Deliverable

D4.4

Evaluation of the Ontology-based Data Integration Service and the Ontologies

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

Josep Rodriguez
Techideas SL

August 31, 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 have proposed a Semantic Integration Service for streaming and stored data sources. This component is evaluated in this deliverable, as well as the ontology suite that is used in several tiers of the SemSorGrid4Env project.

The result of this twofold evaluation shows that the ontology network that has been developed suits the tasks for which it has been created, and in fact as a result of the evaluation some errors were detected and corrected accordingly on the existing versions. With respect to the evaluation of the Semantic Integrator, we have provided empirical results that show that our component is able to respond to semantic queries over sensor data in a timely fashion, even in settings that are more demanding to those of the real deployment needed in the SemSorGrid4Env project.
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 I contains material from [GGF+10] and [CCG11].
Abstract (for dissemination)

In the context of the SemSorGrid4Env project and its Work Package 4 (wp4), we have presented a semantic infrastructure for sensor data representation and integration. This included a Sensor Network Ontology suite [GCHC11], and the design, implementation and deployment of a Semantic Integration Service for streaming and stored data sources [CCG11]. The design of this service follows an ontology-based approach that extends previous works on data access for relational databases. This deliverable focuses on the evaluation of (i) the sensor network and coastal ontology suite (ii) the core component of the semantic Integrator: the Ontology-based streaming data access module.

For the ontology evaluation, we have followed an evaluation framework that establishes the different aspects of an ontology that can be evaluated and proposes different evaluation methods to be applied for each aspect. The aspects are: vocabulary, Syntax, Structure, Semantics, Representation and Context.

For the Semantic Integrator evaluation, we have conducted a series of experiments that validate different features and empirically show how the software performs under different settings.

The ontology-base streaming data access module, named Semantic Integrator, extends the odemapster engine that provided ontology-based access for databases, and uses the language for Stream-to-ontology mappings $s_2o$, or the syntactic variant of the r2rml language. An implementation of the proposed solution has been presented in [CCG11], so that we provided some evidence of the applicability of our approach. This deliverable evaluates the implemented component, through a set of tests that focus on the time response of continuous requests posed to data streams through $sparql$-stream queries.

Keywords
Semantic Integration, Query Translation, Evaluation
# 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:

<table>
<thead>
<tr>
<th>Partner</th>
<th>Acronym</th>
<th>Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universidad Politécnica de Madrid (Coordinator)</td>
<td>UPM</td>
<td>Prof. Dr. Asunción Gómez-Pérez</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Facultad de Informática</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Departamento de Inteligencia Artificial</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Campus de Montegancedo, sn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Boidilla del Monte 28660 Spain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+34-91 336-7439, +34-91 352-4819</td>
</tr>
<tr>
<td>The University of Manchester</td>
<td>UNIMAN</td>
<td>Prof. Norman Paton</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Department of Computer Science</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The University of Manchester</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Oxford Road, M13 9PL, United Kingdom</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+44-161-275-69 10, +44-161-275 62 04</td>
</tr>
<tr>
<td>National and Kapodistrian University of Athens</td>
<td>NKUA</td>
<td>Prof. Manolis Koubarakis</td>
</tr>
<tr>
<td></td>
<td></td>
<td>University Campus, Ilissia</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Athens GR-15784 Greece</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+30 210 7275213, +30 210 7275214</td>
</tr>
<tr>
<td>University of Southampton</td>
<td>SOTON</td>
<td>Dr. Kirk Martinez</td>
</tr>
<tr>
<td></td>
<td></td>
<td>University Road</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Southampton SO17 1BJ United Kingdom</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+44 23 80594491, +44 23 80595499</td>
</tr>
<tr>
<td>Deimos Space, S.L.U.</td>
<td>DMS</td>
<td>Mr. Agustín Izquierdo</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ronda de Poniente 19, Edif. Fiteni VI, P 2, 2º</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tres Cantos, Madrid – 28760 Spain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+34-91-8063450, +34-91-806-34-51</td>
</tr>
<tr>
<td>EMU Limited</td>
<td>EMU</td>
<td>Dr. Bruce Tomlinson</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mill Court, The Sawmills, Durley number 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Southampton, SO32 2EJ, United Kingdom</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+44 1489 860050, +44 1489 860051</td>
</tr>
<tr>
<td>Techideas Asesores Tecnológicos, S.L.</td>
<td>TI</td>
<td>Dr. Jesús E. Gabaldón</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C/ Marie Curie 8-14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>08042 Barcelona, Spain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+34.93.291.77.27, +44.93.291.76.00</td>
</tr>
</tbody>
</table>
# Contents

1 **Introduction**  1  
   1.1 Ontologies and Data Integration in SemSorGrid4Env  1  

2 **Evaluation of the SemSorGrid4Env ontologies**  2  
   2.1 Vocabulary evaluation  2  
   2.2 Syntax evaluation  4  
   2.3 Structure evaluation  4  
      2.3.1 Ontology complexity metrics  4  
      2.3.2 Anti-patterns  5  
      2.3.3 OntoClean  6  
   2.4 Semantics evaluation  6  
   2.5 Representation evaluation  6  
   2.6 Context evaluation  7  

3 **Evaluation of the Semantic Integrator Service**  8  
   3.1 Settings  8  
   3.2 Experiments  9  
      3.2.1 Single vs. Multiple Sensor Queries  9  
      3.2.2 Concurrent Continuous Queries  11  
      3.2.3 Window Operations  11  
      3.2.4 Join Queries  13  
      3.2.5 Query Translation  16  
   3.3 Stress Tests  19  
      3.3.1 Test settings  19  
      3.3.2 Platform  20  
      3.3.3 Test Results  20  

4 **Conclusions**  22  

A **Annex: Mappings**  23  

D4.4 Evaluation of the Ontology-based Data Integration Service and the Ontologies  v
List of Figures

3.1 Response times for a single-sensor query, for different tuple rates ................. 10
3.2 Response times for a single-sensor query, compared to a 26 sensor union query, for different tuple rates ................................................................. 12
3.3 Response times for different number of queries launched simultaneously, for different tuple rates ................................................................. 13
3.4 Response times for a time window queries, different window sizes ................. 14
3.5 Response times for time window queries, with different slide parameters ........ 15
3.6 Response time for a Join query, at different rates ........................................ 16
3.7 Execution times of a SNEEql query without query and data translation ......... 17
3.8 Execution time of the equivalent SPARQLStream query, including query and data translation ................................................................. 18
3.9 Average time elapsed during the query translation phase, for different queries ... 18
List of Tables

2.1 Reuse metrics for the SemSorGrid4Env ontologies. ............................................. 3
2.2 Structural metrics for the SemSorGrid4Env ontologies. The names of the ontologies
    have been shortened: AdditionalRegions = AR, CoastalDefences = CD, Role = RO,
    SchemaMetadata = SM, ServiceOntology=SO, SsnExtension=SE. .......................... 5
2.3 Ontological metrics for the SemSorGrid4Env ontologies. ..................................... 7

3.1 Stress test general statistics ................................................................................ 20
3.2 Stress test request statistics ................................................................................ 21
1. Introduction

This document describes the evaluation of the SemSorGrid4Env ontology suite, and the ontology-based semantic integration service of the SemSorGrid4Env architecture.

1.1 Ontologies and Data Integration in SemSorGrid4Env

The main objective of the SemSorGrid4Env 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 \cite{GGF10}. 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 evaluation of the Semantic Integrator service and the ontologies used throughout the project.

For the ontology evaluation, we have followed the ontology evaluation framework proposed by Vrandečić \cite{Vra10}, which establishes the different aspects of an ontology that can be evaluated and proposes different evaluation methods to be applied for each aspect. The aspects are: vocabulary, Syntax, Structure, Semantics, Representation and Context.

For the Semantic Integrator evaluation, we have conducted a series of experiments that validate different features and empirically show how the software performs under different settings.

The remainder of this deliverable document is organized as follows. In Section 2 we present the evaluation of the SemSorGrid4Env ontology suite. Then we provide details about the evaluation of the Semantic Integrator in Section 3 and finally we present our conclusions in Section 4.
2. Evaluation of the SemSorGrid4Env ontologies

This chapter presents how we have evaluated the SemSorGrid4Env ontologies [GCHC11] and the results obtained from that evaluation.

In order to evaluate the ontologies, we have followed the ontology evaluation framework proposed by Vrandečić [Vra10]. This framework, establishes the different aspects of an ontology that can be evaluated and proposes different evaluation methods to be applied for each aspect. These aspects are the following:

- **Vocabulary.** Evaluating the vocabulary aspect of an ontology means to evaluate the names used in the ontology. Names can be URI references or literals and this aspect deals with the different choices with regards to naming entities or leaving entities unnamed (i.e., blank nodes).

- **Syntax.** This aspect deals with evaluating the syntax in which the ontology is serialised. Ontologies can be described in a number of different surface syntaxes and often syntactic descriptions within a certain syntax can differ widely.

- **Structure.** Since an ontology can be described by an RDF graph, this aspect deals with the evaluation of the structure of such graph. The structure can vary highly even describing semantically the same ontology.

- **Semantics.** This aspect deals with the evaluation of the formal meaning of an ontology through different metrics that go beyond the structure of the ontology and exploit its semantics.

- **Representation.** This aspect captures the relation between the structure and the semantics. Representational aspects are usually evaluated by comparing metrics calculated on the RDF graph with features of the possible models as specified by the ontology.

- **Context.** This aspect is about the features of the ontology when compared with other artifacts in its environment; e.g., a data source that the ontology describes, a different representation of the data within the ontology, or formalized requirements for the ontology in form of competency questions or additional semantic constraints.

The next sections present how the SemSorGrid4Env ontologies were evaluated focusing on each of these aspects.

### 2.1 Vocabulary evaluation

The evaluation methods proposed for this aspect mainly consisted in checking for certain characteristics of the naming used in the ontologies. Next we present the results of these checks.

- **Used protocols.** All the URIs used in the ontologies are well-formed URIs. In our case, the only allowed protocol is HTTP, which is the one used in all the URIs.

- **Response codes.** While checking the response codes of the used URIs by making GET calls to them, we noticed that two URIs from the Ordnance Survey (OS) ontologies returned a 404 (Not Found) error; this was because the OS ontologies had changed and the references to the URIs of the old ontology versions could not be resolved anymore. After updating the ontologies to the last version of the OS ontologies, the problems with these two URIs disappeared and the response codes for all the URIs are 200 (OK) or 303 (See Other); besides, all the URIs with the same slash namespace return the same response code.
• **Content types.** The content types returned when making GET calls to the used URIs are correct.

• **Naming conventions.** Naming conventions are applied throughout all names of the namespaces used in the ontologies. This was expected, since the same naming and labelling scheme was used during the ontology development.

• **Name declarations.** While checking whether the type of every URI was declared (as an OWL class, individual, datatype, object or annotation property), we found that in two ontologies not every URI was declared: in the CoastalDefences ontology the three individuals that are reused from the SWEET units of measurement ontology and in the SchemaMetadata ontology all the individuals that define the SQLType enumerated class. Even if it is straightforward to identify these URIs as individuals (either by a person or by a reasoner), we added the declarations for those URIs to avoid having OWL Full ontologies.

• **Datatypes.** The datatypes used in the ontologies are `xsd:string`, `xsd:float`, `xsd:dateTime`, and `xsd:anyURI`. Even if the recommended practice is to prefer `xsd:integer` and `xsd:string` over other datatypes (because they are normative in OWL), we decided to provide a explicit semantics to some literal values.

• **Labels and comments.** The approach followed during ontology development has been to add labels to every class and comments to every class and property. This was made only for those entities defined in the ontologies and not for the ones reused from other ontologies.

• **Language tags.** All the literals in the ontologies include a language tag (i.e., `en`) except those that contain ontology version numbers.

• **Superfluous blank nodes.** The ontologies do not contain superfluous blank nodes. The only blank nodes included in them are those resulting from the serialisation of restrictions, unions and enumerations.

Apart from these checks, different ontology reuse metrics are proposed in [Vra10]:

- Number of namespaces used in the ontology ($N_{NS}$)
- Number of unique URIs used in the ontology ($N_{UN}$)
- Number of URI name references used in the ontology (every mention of a URI counts) ($N_{N}$)
- Ratio of name references to unique names ($R_{NU} = \frac{N_{NU}}{N_{N}}$)
- Ratio of unique URIs to namespaces ($R_{UNS} = \frac{N_{UN}}{N_{NS}}$)

Table 2.1 shows the reuse metrics obtained from the SemSorGrid4Env ontologies without including those names and namespaces that are part of the RDF(S) or OWL vocabularies.

<table>
<thead>
<tr>
<th>Ontology</th>
<th>$N_{NS}$</th>
<th>$N_{UN}$</th>
<th>$N_{N}$</th>
<th>$R_{NU}$</th>
<th>$R_{UNS}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdditionalRegions</td>
<td>5</td>
<td>26</td>
<td>100</td>
<td>0.26</td>
<td>5.2</td>
</tr>
<tr>
<td>CoastalDefences</td>
<td>8</td>
<td>94</td>
<td>503</td>
<td>0.19</td>
<td>11.75</td>
</tr>
<tr>
<td>Role</td>
<td>4</td>
<td>26</td>
<td>113</td>
<td>0.23</td>
<td>6.5</td>
</tr>
<tr>
<td>SchemaMetadata</td>
<td>2</td>
<td>65</td>
<td>293</td>
<td>0.22</td>
<td>32.5</td>
</tr>
<tr>
<td>ServiceOntology</td>
<td>7</td>
<td>56</td>
<td>189</td>
<td>0.30</td>
<td>8</td>
</tr>
<tr>
<td>SSnExtension</td>
<td>3</td>
<td>33</td>
<td>130</td>
<td>0.25</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 2.1: Reuse metrics for the SemSorGrid4Env ontologies.

Compared to the results obtained in [Vra10] for the ontologies in the Watson EA corpus (a subset of the Watson corpus that is part of the Billion Triple Challenge data[^1]), the values of $R_{NU}$ are

[^1]: http://challenge.semanticweb.org/
consistent with those of the ontologies in the corpus (lower than 0.5) but the values of $R_{UN}$ and $N_{NS}$ are not consistent, since they are expected to be lower than 5 and greater than 10, respectively.

Regarding the ratio of unique URIs to namespaces, expecting less than five unique URIs per namespace is expecting that ontologies have a low number of entities (five for an ontology with one namespace, ten for an ontology with two namespaces, and so on). Regarding the number of namespaces, expecting that ontologies include more than ten namespaces is expecting a lot of reuse in ontologies.

One issue here can be the way in which the expected values were calculated; this is not clear in [Vra10] and could significantly affect the values of the metrics. For example, creating an empty ontology with Protégé already creates five namespaces in the RDF/XML serialisation (i.e., rdf, rdfs, owl, xsd, and owl2xml). As mentioned above, we didn’t count these namespaces in our metrics but if we added five more namespaces to each ontology the values would be similar to the expected ones.

2.2 Syntax evaluation

The evaluations performed regarding the syntax aspect were performed over the RDF/XML serialisations of the ontologies:

- To check for problems with the serialisations using the RDF Validator[2] and the OWL Validator[3].
- To check whether the character encoding of the serialisations is UTF-8.

We found no problems in this aspect, as expected, since we used an ontology engineering tool during ontology development (i.e., Protégé).

2.3 Structure evaluation

The evaluation methods proposed for this aspect consisted in checking the ontology complexity and searching for problems in the ontology using anti-patterns and the OntoClean method.

2.3.1 Ontology complexity metrics

We counted how many times each ontology language feature appeared in the ontologies. This feature analysis allowed us to calculate the complexity of the language fragment used in each ontology by means of an online complexity calculation tool[4].

Table 2.2 shows the structural metrics that we obtained (we only show those features that appear at least in one ontology), the DL expressiveness, the OWL species, and the reasoning complexity for each ontology. As can be seen, most of the ontologies have Pspace complexity except the SsnExtension ontology that has ExpTime complexity.

http://www.w3.org/RDF/Validator/
http://www.mygrid.org.uk/OWL/Validator
http://www.cs.man.ac.uk/~ezolin/dl/
<table>
<thead>
<tr>
<th>Metrics</th>
<th>AR</th>
<th>CD</th>
<th>RO</th>
<th>SM</th>
<th>SO</th>
<th>SE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class</td>
<td>2</td>
<td>35</td>
<td>10</td>
<td>11</td>
<td>27</td>
<td>9</td>
</tr>
<tr>
<td>Object property</td>
<td>1</td>
<td>4</td>
<td>15</td>
<td>6</td>
<td>11</td>
<td>19</td>
</tr>
<tr>
<td>Data property</td>
<td></td>
<td></td>
<td>19</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Individual</td>
<td>22</td>
<td>53</td>
<td>28</td>
<td>15</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Class axioms**

| Sub-class               | 35 | 6  | 15 | 2  |
| Equivalent class        | 1  | 1  |    |    |

**Object property axioms**

| Sub-object property     |    |    | 9  |    |
| Functional object property | 2 |    |    |    |
| Object property domain  | 1  | 15 | 6  | 11 | 9  |
| Object property range   | 1  | 15 | 6  | 11 | 9  |

**Data property axioms**

| Sub-data property       |    |    | 2  |    |
| Data property domain    | 19 | 2  |    |    |
| Data property range     | 19 | 2  |    |    |

**Individual axioms**

| Class assertion         | 22 | 50 | 15 |    |
| Object property assertion| 10 | 42 |    |    |

**Annotation axioms**

| Entity annotation       | 22 | 151 | 25 | 45 | 36 | 26 |
| DL expressiveness       | AL | AL  | AL | ALCOF(D) | AL(D) | ALH(D) |
| OWL Species             | Lite | Lite | Lite | DL | Lite | Lite |
| Complexity              | Pspace | Pspace | Pspace | Pspace | Pspace | ExpTime |

Table 2.2: Structural metrics for the SemSorGrid4Env ontologies. The names of the ontologies have been shortened: AdditionalRegions = AR, CoastalDefences = CD, Role = RO, SchemaMetadata = SM, ServiceOntology=SO, SsnExtension=SE.

### 2.3.2 Anti-patterns

We executed two SPARQL queries over the ontologies to discover potentially problematic patterns. The two queries executed were those proposed in [Vra10] for detecting the anti-pattern of subsuming nothing:

```sparql
select ?a
where
{
  ?a rdfs:subClassOf owl:Nothing .
}
```

and for detecting the anti-pattern of skewed partitions:

```sparql
where
{
}
```

We did not find any of these anti-patterns in the ontologies.
2.3.3 OntoClean

We used the OntoClean method [GW02] to detect subsumption misuses in the ontologies. The method consists in tagging every class in an ontology using a certain of meta-properties (i.e., rigidity, unity, dependence, and identity) and then checking whether a set of subsumption constraints hold.

We applied the OntoClean method manually to the ontologies, since their number is low and there are few subsumptions in them. While applying the method we did not find any problem in the ontologies.

2.4 Semantics evaluation

The evaluation performed regarding the semantics aspect consisted in comparing the normalised class depth measure of the ontologies (i.e., the length of the longest subsumption path on the normalised version of the ontology) with the stable minimal depth of the ontology (which takes the open world assumption into account) [Vra10].

If these two values are the same there is no problem with the ontology, which is our case for all the ontologies that include class hierarchies.

2.5 Representation evaluation

The evaluation methods proposed for this aspect consist in calculating a set of ontological metrics and checking whether a set of constraints hold for them. These metrics are the following:

- **Explicitness of the subsumption hierarchy** ($ET(O)$), which is the ratio between the maximum depth of the taxonomy of an ontology and the maximum subsumption path length of the normalized version of the ontology.

- **Ratio of classes and class names** ($RC(O)$), which is the ratio between the number of classes in the normalized version of the ontology and the number of classes in the ontology.

- **Ratio of properties and property names** ($RP(O)$), which is the ratio between the number of properties in the normalized version of the ontology and the number of properties in the ontology.

The expected result for these three metrics is that their values are one (i.e., that the counts in the ontology and in its normalized version are the same) and this is the case for all the ontologies except in one case, as can be seen in Table 2.3 (in those ontologies without hierarchies the value of $ET(O)$ cannot be calculated).

In our case most of the values are one because almost every normalised ontology is identical to the original one. The only exception is in the SchemaMetadata ontology where one of the constraints from [Vra10] is not satisfied ($RC(O) > 1$) indicating “that not all interesting classes or properties have been given a name, i.e. the coverage of classes and properties with names may not be sufficient”.

The reason of this is that the SchemaMetadata ontology includes a complex class description (i.e., union) as domain of a property that, after normalisation, produces a new class name. It was a design decision not to include that artificial class name in the ontology and we defined the domain as a union of classes to increase clarity. Therefore, this constraint violation entails no defect in the ontology.
2.6 Context evaluation

For evaluating the context aspect we did not follow any of the methods proposed in [Vra10], since they rely on having a set of artefacts that we had not produced during ontology development (i.e., competency questions, test ontologies, highly-axiomatised versions of the ontology, or translation of the ontology to a logic program with rules) and we think that explicitly preparing any new artefact from the ontology to be used for the evaluation may produce biased results.

In their survey, Brank et al. [BGM05] classify ontology evaluation approaches in comparing to a golden standard, using the ontology in an application, comparing with a source of data, and performing human-based evaluations.

Even if no formal evaluation has been performed in this sense, to a certain extent the ontologies can be considered as validated since they are being successfully used in the prototype developed in the SemSorGrid4Env project[5] and they have been manually inspected by the domain experts and developers that worked in the development of this prototype.

Besides, in our case we did not have a golden standard to base the evaluation on; however, we had the data used in the prototype mentioned above so we analyzed the coverage of the ontologies to these data. All those ontologies that are part of the SemSorGrid4Env infrastructure (SchemaMetadata, ServiceOntology and SsnExtension) were developed with the aim of being generic and, therefore, their coverage is not high. On the other hand, those domain ontologies explicitly developed for the prototype (AdditionalRegions, CoastalDefences and Role) have a high coverage because we prioritized their applicability in the prototype over their reusability and we just modelled all those entities that were expected to be used.

Table 2.3: Ontological metrics for the SemSorGrid4Env ontologies.

<table>
<thead>
<tr>
<th>Ontology</th>
<th>ET(O)</th>
<th>RC(O)</th>
<th>RP(O)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdditionalRegions</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>CoastalDefences</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Role</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SchemaMetadata</td>
<td>1</td>
<td>1.09</td>
<td>1</td>
</tr>
<tr>
<td>ServiceOntology</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SsnExtension</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

3. Evaluation of the Semantic Integrator Service

For the evaluation of the Semantic Integrator service we decided to focus on the query response times, under different load scenarios. Since our ontology-base query translation approach relies on existing query processing engines for streams such as SNEE \cite{GGF11}, we are concerned with the potential overhead that our techniques add to the complete query cycle.

While there have been initiatives proposing benchmarks for streaming data engines (e.g. Linear Road benchmark \cite{ACG04}, or for massive SPARQL queries (e.g. Berlin SPARQL Benchmark \cite{BS09}) over RDF, we have no records of a streaming data benchmark for query translation. However and for the purpose of evaluating our component, we opted for performing a series of tests, measuring mainly the query response times under different load settings. In Section 3.1 we provide details about the experiment settings and then we show the results in Section 3.2 We have also performed a set of stress tests over the Semantic Integrator service, which are detailed in Section 3.3.

3.1 Settings

Most of the real SemSorGrid4Env data sources are low throughput environmental sensor streams, with update rates that range around 5 or 10 minutes. Therefore they are not suitable for performance experimentation and evaluation. Consequently the tests reported in this document have been performed using synthetic data streams that use a tuple generator, configured with the desired settings. These synthetic streams emulate the SemSorGrid4Env sensor names and data types used in the project, but they can be easily tuned and parametrized.

The parameters we used are:

- The query: We have experimented with different SPARQLStream \cite{CCG11, CCG10} queries, each with a different level of complexity. For each case we provide the result of the query translation in the SNEEql language \cite{BGFP08, GGF11}.
- The number of deployed sensors: We used a fixed number of 25 wave sensors, mimicking those we have in the CCO \cite{GGF10} deployment in the south coast of England. We also have 7 sensor streams for the tide sensors.
- The number of concurrent queries launched: In our system queries are typically continuously run, i.e. they get results periodically, so that a client can pull the latest result sets. The number of queries launched can affect the query response time as they are simultaneously fetching data from the streams.
- The stream rates: each sensor produces data tuples at some rate. If the rate is high, e.g. 1 tuple per second, the query response time can be affected, specially because of data translation.
- Query time windows: time windows specify the time range of the stream tuples that will be included in the query.

A sample synthetic stream logical schema is given in Listing 1. Its corresponding physical schema is given in Listing 2. Both SNEE schemas are described in \cite{GGF11}.
The mapping document in the R2RML syntax is given in Annex A. All tests have been performed on an Intel Core i7 1.60 GHz, 6 GB.

3.2 Experiments

In the following subsections we present the experiments and results.

3.2.1 Single vs. Multiple Sensor Queries

In this experiment we considered a simple query that requests the wave height values in one sensor location. The query is specified in SPARQLStream below in Listing 3.

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX cd: <http://www.sensorsgrid4env.eu/ontologies/CoastalDefences.owl#>
PREFIX ssg: <http://sensorsgrid4env.eu/ns#>
SELECT ?wavets ?waveheight
WHERE {?WaveObs a ssn:Observation;
  ssn:observationResultTime ?wavets;
  ssn:observationResult ?waveheight;
  ssn:observedProperty cd:WaveHeight;
  ssn:observedBy ssg:MilfordSensor .}
```

Listing 3: SPARQLStream query requesting the wave heights of the Milford sensor

As it is specified by the last triple pattern, only those measurements observed by the ssg:MilfordSensor will be considered in the query. Following the mapping definition for the Milford sensor (see Listing 20), the query is translated to the SNEE expression in Listing 5.

---

1. [http://www.w3.org/TR/r2rml/]
As we can see the generated query is very simple and only accesses 1 sensor. We plotted the query result times in Figure 3.1 for low update rates, from 0.1 tuples per second to 5 tuples per second. As we can see the service answers quickly until we reach the 1 tuple-per-second rate, at which it grows drastically. In this experiment we have 50 queries running simultaneously, so for high update rates there is a lot of contention, hence the high response time.

![Single Sensor Query - Low Rates](image)

**Figure 3.1:** Response times for a single-sensor query, for different tuple rates

We also analyzed the behavior when we change the SPARQL-Stream query and eliminate the ssg:MilfordSensor restriction. Then we end up with the following SPARQL-Stream query (Listing 6) and its corresponding Sneeql translation (Listing 7).

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX cd: <http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#>
PREFIX ssg: <http://www.semsorgrid4env.eu/ns#>

SELECT ?wavets ?waveheight
```

Listing 4: Mapping a sensor to ObservationValue in R2RML

Listing 5: Translated query in Sneeql

As we can see the behavior when we change the SPARQL-Stream query and eliminate the ssg:MilfordSensor restriction. Then we end up with the following SPARQL-Stream query (Listing 6) and its corresponding Sneeql translation (Listing 7).
WHERE {
?WaveObs a ssn:Observation;
ssn:observationResultTime ?wavets;
ssn:observationResult ssn:waveheight;
ssn:observedProperty cd:WaveHeight;
ssn:observedBy ?sensor .
}

Listing 6: SPARQL Stream query for all wave height sensors

(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_bidefordbay) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_boscombe) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_bracklesham) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_chesil) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_goodwin) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_haylingisland) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_hornsea) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_haylingisland) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_looebay) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_milford) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_minehead) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_pevenseybay) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_perranporth) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_rye) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_sandownbay) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_seaford) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_starbay) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_torbay) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_westbay) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_westonbay) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_weymouth) UNION
(SELECT Hs AS waveheight, DateTime AS wavets FROM envdata_wavenet_poolebay) ;

Listing 7: Translated sneeql query requesting all wave heights in a UNION

In this new scenario the query accesses 26 sensors simultaneously. We executed a continuous query under both settings and measured the response times, with high tuple update rates: up to 1000 tuples per second for each sensor. The results, comparing both the single sensor and 26-sensors are depicted in Figure 3.2.

As the query including the 26 sensors is much more complicated, the query is expected to take more time. However the main explanation for this delay is that it is a union query, hence it includes more data that will pass through the data translation process into ontology instances (SPARQL bound variables to be precise in this case).

### 3.2.2 Concurrent Continuous Queries

In our approach, the user typically issues a query once, and it gets executed continuously, so that results can be fetched upon a request. As the number of simultaneous queries increases, the query response time of each individual query may be affected, specially if the tuple rate is very high. In the following experiment we experimented with high tuple rates (1 to 1000 tuples per second), with 1 to 50 simultaneous queries. The results are depicted in Figure 3.3.

With these very high rates, we can see that the service responds in around 2 seconds until we reach the number of 50 simultaneous queries. at this point the response time increases importantly (for all rates), and beyond that number of queries the service get stuck and out of memory. Notice that rates above 1 tuples per second already show signs of low response times altogether.

### 3.2.3 Window Operations

Queries in SPARQL Stream allow specifying time windows that limit the tuples to be considered in the query, based on temporal constraints specified by the window boundaries. For instance the
Figure 3.2: Response times for a single-sensor query, compared to a 26 sensor union query, for different tuple rates

query in [8] is a slight modification of query in Listing 3 with the addition of a 10 minute time window. The slide parameter indicates how often is the window computed (e.g. every 10 seconds). The query is translated as in Listing 9

Listing 8: SPARQLStream query with a time window

```
(SELECT ?wavets ?waveheight
FROM NAMED STREAM <http://semsorghgrid4env.eu/ns#ccometeo.srdf>
[NOW - 10 MINUTE SLIDE 10 S]
WHERE {
  ?WaveObs a ssn:Observation;
  ssn:observationResultTime ?wavets;
  ssn:observationResult ?waveheight;
  ssn:observedProperty cd:WaveHeight;
  ssn:observedBy ssg:MilfordSensor.
})
```

Listing 9: Translated SNEEq with time window

```
(SELECT Hs AS waveheight, DateTime AS wavets
FROM envdata_milford[FROM NOW - 10 MINUTES TO NOW - 0 MINUTES SLIDE 10 SECONDS] envdata_milford);
```

We executed an experiment with this query, and then changed the window to 1 minute. The results of both cases are displayed in Figure 3.4. As the experiment lasted less than 10 minutes, both queries show very similar characteristics: both have similar data volumes being transformed. Again, we observe that above 1 tuple per second, the query response time increases significantly. However we see that the window size is not determinant for the response time.

However we can also vary the window slide (i.e. how often the window is constructed). We argue that very frequent sliding may increase the response times significantly. The following experiment was conducted to verify this. We executed queries with different slides: 60 seconds, 10 seconds and
10 milliseconds, as in Figure 3.5. As we can see, as the slide decreases, the query response time grows drastically. Even with relatively low rates, with a 10 ms slide the query quickly becomes unresponsive.

### 3.2.4 Join Queries

So far we analyzed simple queries and queries with windows. SPARQLStream queries can be translated into Join queries if the corresponding mapping requires combining two sensor streams. For instance consider the query in Listing 10 that asks for waves heights in Milford higher than tide heights in other areas (no specific sensor for the tides).

```sparql
PREFIX sb: <http://www.w3.org/2009/SSN-XG/Ontologies/SensorBasis.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX ssn: <http://www.sensorsgrid4env.eu/ontologies/CoastalDefences.owl#>
PREFIX ssg: <http://www.sensorsgrid4env.eu/ns#>
SELECT ?wavets ?waveheight ?tideheight
WHERE {
  ?WaveObs a sb:Observation;
  ssn:observationResultTime ?wavets;
  ssn:observedProperty cd:WaveHeight;
  ssn:observedBy ssg:MilfordSensor.

  ?TideObs a sb:Observation;
  ssn:observationResultTime ?tidets;
  ssn:observedProperty cd:TideHeight.

  FILTER (?waveheight >?tideheight)
}
Listing 10: SPARQLStream query requesting wave heights in Milford higher than tide heights
```

The query is translated into the SNEEQL expression in Listing 11. As it can be observed, it is a union of joins, between the Milford stream and all of the tide streams.

```sparql
(SELECT envdata_milford.Hs AS waveheight, envdata_milford.DateTime AS wavets, envdata_deal_tide.Tp AS tideheight
FROM envdata_milford, envdata_deal_tide
WHERE Hs > Tp) UNION
(SELECT envdata_milford.Hs AS waveheight, envdata_milford.DateTime AS wavets, envdata_hernebay_tide.Tp AS tideheight
FROM envdata_milford, envdata_hernebay_tide
WHERE Hs > Tp) UNION
```

Figure 3.3: Response times for different number of queries launched simultaneously, for different tuple rates.
Unfortunately we couldn’t evaluate this query as SNEE supports joins only for windows. We modified the query as in Listing 12 so that it includes a time window. The query is translated into the expression displayed in Listing 13.

```
PREFIX sb: <http://www.w3.org/2009/SSN-XG/Ontologies/SensorBasis.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX cd: <http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#>
PREFIX ssg: <http://semsorgrid4env.eu/ns#>

SELECT ?wavets ?waveheight ?tideheight
FROM NAMED STREAM <http://semsorgrid4env.eu/ns#ccometeo.srdf> [NOW − 300 S SLIDE 1 S]
WHERE {
    ?WaveObs a ssn:Observation;
    ssn:observationResultTime ?wavets;
    ssn:observationResult ?waveheight;
    ssn:observedProperty cd:WaveHeight;
    ssn:observedBy ssg:MilfordSensor.
    ?TideObs a ssn:Observation;
    ssn:observationResultTime ?tidets;
    ssn:observationResult ?tideheight;
    ssn:observedProperty cd:TideHeight.
    FILTER (?waveheight > ?tideheight)
}
```

Listing 12: SPARQL Stream query variation of Listing [10] with windows
Figure 3.5: Response times for time window queries, with different slide parameters

FROM envdata_milford[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS] envdata_milford,
envdata_deal_tide[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS] envdata_deal_tide
WHERE Hs > Tp) UNION
(SELECT envdata_milford.Hs AS waveheight, envdata_milford.DateTime AS wavets, envdata_hernebay_tide.Tp AS tideheight
FROM envdata_milford[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS] envdata_milford,
envdata_hernebay_tide[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS] envdata_hernebay_tide
WHERE Hs > Tp) UNION
(SELECT envdata_milford.Hs AS waveheight, envdata_milford.DateTime AS wavets,
enodata_lymington_tide.Tp AS tideheight
FROM envdata_milford[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS] envdata_milford,
enodata_lymington_tide[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS] envdata_lymington_tide
WHERE Hs > Tp) UNION
(SELECT envdata_milford.Hs AS waveheight, envdata_milford.DateTime AS wavets,
enodata_sandownpier_tide.Tp AS tideheight
FROM envdata_milford[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS] envdata_milford,
enodata_sandownpier_tide[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS] envdata_sandownpier_tide
WHERE Hs > Tp) UNION
(SELECT envdata_milford.Hs AS waveheight, envdata_milford.DateTime AS wavets,
enodata_swanagepier_tide.Tp AS tideheight
FROM envdata_milford[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS] envdata_milford,
enodata_swanagepier_tide[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS] envdata_swanagepier_tide
WHERE Hs > Tp) UNION
(SELECT envdata_milford.Hs AS waveheight, envdata_milford.DateTime AS wavets,
enodata_teignmouthpier_tide.Tp AS tideheight
FROM envdata_milford[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS] envdata_milford,
enodata_teignmouthpier_tide[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS] envdata_teignmouthpier_tide
WHERE Hs > Tp) UNION
(SELECT envdata_milford.Hs AS waveheight, envdata_milford.DateTime AS wavets,
enodata_westbaypier_tide.Tp AS tideheight
FROM envdata_milford[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS] envdata_milford,
enodata_westbaypier_tide[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS] envdata_westbaypier_tide
WHERE Hs > Tp);

Listing 13: Translated query of Listing 12

However we could not evaluate this one either, as SNEE does not allow unions of windows. Therefore,
we further modified the query so that it only takes into account Tides from the Hernebay Sensor
(Listing 14). The translation in SNEEq is in Listing 15.

PREFIX sb: <http://www.w3.org/2009/SSN-XG/Ontologies/SensorBasis.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
PREFIX cd: <http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#>
SemSorGrid4Env

PREFIX ssg: <http://semsorgird4env.eu/ns#>
PREFIX s: <http://semsorgird4env.eu/ns#ccometeo.srdf>  
PREFIX ss: <http://semsorgird4env.eu/ns#>

SELECT ?wavets ?waveheight ?tideheight
FROM NAMED STREAM <http://semsorgird4env.eu/ns#ccometeo.srdf>
WHERE {
  ?WaveObs a ss:Observation;
  ss:observationResultTime ?wavets;
  ss:observedProperty cd: WaveHeight;
  ss:observedBy ssg: MilfordSensor.

  ?TideObs a ss:Observation;
  ss:observationResultTime ?tides;
  ss:observedProperty cd: TideHeight;
  ss:observedBy ssg: HernebaySensor.

  FILTER (?waveheight > ?tideheight)
}

Listing 14: SPARQL Stream query variation of Listing 12 limited to Hernebay Tides

SELECT envdata_milford.Hs AS waveheight, envdata_milford.DateTime AS wavets, envdata_hernebay_tide.Tp AS tideheight
FROM envdata_milford[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS], envdata_hernebay_tide[FROM NOW - 300 SECONDS TO NOW - 0 SECONDS SLIDE 1 SECONDS]
WHERE Hs > Tp;

Listing 15: Translation of Listing 14

The results of our experiments are depicted in Figure 3.6. As we can see the join query is affected drastically by very high rates. Already with 5 tuples per second there is a noticeable increase. Above that the service becomes unresponsive, even if we are only querying only two sensors in the join. This is an expected behavior as the joins are expensive operators, even more in the presence of time windows, and high rates in tuple updates.

Figure 3.6: Response time for a Join query, at different rates

3.2.5 Query Translation

In all the previous experiments we focused on an end to end evaluation of our component. However, as it is explained in [CCG11], the Semantic Integrator relies on underlying streaming data engines, in this particular case snee. Therefore the query response times also depend on the underlying
data processing platform. In this section we try to isolate the measurement of the query translation process, to check to what extent the query and data translation process, from relational streams to ontological elements, hits the service performance. We have simply plotted in Figure 3.7 the query execution time of the $\texttt{sneeql}$ query in Listing 5 and its corresponding $\texttt{SPARQL-Stream}$ query (Listing 3 in Figure 3.8).

![Figure 3.7: Execution times of a $\texttt{sneeql}$ query without query and data translation](image)

As we can see there is a significant increase in the query response time, that follows exactly the same shape as the response time of the $\texttt{sneeql}$ query. In fact most of the overhead lies on the data translation, more than the query translation itself, as every tuple resulting from the $\texttt{snee}$ result set, is transformed to a $\texttt{SPARQL}$ bound variable or to an$\texttt{RDF}$ instance. This iterative process is the main cause for delays, as the query translation only occurs once, when the query is registered. For the rest of the interactions, the data is only pulled and transformed.

After verifying this, we measured how long it takes to perform this Query Translation step, before actually registering and sending the $\texttt{sneeql}$ query for evaluation. We took 5 sample queries, A query over only one sensor (Listing 3), a union query over all wave height sensors (Listing 6), variations of those two queries adding time windows, and a join query (Listing 14). As we can see the query translation is not affected too much by window operators but is sensible to join queries, as these are more complex during the algebra tree construction (See Figure 3.9). In any case the query translation step is an expensive process whose result should be cached in order to minimize its influence over query processing response times.
Figure 3.8: Execution time of the equivalent SPARQLStream query, including query and data translation.

Figure 3.9: Average time elapsed during the query translation phase, for different queries.
3.3 Stress Tests

The empiric evaluation in the previous section focused on the query and data translation processes. In this section we present the results of stress tests performed against the Semantic Integrator service, using the same test platform used in other components of the SemSorGrid4Env architecture.

3.3.1 Test settings

The web services composing the SemSorGrid4Env middleware have been tested using a stress test tool called LoadUI, developed by Eviware. The SemSorGrid4Env middleware has been installed in a virtual machine guest O.S., while the stress test tool runs from the Host O.S. (i.e. the O.S. that runs physically on the machine). By doing so, the stress test tool (which is quite demanding in both CPU usage and RAM memory) won’t affect the status of the web service. At the same time, the network won’t affect either the process of the test, as it is not used. To measure the impact that the stress test has created on the virtual machine, a measurement of the memory status will be performed on both machines, host and guest, before and after the stress tests.

LoadUI uses a set of predefined test cases to send to the server, and loops them over and over to perform the stress tests, so the requests performed are a limited and well-known soap requests. The test cases tend to perform requests on the most common requests that the server will have to handle and the requests that will have a bigger impact in the server in terms of performance, and memory usage.

The stress test itself will be split in three different phases, during phase I, the purpose is to discover the limit of requests per second the server is capable of handling without starting to queue requests. During the phase II, the server will be suffocated with a quantity of requests per second higher than the limit obtained at the phase I, expecting the server to queue requests until some of them are discarded (a fresh install of Apache Tomcat will need 1000 queued requests before starting to discard incoming connections). Phase III will try to discover if the server is able to, after being stressed on phase II, get back to normality, by serving all the queued connections.

We have kept the 3-phase test plan, but instead of using equally all the functionalities of the server, the tests intended to represent how a real Integrator Service client would behave:

- In the first step, we ran on the Integrator Service, the IntegrateAs operation, creating the integrated resource we will be working with during the stress tests.
- After the integrated streaming resource is created, a query is posed to the Integrator Service, therefore generating a new resource which can be pulled using the getStreamItem operation.
- Finally, the stress tests are ran querying to the Integrator Service stream items.

Even though we did not test the whole functionality of the Integrator Service, the functions IntegrateAs and ExecuteQueryFactory have shown quite a low latency which creates too much noise for the stress tests to work properly, and as those functions will be called probably once per client and stream required, the getStreamItem will be called periodically, meaning that the first two functions will be executed not very often. Therefore, we can assume that the Integrator Service will have more chances to be stressed by getStreamItem requests rather than IntegrateAs and ExecuteQueryFactory petitions.

---

2http://www.loadui.org/
3.3.2 Platform

The platform to be tested represents a medium resource server which could be quite affordable to act as a SemSorGrid4Env middleware server. The CPU of the computer used to perform the tests is an Intel Pentium(R) Dual-Core E6300 @2.80GHz, 8Gb of RAM memory (with two Gb dedicated entirely to the Guest OS). The OS used is a 64 bits Ubuntu Natty Narwhal(11.04) on both Host OS and Guest OS, the only difference is that the Host OS is a desktop edition, while the Guest OS is a server edition.

Before starting the tests, the output for the command `free -m` on the Host OS were (Listing 16):

```
Mem: 7875 7595 280 0 272 1604
-/+ buffers/cache: 5718 2157
Swap: 4094 0 4094
```

Listing 16: Output of the free command at the Host

In brief, the host O.S. has plenty of free RAM memory, and swap space, so no swapping is expected while performing the tests. Meanwhile, before starting the tests, the guest O.S. `free -m` results are (Listing 17):

```
Mem: 2008 551 1457 0 8 136
-/+ buffers/cache: 406 1601
Swap: 1019 53 966
```

Listing 17: Output of the free command at the Guest

3.3.3 Test Results

Finally, after the stress tests have been executed, the output for the `free -m` command at the host OS has been (Listing 18):

```
Mem: 7875 7738 137 0 29 608
-/+ buffers/cache: 7100 774
Swap: 4094 0 4094
```

Listing 18: Output of the free command at the Host after the tests

And for the Guest OS(Listing 19):

```
Mem: 2008 761 1246 0 131 123
-/+ buffers/cache: 505 1502
Swap: 1019 7 1012
```

Listing 19: Output of the free command at the Guest after the tests

**Tests summary:** We present the test general statistics in Table 3.1 and the request statistics in Table 3.2.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Seconds</th>
<th>Queued</th>
<th>Running</th>
<th>Completed</th>
<th>Time</th>
<th>Requests</th>
<th>Failed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phase I</td>
<td>261</td>
<td>0</td>
<td>100</td>
<td>403</td>
<td>02:30:34 PM</td>
<td>503</td>
<td>0</td>
</tr>
<tr>
<td>Phase II</td>
<td>269</td>
<td>1000</td>
<td>100</td>
<td>583</td>
<td>02:35:03 PM</td>
<td>583</td>
<td>35</td>
</tr>
<tr>
<td>Phase III</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>1100</td>
<td>02:43:23 PM</td>
<td>1000</td>
<td>73</td>
</tr>
</tbody>
</table>

Table 3.1: Stress test general statistics.

We can summarize these results as follows:
Table 3.2: Stress test request statistics.

- Phase I: During phase I, the Integrator Service has been able to handle 1.54 requests per second for 261 seconds without queuing any request, and completing 403 requests successfully.

- Phase II: The number of requests per second has been increased to 6.05 per second, and after 269 seconds, the Integrator Service has finally crashed, queuing 1000 requests, discarding 12 and obtaining a Time-out error on 35 requests. Even though, the number of requests completed per second has increased from 1.54 to 2.16.

- Phase III: Finally, after stopping the request generator, the Integrator Service has slowly recovered from its saturation, and after 500 seconds, it has been able to serve all the pending requests, except for 73 more requests which were canceled due to a timeout error. During this phase, the number of completed requests per second has, again, increased, even though not as significantly as before, to 2.2 requests per second.
4. Conclusions

Regarding the evaluation of the SemSorGrid4Env ontologies, the evaluation results unveiled some problems (or potential ones) in the ontologies that were fixed.

It is also important to remark the important role of evaluation in the ontology development process, since sometimes problems in the ontologies may not come from the same process but from other factors, such as changes in the reused ontologies as happened in our case.

However, ontology evaluation is not a straightforward task. Even if different ontology evaluation methods are proposed in the literature, applying them requires manual intervention minimally supported by software tools. Therefore, we remark the need for automation of ontology evaluation activities by means of software ontology evaluation frameworks that facilitate evaluation and enable advanced evaluation scenarios (e.g., regression evaluations).

About the evaluation of the Semantic Integrator, we have provided empirical results that show that our component is able to respond to semantic queries over sensor data in a timely fashion, in settings even more demanding than those of the real deployments in the SemSorGrid4Env project.

However, we have observed that our solution response times under very high rates tend to render the system unresponsive. Although in environmental sensor deployments these rates are not realistic, we point out this as an engineering problem that could be addressed in the future for adaptation to other kind of use-cases.

In addition, we are aware that our approach is dependent on a stream processing engine, in this case sNee. While it has served very well for the purposes of the SemSorGrid4Env project, for other scenarios other underlying platforms could be used instead, and throw entirely different results in terms of query response times, with minimal changes on the data and query translation sides. We hope to explore this point in the future as well.
A. Annex: Mappings

```rml
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ssg: <http://semsorgrid4env.eu/ns#> .
@prefix ssn: <http://purl.oclc.org/NET/ssnx/ssn#> .
@prefix cd: <http://www.semsorgrid4env.eu/ontologies/CoastalDefences.owl#> .
@prefix dul: <http://www.loa-cnr.it/ontologies/DUL.owl#> .
@prefix time: <http://www.w3.org/2006/time#> .
@prefix regions: <http://www.semsorgrid4env.eu/ontologies/AdditionalRegions.owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix morph: <http://es.upm.fi.dia.oeg/morph#> .
@prefix : <http://es.upm.fi.dia.oeg/R2RMapping#> .

:observedByPred rr:predicate ssn:observedBy.

>:windSpeedObsResult rr:predicateMap [ rr:predicate ssn:observationResult ];
  rr:objectMap [ rr:column "WindSpeed" ].
>:abpWindSpeedObsResult rr:predicateMap [ rr:predicate ssn:observationResult ];
  rr:objectMap [ rr:column "wind_speed" ].
>:waveObsResult rr:predicateMap [ rr:predicate ssn:observationResult ];
  rr:objectMap [ rr:column "Hs" ].
>:tideObsResult rr:predicateMap [ rr:predicate ssn:observationResult ];
  rr:objectMap [ rr:column "Tp" ].
>:heaveObsResult rr:predicateMap [ rr:predicate ssn:observationResult ];
  rr:objectMap [ rr:column "VerticalHeave" ].
>:speedObsResult rr:predicateMap [ rr:predicate ssn:observationResult ];
  rr:objectMap [ rr:column "speed" ].
>:obsTime rr:predicateMap [ rr:predicate ssn:observationResultTime ];
  rr:objectMap [ rr:column "DateTime" ].
>:obsTimestamp rr:predicateMap [ rr:predicate ssn:observationResultTime ];
  rr:objectMap [ rr:column "timestamp" ].
>:fOILsea rr:predicateMap [ rr:predicate ssn:featureOfInterest ];
  rr:objectMap [ rr:object ssg:Sea ].
>:fOILwind rr:predicateMap [ rr:predicate ssn:featureOfInterest ];
  rr:objectMap [ rr:object ssg:Wind ].
>:foIShip rr:refPredicateMap [ rr:predicate ssn:featureOfInterest ];
  rr:refObjectMap [ rr:parentTriplesMap :sotonShipTMap; rr:joinCondition "" ].
>:waveObsProp rr:predicateMap [ rr:predicate ssn:observedProperty ];
  rr:objectMap [ rr:object cd:WaveHeight ].
>:tideObsProp rr:predicateMap [ rr:predicate ssn:observedProperty ];
  rr:objectMap [ rr:object cd:TideHeight ].
>:heaveObsProp rr:predicateMap [ rr:predicate ssn:observedProperty ];
  rr:objectMap [ rr:object cd:VerticalHeave ].
>:windSpeedObsProp rr:predicateMap [ rr:predicate ssn:observedProperty ];
  rr:objectMap [ rr:object cd:WindSpeed ].
>:speedObsProp rr:predicateMap [ rr:predicate ssn:observedProperty ];
  rr:objectMap [ rr:object cd:Speed ].
>:waveLat rr:predicateMap [ rr:predicate ssn:hasLatitude ];
  rr:objectMap [ rr:column "Lat" ].
>:waveLon rr:predicateMap [ rr:predicate ssn:hasLongitude ];
  rr:objectMap [ rr:column "Lon" ].

:subjectWave rr:template "http://semsorgrid4env.eu/ns#Observation/WaveHeight/CCO/{timestamp}";
  rr:template "http://semsorgrid4env.eu/ns#Observation/WaveHeight/WaveNet/{timestamp}";

:subjectTide rr:template "http://semsorgrid4env.eu/ns#Observation/TideHeight/CCO/{timestamp}";

:subjectHeave rr:template "http://semsorgrid4env.eu/ns#Observation/VerticalHeave/EMI/{timestamp}";

:subjectWind rr:template "http://semsorgrid4env.eu/ns#Observation/WindSpeed/CCO/{timestamp}";

:subjectAbpWind rr:template "http://semsorgrid4env.eu/ns#Observation/WindSpeed/ABP/{timestamp}";

```

D4.4 Evaluation of the Ontology-based Data Integration Service and the Ontologies 23
:looeBaySensor rr:objectMap [rr:object ssg:LooeBaySensor]; rr:predicateMap :observedByPred.
:emuWestBaySensor rr:objectMap [rr:object :EmuWestBaySensor]; rr:predicateMap :observedByPred.

:poolebayWaveObservation a rr:TriplesMap;
rr:subjectMap :subjectWaveNet;
rr:logicalTable [rr:tableName "envdata_wavenet_poolebay"]; rr:predicateObjectMap :waveObsResult,:obsTime, :foISea, :waveObsProp,:waveLat,:waveLon,
:poolebaySensor;

:MilfordWaveObservation a rr:TriplesMap;
rr:subjectMap :subjectWave; rr:logicalTable [rr:tableName "envdata_milford"]; rr:predicateObjectMap :waveObsResult,:obsTime, :foISea, :waveObsProp,:waveLat,:waveLon,
:milfordSensor;

:perranporthWaveObservation a rr:TriplesMap;
rr:subjectMap :subjectWave; rr:logicalTable [rr:tableName "envdata_perranporth"]; rr:predicateObjectMap :waveObsResult,:obsTime, :foISea, :waveObsProp,:waveLat,:waveLon,
perranporthSensor;

:pevenseybayWaveObservation a rr:TriplesMap;
rr:subjectMap :subjectWave; rr:logicalTable [rr:tableName "envdata_pevenseybay"]; rr:predicateObjectMap :waveObsResult,:obsTime, :foISea, :waveObsProp,:waveLat,:waveLon,
:pevenseybaySensor;

:goodwinWaveObservation a rr:TriplesMap;
rr:subjectMap :subjectWave; rr:logicalTable [rr:tableName "envdata_goodwin"]; rr:predicateObjectMap :waveObsResult,:obsTime, :foISea, :waveObsProp,:waveLat,:waveLon,
goodwinSensor;

:torbayWaveObservation a rr:TriplesMap;
rr:subjectMap :subjectWave; rr:logicalTable [rr:tableName "envdata_torbay"]; rr:predicateObjectMap :waveObsResult,:obsTime, :foISea, :waveObsProp,:waveLat,:waveLon,
torbaySensor;

:rustingtonWaveObservation a rr:TriplesMap;
rr:subjectMap :subjectWave; rr:logicalTable [rr:tableName "envdata_rustington"]; rr:predicateObjectMap :waveObsResult,:obsTime, :foISea, :waveObsProp,:waveLat,:waveLon,
rustingtonSensor;

:bidefordbayWaveObservation a rr:TriplesMap;
rr:subjectMap :subjectWave; rr:logicalTable [rr:tableName "envdata_bidefordbay"]; rr:predicateObjectMap :waveObsResult,:obsTime, :foISea, :waveObsProp,:waveLat,:waveLon,
bidefordbaySensor;

:folkestoneWaveObservation a rr:TriplesMap;
rr:subjectMap :subjectWave; rr:logicalTable [rr:tableName "envdata_folkestone"]; rr:predicateObjectMap :waveObsResult,:obsTime, :foISea, :waveObsProp,:waveLat,:waveLon,
folkestoneSensor;

:boscombeWaveObservation a rr:TriplesMap;
rr:subjectMap :subjectWave; rr:logicalTable [rr:tableName "envdata_boscombe"]; rr:predicateObjectMap :waveObsResult,:obsTime, :foISea, :waveObsProp,:waveLat,:waveLon,
boscombeSensor;

:penzanceWaveObservation a rr:TriplesMap;
rr:subjectMap :subjectWave; rr:logicalTable [rr:tableName "envdata_penzance"]; rr:predicateObjectMap :waveObsResult,:obsTime, :foISea, :waveObsProp,:waveLat,:waveLon,
penzanceSensor;

D4.4 Evaluation of the Ontology-based Data Integration Service and the Ontologies 25
SemSorGrid4Env

D4.4 Evaluation of the Ontology-based Data Integration Service and the Ontologies

26
:teignmouthpierTideObservation a rr:TriplesMap;
  rr:subjectMap :subjectTide; rr:logicalTable [rr:tableName "envdata_teignmouthpier_tide"];
.
:swanagepierTideObservation a rr:TriplesMap;
  rr:subjectMap :subjectTide; rr:logicalTable [rr:tableName "envdata_swancepier_tide"];
.
:sandownpierTideObservation a rr:TriplesMap;
  rr:subjectMap :subjectTide; rr:logicalTable [rr:tableName "envdata_sandownpier_tide"];
.
:westbaypierTideObservation a rr:TriplesMap;
  rr:subjectMap :subjectTide; rr:logicalTable [rr:tableName "envdata_westbaypier_tide"];
.
:dealWindSpeedObservation a rr:TriplesMap;
  rr:subjectMap :subjectWindSpeed; rr:logicalTable [rr:tableName "envdata_deal_met"];
.
:hernebayWindSpeedObservation a rr:TriplesMap;
  rr:subjectMap :subjectWindSpeed; rr:logicalTable [rr:tableName "envdata_hernebay_met"];
.
:looebayWindSpeedObservation a rr:TriplesMap;
  rr:subjectMap :subjectWindSpeed; rr:logicalTable [rr:tableName "envdata_looebay_met"];
.
:worthingWindSpeedObservation a rr:TriplesMap;
  rr:subjectMap :subjectWindSpeed; rr:logicalTable [rr:tableName "envdata_worthing_met"];
.
:arunplatformWindSpeedObservation a rr:TriplesMap;
  rr:subjectMap :subjectWindSpeed; rr:logicalTable [rr:tableName "envdata_arunplatform_met"];
.
:swancepierWindSpeedObservation a rr:TriplesMap;
  rr:subjectMap :subjectWindSpeed; rr:logicalTable [rr:tableName "envdata_swancepier_met"];
.
:sandownpierWindSpeedObservation a rr:TriplesMap;
  rr:subjectMap :subjectWindSpeed; rr:logicalTable [rr:tableName "envdata_sandownpier_met"];
.
:weymouthWindSpeedObservation a rr:TriplesMap;
  rr:subjectMap :subjectWindSpeed; rr:logicalTable [rr:tableName "envdata_weymouth_met"];
.
:westbaypierWindSpeedObservation a rr:TriplesMap;
  rr:subjectMap :subjectWindSpeed; rr:logicalTable [rr:tableName "envdata_westbaypier_met"];
.
:teignmouthpierWindSpeedObservation a rr:TriplesMap;
  rr:subjectMap :subjectWindSpeed; rr:logicalTable [rr:tableName "envdata_teignmouthpier_met"];
.
:folkestoneWindSpeedObservation a rr:TriplesMap;
  rr:subjectMap :subjectWindSpeed; rr:logicalTable [rr:tableName "envdata_folkestone_met"];
.
:lymingtonWindSpeedObservation a rr:TriplesMap;
  rr:subjectMap :subjectWindSpeed; rr:logicalTable [rr:tableName "envdata_lymington_met"];
.
:bidefordBayHeaveObservation a rr:TriplesMap;
D4.4 Evaluation of the Ontology-based Data Integration Service and the Ontologies
Listing 20: Mapping a sensor to ObservationValue in R2RML
Bibliography


