}
```

Listing 5: SPARQL$_{\text{Stream}}$ query which every minute returns the wind speed for the last ten minutes.

Now we can pose a query over the ontology using SPARQL$_{\text{Stream}}$, for example to obtain the wind speed measurements taken in the last 10 minutes (See the query in Listing 5).

A class query atom $WindSpeedObservation(x)$ and the $observationResultTime(x,t)$ and $observationResult(x,z)$ datatype property atoms can be extracted from the SPARQL$_{\text{Stream}}$ query.
The window specification:

$$[t_s = \mathsf{NOW} - 10, t_e = \mathsf{NOW}, \delta = 1]$$

is also obtained[8]. The S$_2$O mapping defines that *WindSpeedObservation* instances are generated based on the `sensorId` and `timestamp` attributes of the `WindFolkestone` and `WindHernebay` streams, using a concatenation function to generate each instance URI. Similarly the S$_2$O mapping defines that *observationResult* properties are generated from the values of the `speed` attribute of the streams.
The processor will evaluate this as:

$$\lambda\left(\Phi_{\substack{\texttt{WindFolkestone}\\ \texttt{WindHernebay}}} \begin{pmatrix} x_{\texttt{sensorId}} \\ x_{\texttt{timestamp}} \\ z_{\texttt{speed}} \end{pmatrix} [now{-}10,now,1]\right) =$$
$$\pi_{\substack{\texttt{sensorId}\\ \texttt{timestamp}\\ \texttt{speed}}} \left(\cup \begin{pmatrix} \omega_{now-10,now,1}(\texttt{WindFolkestone}) \\ \omega_{now-10,now,1}(\texttt{WindHernebay}) \end{pmatrix} \right)$$

Similarly the evaluator will compute the *Sensor* instances needed for the join that is produced in order to generate the *isProducedBy* object property. The query generated in the SNEEql language is shown in Listing 6. The relational answer stream that results from evaluating the query in Listing 6 are transformed by the *Data Transformation* module depicted in Figure 1.1 according to the S$_2$O mappings. This results in a stream of tagged triples which are instances of the class *WindSpeedObservation*. According to the select statement of the SPARQL$_{\text{Stream}}$ query, the result in this case will be a set of bound variables.

```
SELECT RSTREAM concat('http://ssg4env.eu#WindSpeedObservation_',
        WindSensor.sensorId,WindSensor.timestamp) AS id,
        WindSensor.speed AS speed, Sensors.lat, Sensors.lon
FROM ( SELECT sensorId, timestamp
        FROM WindFolkestone [FROM NOW - 10 MINUTES TO NOW SLIDE 1 MINUTE]
        UNION
        SELECT sensorId, timestamp
        FROM WindHernebay [FROM NOW - 10 MINUTES TO NOW SLIDE 1 MINUTE]
) AS WindSensor, Sensors WHERE WindSensor.sensorId = Sensors.sensorId;
```

Listing 6: The SNEEql query that is generated for the input query in Listing 5.

[8] For the simplicity of presentation, we assume that the system rewrites all time specifications to minutes. The implemented system uses milliseconds as the common time unit.

3. Source Code

The code delivered comprises both the Semantic Integrator Java library and the IQS web service described in Section 1.3.

3.1 Semantic Integrator

The Semantic Integrator code can be located in its open source SVN repository: `https://semanticstreams.googlecode.com/svn/stream-translator/trunk`. It can be found under the Java package `es.upm.fi.dia.oeg.integration`.

The code follows a Maven[1] project organization:

- `src/main/java`: Java source files.
- `src/main/resources`: Application resources.
- `src/main/javacc`: $\text{SPARQL}_{\text{Stream}}$ parser grammar.
- `src/test/java`: Test source files.
- `src/test/resources`: Test resources.

3.1.1 Dependencies

Table 3.1) specifies the external dependencies for the Semantic Integrator.

Group	Artifact	Version
antlr	antlr	2.7.5
com.hp.hpl.jena	iri	0.7
com.hp.hpl.jena	jena	2.6.4
com.ibm.icu	icu4j	3.4.4
commons-digester	commons-digester	2.0
commons-io	commons-io	2.0.1
es.upm.fi.dia.oeg.integrator	morph	0.0.1-SNAPSHOT
es.upm.fi.dia.oeg.sparql	resultbindings	0.0.1
jakarta-regexp	jakarta-regexp	1.4
junit	junit	4.7
log4j	log4j	1.2.12
org.apache.cayenne	cayenne-nodeps	2.0.4
org.apache.lucene	lucene-core	2.3.1
org.easymock	easymock	3.0
org.nfunk	jep	2.4.0
uk.ac.manchester.cs.snee	snee	1.5.0

Table 3.1: Semantic Integrator dependencies.

[1] http://maven.apache.org, accessed 15 February 2011.

3.1.2 Configuration

Integrator settings The Semantic Integrator is configured using a *Properties* file with a set of key-value pairs. This file is by default located at `config/config.properties` though it can be initialized from elsewhere. The main settings are:

- `integrator.queryexecutor.adapter`: Specify the streaming data processor (e.g SNEE).
- `integrator.repository.provider`: Specify the type of storage of the integrator resources (e.g. memory-only, file system).
- `integrator.repository.url`: Specify the location of the registered mappings and configured integrated resources metadata.

Stream engine settings In case of hosting an embedded stream processing engine (e.g. SNEE), the engine settings must also be set. For further details about the SNEE configuration refer to deliverable D2.2v2[GGF+11].

Logging The system uses standard `log4j` logging mechanisms and is configured using a file named `log4j.properties`.

3.1.3 API

Table 3.2 provides the Java API for the Semantic Integrator.

Constructor Summary	
	`SemanticIntegrator(Properties properties)` Create a SemanticIntegrator configured according to the given properties.
Method Summary	
void	addSource(String url) Register a new streaming data source, so that it can be used to create an integrated data source.
DataSourceMetadata	integrateAs(List<DataSourceMetadata> sources, String integratedSourceName, MappingDocumentMetadata mapping) Register a new integrated data source, based on the list of original streaming data sources. The new source ontological schema is mapped to the underlying data sources schemas.
DataSourceMetadata	pullQueryFactory(String integratedSourceName, QueryDocument query) Register a pull query against the integrated data source.
ResponseDocument	pullData(String pullSourceName) Pull data from a registered pull data source.
ResponseDocument	pullLatestData(String pullSourceName) Pull latest data form a registered pull data source.
DataSourceMetadata	retrieveIntegratedDataSource(String integratedSourceName) Retrieve the metadata for the specified integrated data source.
Collection<DataSourceMetadata>	retrieveIntegratedDataSourceCollection() Retrieve the metadata for all integrated data sources.
DataSourceMetadata	retrievePullDataSource(String pullDataSourceName) Retrieve the metadata for the specified pull data source.
Collection<DataSourceMetadata>	retrievePullDataSourceCollection() Retrieve the metadata for all pull data sources.
void	removeIntegratedDataSource(String integratedSourceName) Remove the specified integrated data source.
void	removePullDataSource() Remove the specified pull data source.

Table 3.2: Java API of the SemanticIntegrator.

3.2 Integration and Query Service

The IQS can be found in the SemSorGrid4Env project SVN repository: `https://ssg4env.techideas.net/repos/IntegrationQueryService/trunk`. The java source files are organized under the `eu.semsorgrid4env.service.iqs` package.

The IQS project is subdivided in two sub-projects: `iqs-ws` and `iqs-client`. The first is the web service implementation itself and the latter is a sample web service client project that can be used for integration testing.

The code follows a Maven project organization:

- `src/main/java`: Java source files.
- `src/main/resources`: Application resources.
- `src/main/webapp`: Web application resources.
- `src/test/java`: Test source files.
- `src/test/resources`: Test resources.

3.2.1 Integration and Query Service WSDL

In addition to the service itself, the Web Service description is specified in WSDL files on a separate project called `iqs-wsdl`. It can be found in the SemSorGrid4Env project SVN: `https://ssg4env.techideas.net/repos/iqs-wsdl/trunk`. Therefore this project is one of the dependencies of IQS.

3.2.2 Dependencies

Table 3.3 provides the external dependencies of the Integration and Query Service.

Group	Artifact	Version
commons-io	commons-io	2.0.1
es.upm.fi.dia.oeg.integrator	iqs-wsdl	0.0.1
es.upm.fi.dia.oeg.integrator	stream-translator	0.1.1
eu.semsorgrid4env.service	SchemaMetadata	0.0.2-SNAPSHOT
javax.servlet	servlet-api	2.5
junit	junit	4.8.1
log4j	log4j	1.2.16
org.apache.cxf	cxf-rt-frontend-jaxws	2.3.2
org.apache.cxf	cxf-rt-transports-http	2.3.2

Table 3.3: IQS dependencies.

3.2.3 Configuration

As the implementation of the IQS service is based on the Semantic Integrator library, it requires its configuration, described in Section 3.1.2.

Additionally, it requires the Web Services endpoint configuration, for the three interfaces: Integration, Query and Pull Stream. The endpoints are configured in the `main/webapp/WEB-INF/beans.xml` file.

4. Source Data

In this section we describe the data sources we have worked with and we also provide the sample mapping files and queries run against the CCO data exposed through the SNEE Web service.

The IQS service can be configured to use either SNEE as an embedded query execution engine or a SemSorGrid4Env web service that exposes the query and pull interfaces for streaming data.

In both scenarios the streaming data source can be configured to operate over arbitrary stream data schemas and data produced by a generator. The generator can produce tuples at a configurable rate.

4.1 CCO Data Source

Alternatively, and for its use in the context of the SemSorGrid4Env project, a SNEE Web Service implementation of the query and pull interfaces has been deployed as part of Work Package 2 [GGF+11]. This service acts as a distributed query processing engine for the streams published by the Channel Coastal Observatory (CCO).

We have worked with a series of streams, which can be classified in three groups: meteorological, wave streams and tide streams. For each sensor location, a different extent exists. So for instance, a tide stream for a deployment in Hernebay is named `envdata_hernebay_tide`. For a complete list of streams refer to Tables 4.1, 4.3 and 4.2.

Meteorological
envdata_deal_met
envdata_hernebay_met
envdata_looebay_met
envdata_worthing_met
envdata_arunplatform_met
envdata_swanagepier_met
envdata_sandownpier_met
envdata_weymouth_met
envdata_westbaypier_met
envdata_teignmouthpier_met
envdata_folkestone_met
envdata_lymington_met

Table 4.1: Channel Coastal Observatory meteorological streams.

Tide
envdata_lymington_tide
envdata_hernebay_tide
envdata_deal_tide
envdata_teignmouthpier_tide
envdata_swanagepier_tide
envdata_sandownpier_tide
envdata_westbaypier_tide

Table 4.2: Channel Coastal Observatory tide streams.

Wave	
envdata_milford	envdata_perranporth
envdata_pevenseybay	envdata_goodwin
envdata_torbay	envdata_rustington
envdata_bidefordbay	envdata_folkestone
envdata_boscombe	envdata_penzance
envdata_weymouth	envdata_rye
envdata_westonbay	envdata_haylingisland
envdata_hornsea	envdata_rhylflats
envdata_chesil	envdata_westbay
envdata_looebay	envdata_startbay
envdata_sandownbay	envdata_minehead
envdata_seaford	envdata_bracklesham

Table 4.3: Channel Coastal Observatory wave streams.

For the schemas of these streams refer to Tables 4.4, 4.5 and 4.6. Notice that the location of the sensor that provides the information is in the stream extent name.

Meteorological Stream Schema	
Timestamp	integer
DateTime	string
TAir	float
WDir	float
GustSpeed	float
WindSpeed	float
AirPressure	float

Table 4.4: Meteorological stream schema.

Wave Stream Schema	
Timestamp	integer
DateTime	string
Lat	float
Lon	float
Hmax	float
Tp	float
Dirp	float
Sprp	float
Tz	float
Hs	float
TSea	float

Table 4.5: Wave stream schema.

Tide Stream Schema	
Timestamp	integer
DateTime	string
Observed	float
Tz	float
Hs	float
Hmax	float
Tp	float

Table 4.6: Tide stream schema.

4.2 Ontologies

The source schemas have been mapped to the ontologies produced by WP4 [GCHC11]. They are available publicly at `http://semsorgrid4env.eu/ontologies/`. Specifically the queries exposed by the integrator require the classes and properties of the Sensor Network and Observation ontology: `http://purl.oclc.org/NET/ssnx/ssn`. Additionally for the coastal vocabulary, the Coastal Defenses ontology has been used: `http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl`.

4.3 Mapping Files

The following sample mapping in S_2O format specifies how to construct an Observation in terms of the ontologies cited in Section 4.2, based on a wave stream schema `envdata_milford`.

```
<conceptmap-def  id="Observation_wave" name="http://www.semsorgrid4env.eu/ontologies/CoastalDefences.
     owl#Observation" virtualStream="http://www.semsorgrid4env/ccometeo.srdf">
 <uri-as>
  <operation oper-id="concat">
    <arg-restriction on-param="string1">
     <has-transform>
      <operation oper-id="extentname"><arg-restriction on-param="value1"/></operation>
     </has-transform>
    </arg-restriction>

    <arg-restriction on-param="string2">
     <has-transform>
      <operation oper-id="concat">
       <arg-restriction on-param="string1"><has-value>_</has-value></arg-restriction>
       <arg-restriction on-param="string2"><has-column>envdata_milford.DateTime</has-column></arg-
            restriction>
      </operation>
     </has-transform>
    </arg-restriction>
  </operation>
 </uri-as>

 <described-by>

  <dbrelationmap-def name="http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#
       observedProperty" toConcept="http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#
       WaveHeight" condition="true">
  </dbrelationmap-def>

  <dbrelationmap-def name="http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#
       featureOfInterest" toConcept="http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#
       Feature" condition="true">
  </dbrelationmap-def>


  <attributemap-def name="http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#
       observationResultTime" dataType="xsd:long">
    <selector><aftertransform>
      <operation oper-id="constant">
       <arg-restriction on-param="const-val"><has-column>envdata_milford.timestamp</has-column></arg-
            restriction>
      </operation>
     </aftertransform></selector>
  </attributemap-def>

  <attributemap-def name="http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#
       observationResultLatitude" dataType="xsd:double">
   <selector><aftertransform>
     <operation oper-id="constant">
      <arg-restriction on-param="const-val"><has-column>envdata_milford.Lat</has-column></arg-
           restriction>
     </operation>
    </aftertransform></selector>
  </attributemap-def>

  <attributemap-def name="http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#
       observationResultLongitude" dataType="xsd:double">
   <selector><aftertransform>
     <operation oper-id="constant">
      <arg-restriction on-param="const-val"><has-column>envdata_milford.Lon</has-column></arg-
           restriction>
     </operation>
    </aftertransform></selector>
  </attributemap-def>

  <attributemap-def name="http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#
       observationResult" dataType="xsd:double">
   <selector><aftertransform>
     <operation oper-id="constant">
      <arg-restriction on-param="const-val"><has-column>envdata_milford.Hs</has-column></arg-
           restriction>
     </operation>
    </aftertransform></selector>
  </attributemap-def>

 </described-by>
</conceptmap-def>
```

Listing 7: The sample mapping in the S_2O format.

5. Installation and Execution

In this section we describe how to build, configure and deploy the IQS service.

5.1 Build

The IQS can be downloaded from the project SVN repository mentioned in Section 3.1(`https://ssg4env.techideas.net/repos/IntegrationQueryService/trunk`). The following software is required to build the code:

- Java Development Kit (version 1.6): `http://www.java.com/`.
- Apache Maven (2.x): `http://maven.apache.org`.
- A subversion SVN client.
- Access privileges to the SemSorGrid4Env Maven repository: `http://ssg4env.techideas.net:8180/archiva/repository/internal`.

The following command downloads the code form the repository to a given directory < **dir** >:

```
>svn co https://ssg4env.techideas.net/repos/IntegrationQueryService/trunk
<dir>
```

Then in the <dir> directory the code and the Maven pom.xml configuration file are located:

```
>cd <dir>
```

To build the `war` deployment file, it suffices to execute the Maven package task, in the same directory where the `pom` file is located:

```
>mvn package
```

All dependencies are automatically downloaded and managed by Maven, as they are already declared in the `pom.xml` file. The resulting war will be located in the `iqs-ws/target` directory (e.g. `iqs-ws-0.0.2.war`).

All available versions of the IQS `war` file are periodically and automatically deployed at the SemSorGrid4Env Maven repository: `http://ssg4env.techideas.net:8180/archiva/repository/internal` The `war` file can be directly downloaded form this repository, under this Maven group identifier: `eu.semsorgrid4env` and this Maven artifact identifier: `iqs-ws`.

5.1.1 Client code

In order to exemplify the use of the IQS service, we provide a client code project in the SemSorGrid4Env SVN. The location of the sample code is `https://semsorgrid.techideas.net/repos/IntegrationQueryService/trunk/iqs-client`. This command downloads the code form the SVN repository:

```
>svn co https://ssg4env.techideas.net/repos/IntegrationQueryService/trunk/iqs-client
<dir>
```

This project can be built with Maven, to do so just enter in the `iqs-client` directory, where the `pom.xml` file is located. Then execute the Maven package command:

```
>mvn package
```

This client code is already configured to automatically generate the Web Service code and includes all necessary library dependencies. Developers of integrator service client applications can use this code as a starting point for their development.

5.2 Configuration

Configuration parameters of the IQS must be set in the `WEB-INF/classes/config/config.properties` file inside the `war` file. The parameters have been described in Section 3.1.2.

5.2.1 Configuring a IQS service

The IQS can be configured to use either the SNEE processor natively or a SemSorGrid4Env streaming data service that exposes the Query and Pull interfaces. In order to specify the stream processor, the setting `integrator.queryexecutor.adapter` in the `config.properties` can be set to either:

- `ssg4e`: SemSorGrid4Env Query Service
- `snee`: SNEE processor

In case of the `ssg4e` option, the url of the pull and query services can be set with the following parameters:

- `integrator.queryexecutor.adapter.ssg4e.pull`
- `integrator.queryexecutor.adapter.ssg4e.query`

The IQS requires a local repository where the mapping files and integrated sources metadata is stored. By default this is stored in the file system and requires to be set in the configuration, for instance: `integrator.repository.url = file:///etc/ssg4env/integrator-repository/`

5.3 Execution

In order to install and execute IQS, it is required to have a Servlet Container. We have used Tomcat 6 [1] for this purpose. To install the IQS `war` file in Tomcat, it must be placed in the `webapps` folder.

In the following sections we highlight the code for performing different aspects of the functionality.

[1] `http://tomcat.apache.org/`, accessed 15 February 2011.

5.3.1 Client Configuration

There are three interfaces provided by IQS and available as three SOAP services with their correspondent WSDL descriptors. Once the service has been deployed in tomcat, the WSDLs can be accessed at the URL: `http://SERVER_NAME:PORT/iqs-ws-VERSION/services/`, where `SERVER_NAME` and `PORT` are replaced by the settings for the deployment of tomcat on your server, and `VERSION` is the version number of IQS. For example `http://forgy.dia.fi.upm.es:8080/iqs-ws-0.0.2/services/`.

Notice that the three interfaces are: IntegrationInterface, PullStreamInterface and SPARQLQueryInterface, and each of them has a number of operations. To use these, it suffices to instantiate the client proxy classes for the corresponding interface.

For example for the PullStreamInterface, first we need to initiate the `PullStreamService`, providing the server URI, and then obtain a `PullStreamInterface` service proxy (Listing 8):

```
String host = "http://forgy.dia.fi.upm.es:8080/iqs-ws-0.0.2";
PullStreamService pss = new PullStreamService(
                new URI(host+"/services/PullStreamInterface?wsdl").toURL(),
                PULL_SERVICE_NAME);
PullStreamInterface pullInterface = pss.getPullStreamInterface();
```

Listing 8: Initializing the pull service proxy.

Then we can access the operations of the PullStreamInterface from this instance. The same initialization process can be applied for the other interfaces (Listing 9):

```
IntegrationQueryService iqs = new IntegrationQueryService(
                new URI(host+"/services/QueryInterface?wsdl").toURL(),
                QUERY_SERVICE_NAME);
SPARQLQueryInterface queryInterface = iqs.getQueryInterface();

IntegrationMappingService ims = new IntegrationMappingService(
                new URI(servicehost+"/services/IntegrationInterface?wsdl").toURL(),
                INTEGRATION_SERVICE_NAME);
IntegrationInterface integrationInterface = ims.getIntegrationInterface();
```

Listing 9: Initializing the integration and query service proxies.

5.3.2 Listing the Registered Integrated Resources

All registered integrated data resources have their metadata stored in the internal IQS metadata registry. Users may be interested in knowing which data sources have already been configured. The following code in Listing 10 iterates over the complete list of integrated resource addresses.

```
GetResourceListRequest request = new GetResourceListRequest();
GetResourceListResponse list = queryInterface.getResourceList(request);
for (DataResourceAddressType ad: list.getDataResourceAddress())
{
  System.out.println("Resource Address: "+ad.getAddress().getValue());
}
```

Listing 10: Listing all registered integrated data sources.

5.3.3 Retrieving an Integrated Resource Property Document

To obtain the metadata of a particular integrated resource it suffices to provide the resource abstract name and invoke the `getSPARQLPropertyDocument` operation as in Listing 11. The metadata is exposed as a Property Document that contains information such as the endpoint addresses, the kind of data it exposes, the schemas, relational or ontological, etc.

```
GetDataResourcePropertyDocumentRequest request =
      new GetDataResourcePropertyDocumentRequest();
request.setDataResourceAbstractName(name);
SPARQLPropertyDocumentType response =
      queryInterface.getSPARQLPropertyDocument(request );
System.out.println(response.getDataResourceAbstractName());
```

Listing 11: Obtaining an integrated resource property document.

5.3.4 Integrating Data Resources

In order to integrate data resources and expose them through an ontological schema, it is necessary to: i) provide the data source addresses, ii) provide an integrated data source name and iii) provide a mapping document that links the source schemas to the ontology concepts. The `IntegrateAs` operation is devised for that purpose and in the example of Listing 12 we show how to expose the multiple streams available from the data source `"http://webgis1.geodata.soton.ac.uk:8080/CCO/services/PullStream?wsdl`, under an ontological view, using a mapping file `ccoMapping.r2r`. The new resource is given the name `ssg4env:iqs:ccoWaveStreamResource`, which can be subsequently used to issue continuous queries over it.

```
IntegrateAsRequestType resourceRequest = new IntegrateAsRequestType();
resourceRequest.setIntegratedSourceName("ssg4env:iqs:ccoWaveStreamResource");
DataResourceAddressListType list = new DataResourceAddressListType();

for (String source:sources)
{
  DataResourceAddressType address = new DataResourceAddressType();
  AttributedURIType uri = new AttributedURIType();
  uri.setValue("http://webgis1.geodata.soton.ac.uk:8080/CCO/services/PullStream?wsdl");
  address.setAddress(uri);
  MetadataType metadata = new MetadataType();
  address.setMetadata(metadata);
  list.getDataResourceAddress().add(address);
}
resourceRequest.setDataResourceAddressList(list );
InputStream is = IntegrationServiceClient.class.getClassLoader().getResourceAsStream("
    ccoMapping.r2r");
String mappingDoc = IOUtils.toString(is);
MappingDocumentType mapping = new MappingDocumentType();
mapping.setBody(mappingDoc);
mapping.setLanguage("http://www.w3.org/ns/r2rml");
mapping.setMappingName(mappingName);
resourceRequest.setMappingDocument(mapping);

DataResourceAddressType resource = integrationInterface.integrateAs(resourceRequest );
logger.info("The address "+resource.getAddress().getValue());
```

Listing 12: Integrating data resources using a mapping document.

5.3.5 Registering a Query to an Integrated Data Source

Once an integrated data resource has been configured, queries can be run against it. Each query launched will produce a new data resource that can be pulled for new stream data periodically. To issue these queries, we can make use of the Query Interface as in Listing 13. The main parameters needed are the integrated resource name and the query itself, expressed in the $\text{SPARQL}_{\text{Stream}}$ language. Notice that the result of this query operation is not a result set but an address for a resource from which the result set can be periodically polled for new values.

```
SPARQLExecuteFactoryRequest sparqlExecuteFactoryRequest = new
    SPARQLExecuteFactoryRequest();
sparqlExecuteFactoryRequest.setDataResourceAbstractName("ssg4env:iqs:
    ccoWaveStreamResource");

SPARQLQueryRequestType spQueryRequest = new SPARQLQueryRequestType();
spQueryRequest.setQuery(sparqlQuery);
sparqlExecuteFactoryRequest.setSPARQLQueryRequest(spQueryRequest);

DataResourceAddressType response = queryInterface.sparqlExecuteFactory(
    sparqlExecuteFactoryRequest);
logger.info("Address received from service:"+ response.getAddress().getValue());
```

Listing 13: Querying an integrated resource.

The `sparqlQuery` string variable is a query like the one shown in Listing 14, which obtains the wave height and timestamps in the last ten minutes.

```
PREFIX flood: <http://www.ssg4env.eu/floodObservation#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT RSTREAM ?height ?time
FROM NAMED STREAM <www.ssg4env.eu/SensorReadings.srdf> [NOW - 10 MINUTES TO NOW STEP 1
    MINUTE]
WHERE {
    ?WaveObservation a flood:Observation;
        flood:observationResult ?height;
        flood:observationResultTime ?time;
      flood:observedProperty flood:WaveHeight.
}
```

Listing 14: $\text{SPARQL}_{\text{Stream}}$ query which every minute returns the wave height for the last ten minutes.

5.3.6 Listing the Registered Pull Resources

It is possible to list all pull data resources created after a query has been registered. The result is a list of the resources addresses, as shown in Listing 15.

```
GetResourceListRequest request = new GetResourceListRequest();
GetResourceListResponse response = pullInterface.getResourceList(request);
for (DataResourceAddressType ad: response.getDataResourceAddress())
{
  System.out.println("Pull Address: "+ ad.getAddress().getValue());
}
```

Listing 15: Listing all available pull data resources.

5.3.7 Pulling data from Generated Resource

Finally we show how to pull data from a resource created after registering a continuous query. In the example in Listing 16 we use the `getStreamNewestItem` operation to obtain the latest data values captured by the query. The result is returned in this case as SPARQL bound variables, encapsulated by a Java `Sparql` class. Notice that we only need to indicate the abstract name of the pull resource, the query is no longer necessary, it is configured only once.

```
GetStreamNewestItemRequest pullRequest = new GetStreamNewestItemRequest();
pullRequest.setDataResourceAbstractName(urn);

GenericQueryResponse pullResponse;
pullResponse = pullInterface.getStreamNewestItem(pullRequest);
DatasetType dt = (DatasetType) pullResponse.getDataset();
Sparql sparqlResult = (Sparql) dt.getDatasetData().getContent().get(0);
```

Listing 16: Pulling data form a resource created with a continuous query.

6. Tests

In this chapter we describe the different tests defined for the Semantic Integrator, including unit and integration tests.

6.1 Unit Tests

These tests are the minimal units of a software whose behavior can be validated. They have been defined by the developers themselves and are integrated to the automatic build process of the components.

The Unit Tests can be found in the `test/java/*Test.java` in both the Semantic Integrator project and the IQS project, according to Maven conventions.

The test coverage of the semantic integrator is summarized in Table 6.1.

Package	Line coverage
es.upm.fi.dia.oeg.common	76%
es.upm.fi.dia.oeg.engine	0%
es.upm.fi.dia.oeg.integration	14%
es.upm.fi.dia.oeg.integration.algebra	72%
es.upm.fi.dia.oeg.integration.algebra.xpr	84%
es.upm.fi.dia.oeg.integration.metadata	42%
es.upm.fi.dia.oeg.integration.registry	23%
es.upm.fi.dia.oeg.integration.translation	86%
es.upm.fi.dia.oeg.integration.algebra.xpr	84%
es.upm.fi.dia.oeg.integration.algebra.xpr	84%
es.upm.fi.dia.oeg.integration.r2o	0%
es.upm.fi.dia.oeg.integration.sparql.lang.sparqlstr	34%
es.upm.fi.oeg.sparql	22%
es.upm.fi.oeg.sparql.syntax	31%

Table 6.1: Semantic Integrator test coverage

The test coverage of IQS is summarized in Table 6.2.

Package	Line coverage
es.upm.fi.dia.oeg.integration.web	39%
eu.semsorgrid4env.service.iqs.impl	12%
eu.semsorgrid4env.service.iqs.integration	32%
eu.semsorgrid4env.service.iqs.query	0%
eu.semsorgrid4env.service.stream	0%

Table 6.2: IQS test coverage

Note that most of the classes uncovered in both tables 6.1 and 6.2 are either auto-generated code or legacy data.

6.2 Integration Tests

Integration tests are designed to test the interaction of different components inside the system. The following interactions have been identified:

- IQS and SNEE Web Service [GGR+09].
- IQS and Registry Web Service [KKK09].

Integration tests can be found in the `test/java/*IT.java` in both the Semantic Integrator project and the IQS project, according to Maven conventions. The integration test will run different scenarios in the two interaction pairs indicated above. The complete details of these tests will be presented in the WP4 evaluation deliverable.

Bibliography

[AAB+05] Daniel J Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Cetintemel, 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 *CIDR 2005*, 2005.

[ABB+06] 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*. Springer, 2006.

[BBCG10] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, and Michael Grossniklaus. An execution environment for C-SPARQL queries. In *EDBT 2010*, pages 441–452, Lausanne, Switzerland, March 2010.

[BC07] Christian Bizer and Richard Cyganiak. D2RQ . Lessons Learned. W3C Workshop on RDF Access to Relational Databases, October 2007.

[BCGP04] Jesús Barrasa, Oscar Corcho, and Asunción Gómez-Pérez. R2O, an extensible and semantically based database-to-ontology mapping language. In *SWDB2004*, pages 1069–1070, 2004.

[BGFP08] Christian Y. Brenninkmeijer, Ixent Galpin, Alvaro A. Fernandes, and Norman W. Paton. A semantics for a query language over sensors, streams and relations. In *BNCOD 2008*, pages 87–99, 2008.

[BGJ08] Andre Bolles, Marco Grawunder, and Jonas Jacobi. Streaming SPARQL - extending SPARQL to process data streams. In *ESWC 2008*, pages 448–462, 2008.

[CC09] Jean-Paul Calbimonte and Oscar Corcho. Implementation and deployment of the SemSorGrid4Env ontology-based data integration service, Phase I. Deliverable. D4.2v1, SemSorGrid4Env, December 2009.

[CCG09] Jean-Paul Calbimonte, Oscar Corcho, and Alasdair J G Gray. Design of the SemSorGrid4Env ontology-based data integration model. Deliverable. D4.1, SemSorGrid4Env, August 2009.

[CCG10] Jean-Paul Calbimonte, Óscar Corcho, and Alasdair J G Gray. Enabling ontology-based access to streaming data sources. In *International Semantic Web Conference (1)*, volume 6496 of *Lecture Notes in Computer Science*, pages 96–111. Springer, 2010.

[CDGL+05] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. DL-Lite: Tractable description logics for ontologies. In *AAAI 2005*, pages 602–607, 2005.

[GBG+11] Ixent Galpin, Christian Y. A. Brenninkmeijer, Alasdair J. G. Gray, Farhana Jabeen, Alvaro A. A. Fernandes, and Norman W. Paton. SNEE: a query processor for wireless sensor networks. *Distributed and Parallel Databases*, 29(1-2):31–85, 2011.

[GCHC11] Raúl García-Castro, Chris Hill, and Oscar Corcho. Sensor network ontology suite v2. Deliverable. D4.3v2,SemSorGrid4Env, February 2011.

[GGF+10] 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, Jesús E Gabaldón, and Juan José Aparicio. SemSorGrid4Env architecture Phase II. Deliverable. D1.3v2, SemSorGrid4Env, December 2010.

[GGF+11] Alasdair J G Gray, Ixent Galpin, Alvaro A A Fernandes, Norman W Paton, Alexis Kotsifakos, George Valkanas, and Dimitrios Gunopulos. Implementation and deployment of data management and analysis, and the query processing components - Phase II. Deliverable. D2.2v2, SemSorGrid4Env, February 2011.

[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-DAI Streaming) specification. Technical report, University of Manchester, August 2009.

[GGR+09] Alasdair J G Gray, Ixent Galpin, Varadarajan Rajagopalan, Alvaro A A Fernandes, Norman W Paton, Alexis Kotsifakos, Dimitris Kotsakos, and Dimitrios Gunopulos. Implementation and deployment of data management and analysis, and the query processing components - Phase I. Deliverable. D2.2v1, SemSorGrid4Env, December 2009.

[KKK09] Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Implementation and deployment of the registry services – phase i. deliverable d3.3v1, semsorgrid4env, December 2009.

[Kos00] Donald Kossmann. The state of the art in distributed query processing. *ACM Computer Surveys*, 32(4):422–469, 2000.

[LT09] Lina Lubyte and Sergio Tessaris. Supporting the development of data wrapping ontologies. In *4th Asian Semantic Web Conference*, December 2009.

[MFHH05] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, 2005.

[PAG09] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems*, 34(3):1–45, 2009.

[PLC+08] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *Journal of Data Semantics*, 10:133–173, 2008.