Integrating Distributed Configurations with RDFS and SPARQL

Gottfried Schenner¹ and Stefan Bischof¹ and Axel Polleres² and Simon Steyskal¹,²

Abstract. Large interconnected technical systems (e.g. railway networks, power grid, computer networks) are typically configured with the help of multiple configurators, which store their configurations in separate databases based on heterogeneous domain models (ontologies). In practice users often want to ask queries over several distributed configurations. In order to reason over these distributed configurations in a uniform manner a mechanism for ontology alignment and data integration is required. In this paper we describe our experience with using standard Semantic Web technologies (RDFS and SPARQL) for data integration and reasoning.

1 INTRODUCTION

Product configuration [9] is the task of assembling a system from predefined components satisfying the customer requirements. Large technical systems are typically configured with the help of multiple configuration tools. These configurators are often specific to a technology or vendor and therefore use heterogeneous domain models (ontologies).

For large interconnected systems (e.g. railway networks, power grid) the configuration of the overall system may be stored across separate databases, each database containing only the information for a sub-system.

The domain models and databases of these configurators are a valuable source of information about the deployed system. But there must be a way to access the information in an uniform and integrated manner in order to exploit this.

Figure 1 shows a typical scenario from the railway domain. The individual stations of a network are built by different vendors (A, B, C). Vendors A and B use proprietary configurators (A, B) and store the configurations of these stations in separate projects. Vendor C does not use a configurator, therefore there is no (digital) data available to integrate.

In the railway scenario the railway company owning the railway network wants to obtain information about the whole network in a vendor-independent way. To achieve this, some form of ontology and data integration is necessary. We can identify three steps: (i) create a vendor-independent ontology, (ii) map or align the vendor-specific ontologies or schemas to the vendor-independent ontology, and (iii) provide the vendor-specific data in terms of the vendor-independent ontology.

This paper investigates, how to use standard Semantic Web technologies (RDFS, SPARQL and OWL) for data integration. Our approach uses SPARQL CONSTRUCT queries to generate a linked system view of the distributed configurations as depicted in Figure 2.

This system view can then (i) be queried in a uniform manner, (ii) be checked for constraint violations taking all relevant configurations into account and (iii) be used for reasoning and general consistency checks (cf. Figure 3).

The remainder of this paper is structured as follows: Chapter 2 discusses the preliminaries of this paper, especially the used Semantic Web technologies. Chapter 3 introduces the working example of this paper. Chapter 4 shows how to derive an integrated view of the system from the individual configurator specific databases, in Chapter 5 we...
Figure 3: Using a linked system view for querying and reasoning over distributed configurations.

The proposed approach builds heavily on Semantic Web standards and technologies. Instance data is represented as RDF triples, domain models are mapped to domain dependent ontologies/vocabularies and queries are formulated in SPARQL.

2.1 Data representation with RDF

The Resource Description Framework (RDF) [15] is a framework for describing and representing information about resources and is both human-readable and machine-processable. These abilities offer the possibility to easily exchange information in a lightweight manner among different applications.

In RDF every resource is identified by its URI and represented as subject - predicate - object triples, where subjects and predicates are URIs and objects can either be literals (strings, integers, . . . ) or URIs as shown in Figure 4. Additionally, subjects or objects can be defined as blank nodes, these blank nodes do not have a corresponding URI and are mainly used to describe special types of resources without explicitly naming them. For example the concept mother could be represented as a female person having at least one child.

Figure 4: A simple RDF triple.

2.2 Querying with SPARQL

SPARQL Protocol And RDF Query Language (SPARQL) [14] is the standard query language for RDF, which has become a W3C Recommendation in version 1.1 in 2013. Its syntax is highly influenced by the previous introduced RDF serialization format Turtle [1] and SQL [4] a query language for relational data.

Besides basic query operations such as union of queries, filtering, sorting and ordering of results as well as optional query parts, version 1.1 extended SPARQL’s portfolio by aggregate functions (SUM, AVG, MIN, MAX, COUNT, . . . ), the possibility to use subqueries, perform update actions via SPARQL Update and several other heavily requested missing features [23].

Furthermore, it is possible to create entirely new RDF graphs based on the variable bindings constituted in graph patterns which are matched against one or more input graphs, using SPARQL CONSTRUCT queries. Using such CONSTRUCT queries offers the possibility to easily define transformations between two or more RDF graphs/ontologies, which serves as a basic building block for the present paper.

2.3 Semantic heterogeneity

In order to be able to integrate two or more ontologies into one integrated knowledge base, it is mandatory to define correspondences between the elements of those ontologies to reduce semantic heterogeneity among the integrated ontologies [8].

The problem of semantic heterogeneity can be caused by several facts, e.g. that different ontologies model the same domain in different levels of precision or use different terms for the same concepts [26] (e.g. a concept Computer is equivalent to another concept Device). Such “simple” differences can be detected by most of the current state-of-the-art ontology matching systems like YAM++ [21] or LogMap [18]. However more complex heterogeneities (e.g. a concept Subnet is equivalent to the union of the concepts Computer and Switch; or a property hasPort, which links a Computer to its Port, is equivalent to an attribute ownsPort, which contains the respective port as string representation) are not only more difficult to detect but also not supported by the majority of ontology matching tools [13, 27], although a few approaches to tackle those problems exist [5, 6, 26]. A slightly different approach was followed by [24], where the authors propose a framework which defines executable semantic mappings between ontologies based on SWRL [16] rules and string similarity.

Nevertheless, based on the absence of ontology matching tools which are capable of detecting such complex correspondences, we assume the presence of already known correspondences between entities of the ontologies for our integration scenario.

3 WORKING EXAMPLE

As working example [3] a fictitious computer network is used and represented as UML class diagrams. Figure 5 shows the customer view (system view) of the network.

The following additional constraints hold for the system view:

- In the computer network every computer has a unique address
- A computer can be part of 1-2 subnets
- A computer is part of exactly one project

3 All listings within this paper are serialized in Turtle syntax.
4 The example ontologies and queries are available upon request from the first author.
A project is some arbitrary subdivision of the whole network (e.g. building)
A subnet can be part of multiple projects

In the example there are 2 vendors (A and B), each providing their own configurator. A project can be configured either with configurator A or configurator B. In both cases there is one configurator database for every project. None of the domain models contains the concept of a subnet as found in the system view.

Figure 6a shows the domain model of configurator A. In the domain model of configurator A computers are called devices. Internal devices are the devices configured in the current project. External devices are devices of other projects that are directly connected to a internal device. These are needed to configure the network cards of the internal device.

Figure 7a shows the domain model of configurator B. Vendor B realizes the computer network with switches. Computers can have 1 or 2 ports, which must be connected to a port of an switch. The attribute external is set to ‘true’ for elements that are external to the current project.

3.1 Converting object-oriented models to ontologies

Although using Description Logics for configuration has a long history [10, 20, 28] in our experience large scale industrial configurators mostly use some form of UML-like object-oriented formalisms. For this paper we use the approach for converting object-oriented data models and their instance data into RDF/OWL shown in Table 1. Because of the clear correspondence between UML class diagrams and OWL ontologies we depict ontologies also as UML class diagrams.

Listing 1: Ontology A with instance data

```xml
@object model
ontoA:Device rdf:type owl:Class .
ontoA:InternalDevice rdf:type owl:Class ;
rdfs:subClassOf ontoA:Device .
ontoA:Device_address rdf:type
```

3.2 Unique Name Assumption and Closed World Assumption

When converting the instance data of a configurator to RDF an identifier (URI) for every object must be generated. Most product configurators impose the Unique Name Assumption, i.e. objects with different object-ID refer to different objects of the domain. In the example above we therefore know that ontoA:a1 and ontoA:aR refer to different Devices.

RDF/OWL does not impose the Unique Name Assumption. This is a desirable feature when reasoning about linked data. If one wants to integrate instance data from different sources using heterogeneous ontologies, these ontologies will often refer to the same entity under different URIs. The same can happen, when we integrate multiple interconnected configurations into one configuration.

Figures 6b and 7b show the configurations of two projects (A and B). Although every computer/device is only represented once in each configuration, some computers/device are known in both projects.
4 DATA INTEGRATION WITH SPARQL

We followed an approach proposed in [7] which motivates the use of SPARQL CONSTRUCT queries to perform data integration (i.e. based on known correspondences between ontologies, we are able to translate their instance data to be conform with the structure of the integrated ontology).

4.1 Creation of the system view

As a first step in our data integration approach a system view of the configurator specific instance data is created. This system view reflects the view of the owner of the configured system and is completely self contained i.e. does not contain any URIs of the domain specific ontologies. To derive the system view from the proprietary configurator data we use SPARQL CONSTRUCT queries. Figure 6b shows a configuration of configurator A, Figure 7b shows a configuration of configurator B. The projects of the two configurations are connected via the subnet containing A1(C1), B1(C4) and B2(C5).

4.1.1 Creating instances

To map an instance of the source ontology to a new instance of the target ontology we can either generate a new URI in the namespace of the target ontology or use blank nodes.

The following example (cf. Listing 2) creates a computer in the system ontology for every device of the source ontology A by creating a new unique URI using a unique identifier of the target object (in this case the attribute address).

```sparql
PREFIX : <http://example.com/]
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX x: <http://example.com/example1#>
PREFIX y: <http://example.com/example2#>

CONSTRUCT { GRAPH ?overlay { ?device :identifier ?identifier ; :address ?address . } }
WHERE { GRAPH ?source { ?device :identifier ?identifier . } }

```
executing the CONSTRUCT queries does not matter.

**Listing 2: Instance creation with new URI**

```
CONSTRUCT {
  ?computer ontoSys:Computer_address ?address.
}
WHERE {
  ?device ontoA:Device_address ?address.
  BIND(URI(CONCAT(URISYS, STR(?address))) AS ?computer)
}
```

If in contrast blank nodes are used, every CONSTRUCT query generates a new blank node for a source object. Therefore we use blank nodes only, when it is not possible or inconvenient to create a unique URI for an instance. In our example, since there is no identifier for projects in the source ontology, new projects can be created with the CONSTRUCT query shown in Listing 3.

**Listing 3: Instance creation with blank node**

```
CONSTRUCT {
  _:p rdf:type ontoSys:Project .
  _:p ontoSys:origin ?project .
}
WHERE {
  ?project rdf:type ontoA:Project .
}
```

By using the special object-property ontoSys:origin, we can keep track what led to the construction of the blank node. This information can then be reused in subsequent CONSTRUCT queries.

4.1.2 Complex mapping

Sometimes it is more convenient to use multiple URIs for the same instance, especially if there is no explicit representation of the concept of the object in the source ontology. These multiple URIs will then be related using owl:sameAs.

In our example the concept of a subnet is not directly represented in ontology A. To create the subnets for instance data of ontology A a more complex query is necessary as depicted in Listing 4.

Listing 4: Creating subnets from instance data

```
# C = abbreviation for URI of computer
# SIRI = abbreviation for URI of subnets
CONSTRUCT {
  ?sub1 rdf:type ontoSys:Subnet .
  ?sub2 rdf:type ontoSys:Subnet .
  ?sub1 owl:sameAs ?sub2 .
}
WHERE {
  { ?d1 ontoA:Device_slot1Connected ?d2 .
    ?d1 ontoA:Device_address ?a1 .
    BIND(CONCAT(STR(?a1),"_1") AS ?sid1) .
    UNION {
      BIND(CONCAT(STR(?a2),"_2") AS ?sid1) .
      ?d1 ontoA:Device_address ?a1 .
      BIND(CONCAT(STR(?a2),"_1") AS ?sid2) .
      UNION {
        BIND(CONCAT(STR(?a1),"_2") AS ?sid2) .
        BIND(URI(CONCAT(C,STR(?sid1))) AS ?c1)
        BIND(URI(CONCAT(SIRI,STR(?sid1))) AS ?c2)
        BIND(URI(CONCAT(SIRI,STR(?sid2))) AS ?sub1)
        BIND(URI(CONCAT(C,STR(?sid2))) AS ?sub2)
      }
    }
  }
}
```

5 USING THE INTEGRATED MODEL

The data of the different systems is available and expressed in terms of a common ontology. We can now access the data in a uniform manner and perform different kinds of operations. This section presents two classes of use cases, namely posing queries over the whole system and checking constraints concerning several systems.
5.1 Queries

After the data-integration the former heterogeneous data can now be queried in a uniform manner using only concepts of the system ontology.

Listing 5: Example Quering the system model

```sparql
PREFIX ?p : owl:sameas
SELECT ?p ?address
WHERE {
  ?c ontoSys:computer_address ?address .
}
```

5.2 Checking constraints

If one wants to query information specific to a domain ontology, this data is still accessible via the ontoSys:origin link. One use case for using the ontoSys:origin property, is to detect inconsistencies in the source data. For example if a subnet is part of two projects, for every computer in that subnet, there must be two representations in the source ontologies (in one of these projects the computer is external). The following query checks this property.

Listing 6: Checking constraints

```sparql
PREFIX ?c : ontoSys:subnet
PREFIX ?project : ontoSys:Project_computers
PREFIX ?sub : ontoSys:Subnet_computers
PREFIX ?o : ontoSys:origin
PREFIX ?project2 : ontoSys:Project_subnets
SELECT ?c ?o
WHERE {
  ?project ontoSys:Project_computers ?c .
  ?project2 ontoSys:Project_subnets ?o .
  FILTER(?project != ?project2)
  { ?c ontoSys:origin ?o .
    MINUS {
      FILTER(?o1 != ?o2)
    }
  }
}
```

So far we checked the integrity of the instance data by writing special SPARQL queries. Whenever these queries are not empty a constraint violation is detected. Alternatively SPARQL CONSTRUCT queries can be used to derive a special property ontoSys:constraintviolation and record the reason for the inconsistencies.

Listing 7: Constraint violations

```sparql
PREFIX _c : ontoSys:constraintviolation
PREFIX _o : ontoSys:description
PREFIX inconsistent_data : "inconsistent_data"
CONSTRUCT {
  _c:constraintviolation ?c .
  _o:description "inconsistent_data" .
} .
```

5.3 Special treatment of owl:sameAs

As discussed in Chapter 3.2, OWL does not impose the unique name assumption (UNA). Therefore it is common to have different names (URIs) refer to the same real-world object. In that case they can be linked via owl:sameAs. SPARQL is unaware of the special semantics of owl:sameAs. This can be a problem, especially when using counting aggregates, since one usually wants to count the number of real-objects and not the number of URIs referring to it. Take for example a query counting the number of subnets. Our construction of subnet-URIs creates a URI for every connected port of a computer (Figure 8). A naive SPARQL query would count all distinct URIs that refer to the same subnet (ontoSys:S1, ontoSys:S2, ontoSys:S3) i.e. resulting in 3 instead of the expected answer 1. To fix this, one has to choose one representative for every element equivalence class induced by owl:sameAs and count the number of representatives. In our approach this is done by choosing the lexicographically smallest element.

Listing 8: Example counting predicates

```sparql
PREFIX ?s : ontoSys:subnet
PREFIX ?p : ontoSys:project
PREFIX ?f : ontoSys:subnet
SELECT (COUNT(DISTINCT ?s) AS ?numberofsubnets)
WHERE {
  ?s a ontoSys:Subnet .
  # first subquery
  { SELECT ?s where {
    ?sub ontoSys:subnet ?s .
    OPTIONAL {
      FILTER (STR(?notfirst) < STR(?s))
      FILTER (!BOUND(?notfirst))
    } } # result: numberofsubnets = 6
  # query with special sameas treatment
  # chooses the lexicographic first element
  # as representation of the equivalence class
  SELECT (COUNT(DISTINCT ?f) AS ?numberofsubnets)
  WHERE {
    ?s ontoSys:Subnet .
    # first stubery
    { SELECT ?s
      WHERE {
      OPTIONAL {
        FILTER (STR(?notfirst) < STR(?s))
        FILTER (!BOUND(?notfirst)))
      } } # result: numberofsubnet = 2
    }
```

An alternative approach would be to replace all cliques of the RDF-graph linked by owl:sameAs with a new unique URI. We did not consider that because it requires a proprietary implementation and by replacing URIs, one loses information about the source of information. For instance, if the instance data of two different configurators refer to the same real-world object but have conflicting data-values for that object, both values and their sources must be communicated to the end-user.

6 RELATED WORK

In order to successfully perform data or information integration using Semantic Web technologies two main issues have to be addressed, namely:

Ontology Mapping Tackling the difficulties of Ontology Mapping (i.e. defining alignments between ontologies) extensive studies have been taken out over the last couple of years [11, 12, 19], mainly focusing on resolving heterogeneity among different ontologies or data sources by detecting similarities amongst them.

Ontology Integration Two main approaches can be identified for integrating different ontologies [22], (i) define an upper ontology which contains general concepts and properties for those in the underlying more specific ones and define mappings between them and (ii) define alignments directly between underlying ontologies and use query rewriting for query support [10, 33].

With its W3C Recommendation for version 1.1 in 2013 [14], introducing e.g. UPDATE queries and a revised entailment regime, SPARQL has become more feasible to be used within information integration scenarios and not only as query language for RDF data.
7 CONCLUSIONS

When we started writing this paper, we were looking for a lightweight approach for data integration for distribute configurations using standard Semantic Web technologies.

In the present paper we show that using solely SPARQL and RDFS is sufficient for an approach that relies only on standards and makes it easy to introduce new concepts and individuals on the fly using SPARQL queries. This is especially important for practical use cases, where it is unpredictable which information a customer will want about the configured system.

We tested our approach with real-world data. On a standard Windows-7 laptop with 8 GB using the SPARQL-API of JENA 2.11.1 a large database (>500000 instances) can be integrated in less than 5 minutes resulting in a RDF-graph with more than 500000 triples.

We also considered using OWL reasoners but could not find a solver-independent way of creating new individuals. Nevertheless, for future work we plan to look into using richer OWL ontologies, which as part-subpart, resource, (hardware-)component etc.

As can be seen in the example SPARQL queries in this paper, some queries (especially the ones that take into account owl:sameAs properties), are only understandable for a SPARQL expert. One approach for making queries more accessible for a SPARQL beginner would be to hide the special treatment of owl:sameAs from the inexperienced user by using query rewriting.

ACKNOWLEDGEMENTS

Stefan Bischof and Simon Stetsyk have been partially funded by the Vienna Science and Technology Fund (WWTF) through project ICT12-015.

Simon Stetsyk has been partially funded by ZIT, the Technology Agency of the City of Vienna (Austria), in the programme ZIT13 plus, within the project COSIMO (Collaborative Configuration Systems Integration and Modeling) under grant number 967327.

REFERENCES