OGC Testbed-14

Characterization of RDF Application Profiles for Simple Linked Data Application and Complex Analytic Applications Engineering Report

Table of Contents

Publication Date: 2019-02-04

Approval Date: 2018-12-13

Submission Date: 2018-12-05

Reference number of this document: OGC 18-094r1

Reference URL for this document: http://www.opengis.net/doc/PER/t14-D022-1

Category: OGC Public Engineering Report

Editor: Stephane Fellah

Title: OGC Testbed-14: Characterization of RDF Application Profiles for Simple Linked Data Application and Complex Analytic Applications Engineering Report

indicate compliance with any LICENSOR standards or specifications.

This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

None of the Intellectual Property or underlying information or technology may be downloaded or otherwise exported or reexported in violation of U.S. export laws and regulations. In addition, you are responsible for complying with any local laws in your jurisdiction which may impact your right to import, export or use the Intellectual Property, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

Chapter 1. Summary

This Engineering Report (ER) enhances the understanding of the concept of **application profiles** (AP) for ontologies based on the Web Ontology Language (OWL) and used by Linked Data (LD) applications. The concept of an **Application Profile** for Unified Modeling Language (UML) and Extensible Markup Language (XML) schemas, in particular Geographic Markup Language (GML) application profiles, is pretty well-defined and understood within the communities of Open Geospatial Consortium (OGC) and International Organization for Standardization (ISO). Moreover, in the context of Linked Data and ontologies, the term is still ill-defined, as ontologies are defined using an Open World Assumption (OWA), as well as classes and properties are first-class modeling objects in ontology modeling. The work documented in this report includes:

- Definition and characterization of Resource Description Framework (RDF) application profiles for simple linked data applications and complex analytic linked data applications.
- Determination of preliminary techniques for the development of subsets of ontologies to support different types of applications (simple linked data and complex analytic)
- An initial model for defining metadata about application profiles, so they can be searched and discovered by agents.

1.1. Requirements & Research Motivation

This ER relates to ontologies developed for the National System for Geospatial Intelligence (NSG). The ER describes the work to define NSG Enterprise Ontology (NEO) Application Profiles and sample data for a Simple Linked Data application and a Complex Analytic Application. The goal of this task is to design NEO application profiles that exercise different aspects of ontology for simple linked data application and for complex analytic application.

A ***simple linked data application*** is defined as an application in which no or limited ontology reasoning would be employed. They are typically used to enable display or navigation of information. Application profiles supporting simple linked data applications are profiles-of-ontologies that are still ontologies but whose utility is restricted to some particular domain-of-discourse (e.g., maritime safety-of-navigation, rather than all-of-geography) and can be consumed by applications having limited capabilities, e.g. handling of a feature model with only simple geometries and attributes.

A ***complex analytic application*** is defined as an application in which a “complete” ontology reasoning would be employed, e.g. an application layered over a triple-store. Application profiles supporting complex analytic applications are profiles-of-ontologies that are still ontologies but whose utility is restricted to some particular domain-of-discourse (e.g., maritime safety-of-navigation, rather than all-of-geography).

One challenge of this task is to define and properly characterize the notion of **Application Profiles** that are based on ontologies. While the definition of Application Profile is pretty well-established for UML and XML Schema-based (e.g. GML application profiles), application profiles based on ontologies are not yet well-defined in the industry. There are some ongoing efforts at World Wide Web Consortium (W3C) attempting to describe application profiles for describing dataset metadata using the Data Catalog (DCAT) ontology and its derivatives DCAT-AP and GeoDCAT.

The following requirements have been addressed by the work documented in this Engineering Report:

- Investigate the derivation of ontology(ies) having different levels of complexity from a single application schema, to support different types of applications (simple linked data and complex analytic). This work will result in a better understanding of the desirable characteristics of ontologies for these applications. Further, it will provide experience on how such ontologies can be created. The results will be complemented by publicly available sample NEO-conformant RDF individual data.

Unfortunately, due to budget and time constraints (including the lengthy analysis phase needed to characterize application profiles for Linked Data applications), no sufficient time was spent on reviewing and refining the application profiles and sample dataset produced for this OGC Testbed-14. More work may be needed in future testbeds.

1.2. Prior-After Comparison

The current version of RDF representation of NEO, v8.0, is derived from the UML model of the National System for Geospatial Intelligence (NSG) Application Schema (NAS) using ShapeChange and a set of conversion rules defined in ISO 19150-2. The resulting ontology contains a very large set of classes and properties that addresses a large number of communities within the National Geospatial Intelligence Agency (NGA). Specific Communities of Interest (CoI) typically deal with a small subset of classes and properties to describe their domain. These communities use this information with a variety of applications ranging from simple linked data applications typically used for navigation or data capture, to more complex analytic applications where reasoning based on the ontology axioms is involved.

The goal of this OGC Testbed-14 work is to characterize and define application profiles (in particular NEO profiles) that can be used by simple linked data applications and complex analytic applications. The goal is to define application profiles that contain only the classes and properties of a specific Community-of-Interest, while preserving the semantic integrity and coherence of the application profile with the original ontology it is derived from.

1.3. Recommendations for Future Work

The following recommendations are made for future work in OGC Testbeds:

- **SHACL support in ShapeChange**: This ER increases OGC community understanding of Application Profiles for OWL-based ontologies. In Testbed-14, the focus of the analysis for converting Object Constraint Language (OCL) constraints was on OWL as the target language. However, some OCL language constructs (e.g. variables, addition, subtraction) and specific OCL constraints (that instances of a specific class are generally not allowed) cannot be represented in OWL. However, it may be possible to encode these constraints as Shape Constraint Language (SHACL) constraints, so that they can be checked for an RDF dataset. Future work should therefore analyze the conversion of OCL constraints to SHACL. If it turns out that SHACL can represent OCL constraints, enhance ShapeChange accordingly. Another aspect could be to enhance ShapeChange to derive profile definitions using SHACL.
- **RDF Application/Validation Profile based on multiple ontologies and SHACL**: The focus of

this ER was mostly on deriving application profiles as a strict subset of an ontology and logical coherence with the base ontology it is derived from. However typical application profiles are often based on one or more based ontologies and often focus on validation rules for specific applications (e.g. data capture). SHACL is emerging as a powerful standard to perform validation, data capture and build application profiles. For future testbeds, investigation should be conducted to encode RDF application profiles in SHACL documents and that are derived from multiple ontologies, so they can be used for validation and data capture. The task should leverage recent work conducted by the DCAT Revision W3C Working Group (WG) and be applied to NEO but also OGC Semantic Registry Information Model (SRIM) application profiles.

- **Metamodel for RDF Application Profiles**: This ER proposes an initial metamodel for describing RDF application profiles. It aims at facilitating search and discovery of application profiles based on specific ontologies. Testbed-12 investigated an Application Programming Interface (API) for a semantic mediation service and defined an SRIM profile for a semantic registry to represent schemas and schema mappings. This work can be extended and refined to accommodate RDF application profiles, so they can be searched and discovered by registry clients and used by a semantic mediation service.
- **SHACL Service**: SHACL is a W3C standard to encode RDF application profiles and support data capture and validation rules. There are a number of SHACL application profiles that are emerging in the data catalog community (DCAT profiles), but there is currently no good practice on how to manage, search and use these SHACL profiles. Future testbed should investigate and design best practices and APIs that address these needs.

1.4. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Contacts

Name	Organization
Stephane Fellah	Image Matters LLC

1.5. Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

Chapter 2. References

The following documents are referenced in this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

- IETF: RFC 2119 Key words for use in RFCs to Indicate Requirement Levels, https://www.ietf.org/rfc/rfc2119.txt
- ISO: ISO 19125-1:2004 ,Geographic information — Simple feature access — Part 1: Common architecture, https://www.iso.org/standard/40114.html
- ISO: ISO/IEC TR 10000-1:1998 - Information technology - Framework and taxonomy of International Standardized Profiles - Part 1: General principles and documentation framework
- NGA: NGA.STND.0064_1.0_NEO National System for Geospatial Intelligence Enterprise Ontology (NEO) Standard (2017-09-15) Edition 1.0, https://nsgreg.nga.mil/NSGDOC/files/doc/Document/NEO_Standard_Ed1.0.pdf
- ISO: ISO/IEC 19507:2012, Information technology - Object Management Group Object Constraint Language (OCL) - apart from introductory material identical to the OMG specification Object Constraint Language v2.3.1, https://www.omg.org/spec/OCL/2.3.1/PDF
- OGC 11-052r4, OGC GeoSPARQL - A Geographic Query Language for RDF Data, http://www.opengis.net/doc/IS/geosparql/1.0
- OGC 12-093, OGC OWS-9 System Security Interoperability (SSI) UML-to-GML-Application-Schema (UGAS) Conversion Engineering Report, available online at https://portal.opengeospatial.org/files/?artifact_id=51784
- W3C: OWL 2 Web Ontology Language, Direct Semantics (Second Edition), W3C Recommendation 11 December 2012, http://www.w3.org/TR/2012/REC-owl2-direct-semantics-20121211/
- W3C: OWL 2 Web Ontology Language, Structural Specification and Functional-Style Syntax (Second Edition), W3C Recommendation 11 December 2012, available online at http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/
- W3C: SKOS Simple Knowledge Organization System Reference, W3C Recommendation 18 August 2009, http://www.w3.org/TR/2009/REC-skos-reference-20090818/
- Shapes Constraint Language (SHACL), W3C Recommendation 20 July 2017, https://www.w3.org/TR/shacl/

Chapter 3. Terms and definitions

For the purposes of this report, the definitions specified in Clause 4 of the OWS Common Implementation Standard OGC 06-121r9 [https://portal.opengeospatial.org/files/?artifact_id=38867&version=2] shall apply. In addition, the following terms and definitions apply.

- Profile

```
A set of one or more base standards or subsets of base standards, and, where
applicable, the identification of chosen clauses, classes, options and parameters
of those base standards, that are necessary for accomplishing a particular
function. [adapted from ISO/IEC TR 10000-1:1998]
```

- Application Profile [DCAT-AP]

```
An Application Profile is a specification that re-uses terms from one or more base
standards, adding more specificity by identifying mandatory, recommended and
optional elements to be used for a particular application, as well as
recommendations for controlled vocabularies to be used.
```

- RDF Application Profile (new proposal)

```
An RDF Application Profile is a specification that re-uses terms from one or more
base ontologies using standard based on RDF Schema (RDFS), adding more specificity
by identifying mandatory, recommended and optional classes and properties to be
used for a particular application, as well as recommendations for controlled
vocabularies to be used and validation constraints that do not affect the
underlying model of reality expressed by ontologies.
```

- Application Schema

```
conceptual schema for data required by one or more applications [ISO 19101]
```

- Concept

```
structured conceptual element including entity types and properties (abstractions
of real-world objects) defined in formal models (e.g. UML, OWL) to enable
understanding of structured datasets.
```

- Conceptual Model

```
model that defines concepts of a universe of discourse [ISO 19101]
```

- Conceptual Schema

  ```
  formal description of a conceptual model [ISO 19101]
  ```

- Application Profile Characteristic

  ```
  logical and validation features of an RDF application profile.
  ```

- Linked Data

  ```
  a set of best practices for publishing and connecting structured data on the Web.
  Data from heterogeneous sources can be combined using typed links. Key technologies
  that support Linked Data are the Uniform Resource Identifier (URI), HyperText
  Transfer Protocol (HTTP) and RDF. A URI is a generic means to identify entities or
  concepts in the world. HTTP is a universal mechanism for retrieving resources, or
  descriptions of resources, RDF is a generic graph-based data model which is used to
  structure and link conceptual data.
  ```

> Linked Data is the data format that supports the Semantic Web. The basic rules for Linked Data are defined as:
>
> - Use URIs to identify things.
> - Use HTTP URIs so that these things can be referred to and looked up ("dereferenced") by people and user agents.
> - Provide useful information about the thing when its URI is dereferenced, using standard formats such as RDF/XML
> - Include links to other, related URIs in the exposed data to improve discovery of other related information on the Web.
>
> — W3C Semantic Web Wiki [https://www.w3.org/2001/sw/wiki/Semantic_Web_terminology#linked_data]

- Simple Linked Data Application

  ```
  An application which uses linked data with limited ontology-based reasoning. The
  purpose of a simple RDF application profile is to capture the list of valid
  concepts (classes, properties) and basic cardinality restrictions, in order to
  support the encoding of instance descriptions. The set of concepts and restrictions
  is generally insufficient for performing complex reasoning.
  ```

- Complex Analytic Application

```
An application in which “complete” ontology-based reasoning would be employed, e.g.
an application layered over a triple-store. RDF application profiles supporting
complex analytic applications are subsets-of-ontologies that are still ontologies
but whose utility is restricted to some particular domain-of-discourse (e.g.,
maritime safety-of-navigation, rather than all-of-geography).
```

- Constraint Component

```
A constraint component is defined by a URI and a set of parameters (mandatory and
optional) and is associated with a validator which provides instructions (for
example expressed via SPARQL queries) on how the specified parameters are used to
validate data. Validating an RDF term against an RDF shape involves validating the
term against each constraint where the RDF shape has values for all mandatory
parameters of the component of the constraint, using the validators associated with
the respective component. [SHACL]
```

- RDF Shape

```
A collection of constraints to be applied to some node in an RDF graph. Under the
Closed World Assumption (CWA), every property attached to that node must be
included in the RDF shape.
```

- Node Shape

```
Constraints about a given focus node in an RDF instance data graph.
```

- Property Shape

```
Constraints about a given property and the values of a path for a node in an RDF
instance data graph.
```

- Path

```
A path in an RDF graph G from RDF term n to RDF term m is a finite sequence of RDF
triples in G such that the subject of the first RDF triple is n, the object of the
last RDF triple is m, and the object of each RDF triple except the last is the
subject of the next
```

- Focus Node

```
An RDF term that is validated against an RDF Shape using the triples from an RDF
instance data graph.
```

- Term

  ```
  Lexical items including words, phrases, or abbreviations (expressed in text
  strings)
  ```

- Value Node

  ```
  Reference node used in the specification of constraint components, as follows: (i)
  For node shapes, the value nodes are the individual focus nodes in an RDF instance
  data graph, forming a set with exactly one member. (ii) For property shapes, for
  given path 'p' the value nodes are the set of nodes in the RDF instance data graph
  that can be reached from the focus node given the path mapping of 'p'.
  ```

3.1. Abbreviated terms

- BFO Basic Formal Ontology
- CWA Closed World Assumption
- DASH Data Shapes Vocabulary
- DCAT Data Catalogue Vocabulary
- DCAT-AP DCAT Application profile for data portals in Europe
- DDL Data Definition Language
- DOLCE Descriptive Ontology for Linguistic and Cognitive Engineering
- DXWG Dataset Exchange Working Group
- ER Engineering Report
- GEOINT Geospatial Intelligence
- GML Geography Markup Language
- IRI Internationalized Resource Identifier
- ISO International Organization for Standardization
- JSON JavaScript Object Notation
- JSON-LD JSON for Linked Data
- NAS NSG Application Schema
- NCV NSG Core Vocabulary
- NEO NSG Enterprise Ontology
- NSG U.S. National System for Geospatial Intelligence
- OCL Object Constraint Language
- OGC Open Geospatial Consortium
- OWA Open World Assumption

- OWL Web Ontology Language
- RDF Resource Description Framework
- RDFS RDF Schema
- SHACL Shapes Constraint Language
- ShEx Shape Expression
- SKOS Simple Knowledge Organization System
- SPIN SPARQL Inferencing Notation
- SQL Structured Query Language
- SUMO Suggested Upper Merged Ontology
- SPARQL SPARQL Protocol and RDF Query Language
- TSC Technical Steering Committee
- UGAS UML to GML Application Schema
- UML Unified Modeling Language
- URI Uniform Resource Identifier
- URL Uniform Resource Locator
- W3C World Wide Web Consortium
- WFS Web Feature Service
- WKT Well Known Text
- XML Extensible Markup Language
- XPath XML Path Language
- XSLT Extensible Stylesheet Language Transformations

Chapter 4. Overview

Section 5 provides an analysis of the difference between semantic and data-centric representations. The analysis aims at providing a better understanding of the difference between application profiles based on syntactic-based application schemas (such as XML Schema, Relax NG,Java Script Object Notation (JSON) Schema) and semantic-based application profiles using ontologies.

Section 6 focuses on the similarities and differences between UML and OWL modeling.

Section 7 provides a definition of ontology, an overview of the different types of ontologies, a comparison of Open World Assumption (OWA) versus Closed World Assumption (CWA) and these assumptions relate to the existing W3C standards OWL and SHACL.

Section 8 compares OWL and SHACL standards to better understand their roles in the definition of application profiles for ontologies.

Section 9 distinguishes RDF application profiles versus RDF Validation Profiles and discusses more in-depth the RDF application profile characteristics and constraints for simple linked data applications and complex analytic applications.

Section 10 documents a proposal for defining metadata about application profiles so they can be more easily searched and discovered, as well as being linked to other standards in a catalog service.

Chapter 5. Semantic versus data-centric representation

To better understand the difference between application profiles based on syntactic-based application schemas (such as XML Schema, Relax NG, Java Script Object Notation (JSON) Schema) and semantic-based applications, an analysis of the difference between semantic and data-centric representation is needed.

Knowledge assertions are expressed by RDF, as sets of triples, each triple being rather like the subject, verb and object of an elementary sentence. In RDF, a document makes assertions that particular things (people, Web pages or whatever) have properties (such as "is a sister of," "is the author of") with certain values (another person, another Web page). This structure turns out to be a natural way to describe the vast majority of the data processed by machines [1] and allow anyone to say anything about anything. This is in tension with conventional data practices which reject data with any assertions that are not recognized by the schema. For Structured Query Language (SQL) schemas, this is enforced by the data storage itself; there is simply no place to record assertions that do not correspond to some attribute in a table specified by the Data Definition Language (DDL). XML Schema offers some flexibility with constructs like *<xs:any processContents="skip">* but these are rare in formats for the exchange of machine-processable data. Schema-based systems put emphasis on strict protocol-adherence, which is an unforgiving requirement and makes current systems very brittle when there is any deviation from protocol or alternative interpretation of the semantics of the protocol. Because of this underlying brittleness, software tends to be built up into layers, which introduces lock-in and sedimentation of ideas and protocols [2].

The main issue with a schema-based system (data-centric) versus a semantic-based system is that in the former the semantic information is implicit and requires out-of-line information to properly interpret the information (by human interpretation or writing code). In a semantic-based system, ontologies make the semantic information explicit and allow machines to immediately interpret the information without the need for writing new code. Semantic-Enabled Approaches add a machine-encoded knowledge layer to existing environments. This layer captures the conceptual meanings, significance, relevance and relationships of all relevant information required for use in a target mission domain. The layer also encodes business rules that are used by enhanced reasoning services to draw inferences and make sense of mission concepts. Finally, the knowledge layer integrates with an array of analytical and reasoning services to enhance information triage, fusion, situation awareness, and sense making. The layer facilitates a higher level of utility, automation, interoperability, extensibility and flexibility, beyond that which is possible with data-centric approaches. It also dramatically reduces the cognitive burden on users.

Some notable advantages of this approach are:

1. The knowledge layer provides a more user friendly and mission-relevant unified conceptual view of all pertinent data and services for a target mission domain. Users are able to interact with familiar, well-understood concepts, rather than deal at the level of fixed data schemata with its commensurate semantic and contextual ambiguities. This allows users to obtain relevant actionable information more readily, which in turn leads to making better decisions more quickly. User productivity and effectiveness increases with the reduction in cognitive

burden.

2. There is substantially more flexibility in accommodating data evolution, variety and veracity. It is easier to add new data sets, make changes to existing data sets, and accommodate data set differences. Changes in information requirements will first affect the "knowledge layer", reducing effects on databases and software.
3. Business rules are encoded in the knowledge layer, where they can be more easily changed without changing business applications. (Current approaches require that business logic must be wired into business applications… a costly, inflexible approach.)
4. The vastly improved filtering, fusion, analytics and reasoning services provide an enhanced information triage mechanism for quickly finding "the needle in the haystack", locating patterns across a complex array of data, and making inferences that are not directly exposed in data (i.e., "determine the haystack from the needles"). User productivity and effectiveness increases with the increased level of automation.

Table 1 presents the main issues with a data-centric approach addressed by the Semantic-Enabled Approach, along with its benefits.

Table 1. Data-Centric Issues with Corresponding Semantic-Enabled Approach and Benefits

Issues with Data-Centric Approaches	**Semantic-Enabled Approach**	**Value Proposition**
Data model standardization relies upon homogeneous data description and organization. *This imposes strict adherence to a standard that is defined at the syntactic-schematic level, whereupon it is harder to achieve consensus and flexibility. Modelers struggle between producing simpler models for which it is easier to gain consensus, but harder to achieve desired business reality, versus those seeking richer models that are closer to reality but have unwanted complexity.*	• A semantic-enabled approach employs a standards-based formal, sharable framework that provides a conceptual domain model to accommodate various business needs. Decentralized model extensions can be accommodated without adversely affecting existing information infrastructure.	• Allows decentralized extensions of the domain model • Accommodates heterogeneous implementations of the domain model (lessens impact on systems; reduces cost) • Shareable machine-processable model and business rules; reduces required code base

Issues with Data-Centric Approaches	Semantic-Enabled Approach	Value Proposition
Data-centric approaches increase the chance for multiple interpretations and misinterpretations of data. *Data interpretation requires knowledge of its semantics (e.g., meanings, significance, and relevance) and surrounding context. Data-centric approaches are unable to capture these semantics and context, which are in turn required for automated fusion, analytics, and reasoning.*	• Semantic-enabled approaches encode data characteristics in ontology. By formalizing the semantic and business rules unambiguously in a declarative ontology, software can use off-the-shelf semantic components to interpret, infer and validate domain data, thus reducing interpretation errors.	• Increased software maintainability • Improved data interpretation and utility • Actionable information for the decision maker
Data model implementations have limited support for business rules, and lack expressiveness. *Data-centric implementations encode business rules using software or database programming languages. Additional programming is necessary to apply business rules when using the data. Robust conceptual and contextual meanings of information may not be captured in the model. The risk is high for inconsistent conceptual encoding and interpretation in each implemented system.*	• Standards-based knowledge encoding (OWL, SPARQL Rules) captures formal conceptual models and business rules, providing explicit, unambiguous meanings for use in automated systems. With richer semantic and contextual expressiveness, automated systems are less complex to design and develop. Proper interpretation and use are more consistent across enterprise systems.	• Reduction of software and associated development cost • Conceptual models and rules that provide enhanced meaning, thus reducing the burden on users • Unambiguous interpretation of domain model; greater consistency in use

Issues with Data-Centric Approaches	Semantic-Enabled Approach	Value Proposition
Data-centric implementations are inflexible when data requirements change. *Whenever business rules and semantic meaning are encoded in a programming language, changes impact the full development life cycle for software and data. When the change includes a conceptual change (new/enhanced business concept), the full standardization process must also be executed.*	• The semantic-enabled approach uses an ontology that contains a flexible, versatile conceptual model that can better accommodate the requirements of each stakeholder in the business domain. Changes or extensions are integrated and implemented by enhancing the domain ontology. Older concepts and heterogeneity can still be supported.	• Increased flexibility to accommodate stakeholder needs; Decentralized and organic evolution of the domain model • Changes only impact affected stakeholders, not others; reduces software updates • Software adapts to domain model as ontology evolves • The enterprise can better keep up with changing environment/requirements
Data-centric approaches require additional software development (tools) to integrate across several domains. *The heterogeneity of data schemas and business models in alien domains requires additional software development (or tools) to integrate and make sense of the alien data in the local domain.*	• Standards-based knowledge technologies provide the means to exchange knowledge, schemas/rules and query knowledge. Knowledge representations are non-disruptively layered on top of existing information assets. They **enhance** rather than displace existing information resources. • Using semantic-enabled technologies, different data sources can be accessed in their native form and linked-fused-reasoned about on-the-fly for more creative uses.	• Leverages and enhances “As-Is” enterprise; preserves investments • Open standards enhance interoperability (the knowledge layer is the “last rung in the interoperability ladder”) • Enables use of diverse Web and enterprise data sources for full business context; enhances business applications • Easy to work across different domains and answer more challenging, relevant business questions

Issues with Data-Centric Approaches	Semantic-Enabled Approach	Value Proposition
Data-centric approaches require that data inferencing and validation rules are encoded in software or delegated to human-intensive validation processes. *Reliable data that is essential for critical systems, inferencing, and effective decision support, requires rules that support inferencing and validation.*	• Semantic-enabled approaches use a formal language (OWL) that provides well-defined semantics in a form compliant with off-the-shelf software that automates data inferencing and validation. • Semantic-enabled approaches can accommodate situations where information may be missing or incomplete.	• Employs off-the-shelf software for inferencing and validation • Reduction of validation and testing in the development process • Uses all available data from sources, including inferences, while accommodating cases of missing/incomplete information
Data-centric approaches presume a priori knowledge of data utility. *Semantics are pre-wired into applications based upon data verbosity, conditions and constraints. Changes in data directly impact code.*	• Encoding the conceptual model and rules explicitly using OWL enables rapid integration of new/changed data. Software accesses data through the “knowledge layer” where it’s easier to accommodate changes without rewriting software.	• Reduced software maintenance owing to emergent data perturbations • Software quickly adapts to evolving domain model • New information are readily introduced and understood in their broader domain context

Chapter 6. UML versus OWL Modeling

The NEO ontology is derived from object-oriented UML models. Modeling ontologies is very different than modeling object-oriented software artifacts. This section attempts to outline the differences and similarities between UML modeling and OWL modeling.

UML is a notation for modeling object-oriented software artifacts. UML is used in the *Model-Driven* development process. OWL is a notation for knowledge representation. OWL is used in the *Ontology-Driven* development process. These are two quite different problems so asking about expressivity differences would lead to an 'apples-versus-oranges' comparison. In particular thinking in object-oriented terms when working with OWL or RDFS will almost always lead a developer astray.

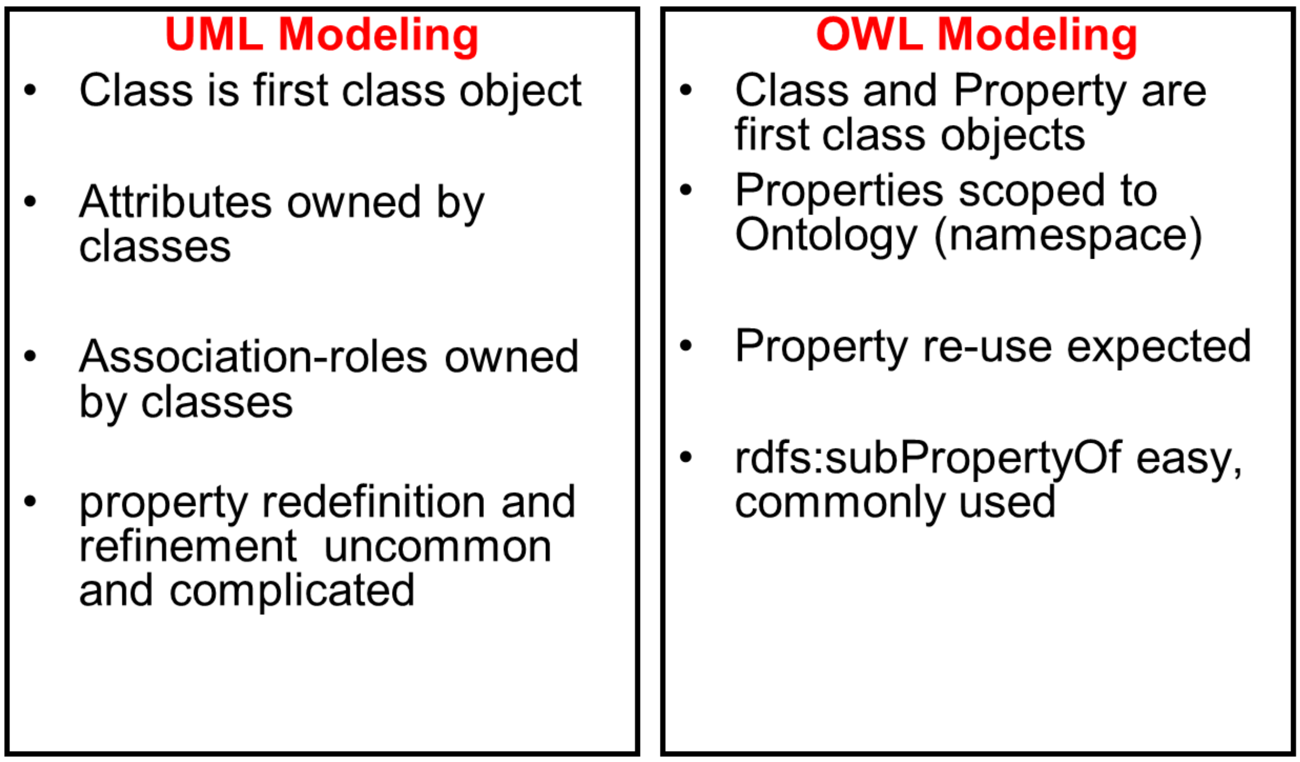
To take an example, both OWL and UML have things called "classes" and those classes are related in some sort of hierarchy that you can depict via a diagram. However, they are not the same notion at all.

In UML, a class is a software object acting as a template (or frame) for objects. You can create instances of a class and that creation process has procedural semantics involving things like assigning values to attributes (a.k.a. slots, members). So procedural notions, such as a default value for an attribute which is resolved at construction time, is a simple well-defined thing in UML. Instance objects also have some associated storage, so UML distinguishes between containment and association - things are stored in the object v. things that are outside the object.

Instance objects have a runtime semantic, so you can have notions of static values and mutable values. Attributes and the association role are owned by classes, i.e. their existence depends on the existence of the classes. They are not first-class modeling concepts. Asserting a property on a class (or discovering such an assertion) cannot lead to inferring that it is a member of further classes (for example the fact that a class has property length does not lead to the conclusion that the class is a spatial object).

None of that is true in RDFS or OWL or similar languages. In OWL, modeling a class is a category, it is a label given to a set of things in a domain. In OWL terms, a class is simply the set of things which are members of that set. So RDFS/OWL has no notion of instantiating or constructing an instance and so no notion of default values for instance creation. OWL has the notion of Individuals that belong to a Class. However, if you have a resource and it meets the criteria for membership of the class, then it is a member of that class; if you didn’t already know that your resource was in that class then you can derive this information by reasoning.

Asserting a property on a resource (or discovering such an assertion) can lead to you inferring that it is a member of further classes. Resources (in RDFS terms, Individuals in OWL terms) are not things with state, storage or runtime semantics. They are simply identifiers for things in your domain. Assertions about those resources can be made, found or derived but the resources themselves are not objects with slots and so notions like static and public/private have no meaning here. This is not a limitation of OWL or RDFS so much as the fundamental nature of what you are modeling. OWL classes are like labels for concepts, UML classes are like templates that define a runtime object and its storage.



UML Modeling

- Class is first class object
- Attributes owned by classes
- Association-roles owned by classes
- property redefinition and refinement uncommon and complicated

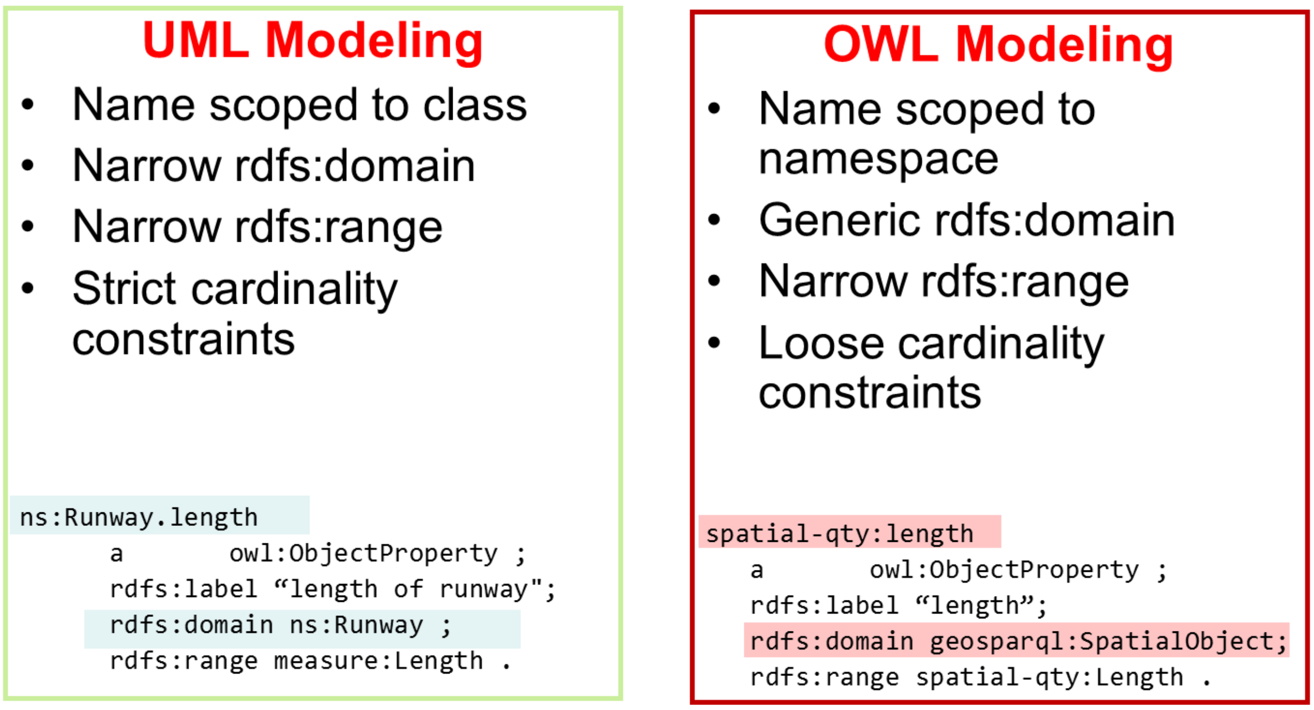
OWL Modeling

- Class and Property are first class objects
- Properties scoped to Ontology (namespace)
- Property re-use expected
- rdfs:subPropertyOf easy, commonly used

Figure 1. UML Modeling versus OWL modeling

In addition, in OWL, properties are first class objects and are scoped to the ontology (namespace). Property re-use in different classes is expected and commonly used as well as specialization of properties (using subPropertyOf hierarchy). Property redefinition and refinement is very uncommon and complicated to model in UML. Figure 1 and Figure 2 summarize the difference between UML modeling and OWL Modeling.

In UML, the domain and range of the properties are very narrow as the properties belong by default to the Class, thus the property has a default scope to the class. Additional annotations are needed to indicate that the properties are global (as defined in ISO 19150-2). In OWL, properties are typically defined globally and thus have a broader domain and narrow range. Further, cardinality of OWL properties is looser than cardinality in UML model. Figure 2 shows an example where the length of a runway property has a domain Runway. In OWL, the length can be defined globally and has a broader domain of SpatialObject. The conversion from UML to OWL using the default settings generates ontology with very limited expressiveness and reusability. In OWL, reuse of external vocabularies is commonly expected. The choice of the vocabularies requires the expertise of an ontologist and a subject-matter expert, and cannot be automated.



UML Modeling

- Name scoped to class
- Narrow rdfs:domain
- Narrow rdfs:range
- Strict cardinality constraints

```
ns:Runway.length
        a        owl:ObjectProperty ;
        rdfs:label “length of runway";
        rdfs:domain ns:Runway ;
        rdfs:range measure:Length .
```

OWL Modeling

- Name scoped to namespace
- Generic rdfs:domain
- Narrow rdfs:range
- Loose cardinality constraints

```
spatial-qty:length
   a        owl:ObjectProperty ;
   rdfs:label “length”;
   rdfs:domain geosparql:SpatialObject;
   rdfs:range spatial-qty:Length .
```

Figure 2. Property modeling in UML versus OWL

Despite all these differences there is however some connection between both modeling languages. Both UML and OWL are languages for modeling. They are used for modeling totally different things and so have different capabilities. They are completely different approaches to semantics but have some structural similarities. MOF (Meta Object Facility) is the meta-modeling tool on which UML is based. It is a language in which you can express other modeling languages. So, UML is specified in MOF. You can do the same for RDFS and OWL - that is, express their metamodels in MOF. This is what ODM (Ontology Definition Metamodel) provides which provides a profile for writing RDF and OWL within UML. It also includes mappings between UML and OWL as well as mappings amongst RDF, RDFS, Common Logic and Topic Maps.

This discussion should highlight why modeling ontologies cannot be done by simply converting (automatically or not) UML diagrams to OWL. Ontologies are better designed from the ground-up but can be informed by concepts expressed in UML models.

Chapter 7. Ontology overview

7.1. Definition

Ontologies provide a way to share the semantics of concepts in some area-of-interest. It is all about common understanding of essential concepts. A generally accepted short definition of ontology is given by Gruber [3]:

```
An ontology is an explicit specification of a conceptualization.
```

The term conceptualization is defined as follows:

```
A conceptualization is the combination of objects, concepts, and other entities that
are assumed to exist in some area of interest and the relationships that hold among
them. A conceptualization is an abstract, simplified view of the world that we wish to
represent for some purpose.
```

Ontologies are used to formally represent an inevitably reductionist view of reality by defining the concepts (entities and properties) and logical axioms found in the world. These concepts are typically formalized using Description Logic (DL), which can be leveraged to infer additional information about things using deductive logic.

7.2. Ontology classification

Ontologies are often classified based on scope of their domain of discourse. The following classification of ontologies are typically used:

Upper-ontologies: An *upper-ontology* (also known as a **foundation ontology** or **top-level ontology**) is an ontology which consists of very general terms (such as "object", "process", "role", "function") that are common across all domains. An important function of an upper ontology is to support broad semantic interoperability and alignment among a large number of domain-specific ontologies and to provide a common starting point for developing more specific domain ontologies. Upper ontologies are limited to concepts that are meta, generic, abstract or philosophical. Examples of upper ontologies include Suggest Upper Merged Ontology (SUMO), OpenCyc, Basic Formal Ontology (BFO) and Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE).

Mid-level ontologies: A *mid-level ontology* serves as a bridge between abstract concepts defined in an upper ontology and low-level domain specific concepts specified in a domain ontology. While ontologies may be mapped to one another at any level, the mid-level and upper ontologies are intended to provide a mechanism to make this mapping of concepts across domains easier. Mid-level ontologies may provide more concrete representations of abstract concepts found in upper ontologies. This ontology category also encompasses the set of ontologies that represent commonly used concepts, such as Location and Time. These commonly used ontologies are sometimes referred to as **utility** ontologies [4].

Domain ontologies: A *domain ontology* specifies concepts particular to a domain of interest and represents those concepts and their relationships from a domain specific perspective. While the same concept may exist in multiple domains, the representations may vary widely due to the differing domain contexts and assumptions. Domain ontologies may be composed by importing mid-level ontologies. They may also extend concepts defined in mid-level or upper ontologies. Reusing well established ontologies in the development of a domain ontology allows one to take advantage of the semantic richness of the relevant concepts and logic already built into the reused ontology [4]. Not everything that is part of an entity within a given domain is also part of that domain. For example, every human being has molecules as parts, but molecules do not form part of the domain of human geography or sociology for example.

Application ontologies: An *application ontology* is an ontology that is created to accomplish some specified local task or application. Portions of multiple existing ontologies can be reused in an application ontology, which will typically also contain new ontology content created to address its specific local purpose. They are typically narrower in scope than domain ontologies, as they are designed for a specific application, thus less reusable for other domain or applications.

7.3. Open World Assumption

An important aspect of DL knowledge bases (and OWL ontologies) is that they are based on the **Open World Assumption** (OWA), which has implications for the reasoning on them. OWA means that there is an assumption that everything can be true until it is explicitly stated that it does not exist. This implies that the absence of information in a knowledge base only indicates the lack of knowledge. In this respect, instances of DL concepts are treated differently as instances of database schemas [5]. However, the actual openness of that world is affected when adding ontological constraints such as domain, range or cardinality restrictions. These constraints, which provide context and meaning, and also provide the rules for inferencing, are sometimes confused with the rules that metadata-producing communities commonly employ for quality control (data validation) and data consistency.

7.4. Closed World Assumption

The opposite of OWA is called **Closed-World Assumption** (CWA). It is the presumption that a statement that is true is also known to be true. Therefore, conversely, what is not currently known to be true, is false. Decisions on CWA vs. OWA determine the understanding of the actual semantics of a conceptual expression with the same notations of concepts. In the context of knowledge management, the closed-world assumption is used in at least two situations: (1) when the knowledge base is known to be complete (e.g., a corporate database containing records for every employee), and (2) when the knowledge base is known to be incomplete, but a "best" definite answer must be derived from incomplete information (typically returning a default value). In OWA, the use of default value is not allowed and assume that the lack of knowledge does not imply falsity, it is just not known. When developing application profiles for an ontology, it is possible that some constraints needs to be expressed using CWA. These constraints are typically used for data validation, data completion, but also reasoning in CWA. They often cannot be expressed in OWL. In particular, OWL ontologies do not provide constraints that one would typically view as data validation. Other standards, such as Shape Constraint Language (SHACL), Shape Expression (ShEx) are a better fit for addressing the aspect of data validation. These standards can be seen as complementary modeling techniques to OWL.

Chapter 8. OWL versus SHACL

Now that there is an understanding of OWA and CWA, this section will focus on the similarities and differences between OWL and SHACL, which are both used to define RDF application profiles. OWL has been designed for classification tasks (inferencing in an "open world"), while SHACL covers data validation (in a "closed world") similar to traditional schema languages. Given this division of roles, both technologies can be used together or individually. Further, SHACL can also be used for general purpose rule-based inferencing.

The next sub-sections explain the use of OWA versus CWA and their implications in OWL and SHACL. The observations are based on the excellent comparison of OWL versus SHACL made by Holger Knublauch [6](author of Protege, Topbraid applications and SPIN and SHACL specifications).

8.1. OWL

OWL is designed as an extension of RDF Schema. OWL adds in the language **restrictions** properties such as *owl:maxCardinality* , property characteristics (transitive, symmetric,functional, inverseFunction, reflexive,etc), logical classes (union, intersection, complement) and richer datatype expressions. Like RDFS, OWL was designed for inferencing. In OWL, restrictions are actually used for inferences not for data constraints. The implication of this usage can lead to surprising outcomes for OWL novices. For example, assuming there is an *owl:maxCardinality 1* restriction stating that a person can only have 1 value for *ex:hasFather* and there is an instance of *ex:Person* that has two *ex:hasFather* values, then an OWL processor will infer that these two values represent in fact the same real-world entity, just with different URIs. This topic is sometimes called **Unique-Name Assumption** (or, in the case of OWL: the lack of that assumption).

Holger [6] notes that "OWL’s interpretation is based on a (rather philosophical) distinction between a resource (URI) and a real-world entity that is represented by that resource. If you follow the OWL spec, you cannot send a set of instances to an OWL processor and ask whether these instances "match" or "conform to" the given schema in the same way that you would send an XML file to an XML Schema validator. Instead, an OWL processor will actually add to the data in attempt to conform to the restrictions rather than report an error. If the addition of new facts results in logical contradictions, then the processor will report an error. However, the error is rarely traceable to the original data statement(s) that have caused the contradictions".

OWL properties are given a certain meaning (aka semantics) that make OWL-based data fit for the open world of the Web in which any RDF resource may link to any other RDF resource without having full control over which triples are actually present when the resource is in use. As a consequence of OWA, an application should not assume that the absence of a certain statement means that the statement is false. Holger [6] gives the following example: "if an OWL ontology states that the rdfs:range of the property ex:hasFather is ex:Person and an application only sees a triple stating ex:John ex:hasFather ex:Bob and nothing else then it should not assume that ex:Bob is an not instance of ex:Person. In fact, the application should assume the opposite, and automatically infer the triple ex:Bob rdf:type ex:Person. Another surprise for newer OWL users is that missing a value for a property with a restriction of owl:minCardinality 1 is not reported as an error by an OWL processor, because more data may appear at any time to satisfy that restriction under the Open-World Assumption".

8.2. SHACL

Although data validation is an important practical use case for the RDF stack, until SHACL came around, there was no W3C standard mechanism for defining data constraints. Over time, people became creative in working around this limitation. Many tools simply decided that for all practical purposes, the open-world and non-unique-name assumptions should simply be ignored. OWL-aware tools including TopBraid and Protégé, for example, provide data entry forms that restrict users from entering more than one value if there is a corresponding owl:maxCardinality 1 restriction, or require the selection of a specific instance of ex:Person if that class is the rdfs:range of the property.

SHACL provides a high-level vocabulary with properties such as sh:minCount and sh:datatype as well as a fallback mechanism that allows users to express basically any constraint using SPARQL or (using the SHACL-JS extension) in JavaScript. The high-level vocabulary makes SHACL also a schema/ontology language, allowing tools to examine the structure of a class to, for example, suggest how instances of a class should be presented on an input form. Since OWL performs (limited, as described above) data validation through inferencing, it has no separation between data validation and reasoning. SHACL separates checking data validity from inferring new facts. Both, however, are possible with SHACL.

The W3C launched the RDF Data Shapes Working Group in 2014 to address the lack of a suitable standard to express constraints and schemas with a Closed World Assumption. SPARQL Inferencing Notation (SPIN) and other member submissions such as IBM's Resource Shapes were used as inputs by the Working Group and the term **shape** was adopted to mean a collection of constraints that apply to targeted RDF resources. SHACL was standardized as a W3C Recommendation in July 2017.

A key differentiator between SHACL and OWL is that SHACL is extensible while OWL is limited to exactly the features that have been specified by the OWL committee. SHACL-SPARQL and SHACL-JS specifications provides more details on some SHACL extension points. In particular, SPARQL includes the concept of variables which has no equivalent in OWL and therefore makes it impossible to express many real-world use cases in OWL alone.

SHACL also includes a very rich results vocabulary in which the results of the validation process are returned. For example, it includes both human-readable and machine-readable pointers to specific data that violates SHACL constraints, the specific constraint type that was violated (e.g. sh:minCount), and it distinguishes errors from warnings and informational results. Even if OWL processors would return constraint violations, they would not have a standard way of expressing them.

8.3. Comparison of Built-in Constraint Types in OWL and SHACL

Table 2 summarizes the available kinds of constraints that are built (hard-coded) into OWL and SHACL processors. The table does not necessarily mean that OWL and SHACL are equivalent in their interpretation - as mentioned before there are fundamental differences in how OWL interprets restrictions (for inferencing) from how SHACL interprets constraints (for validation).

Note that SHACL offers its extension mechanisms (SPARQL,JavaScript) allowing anyone to create their own constraint types that can then be used with properties similar to the ones below. Such extension namespaces can be published on the web, for anyone to reuse. An example of such an extension namespace is the Data Shapes Vocabulary (DASH), which is indicated where suitable below. The SHACL Core vocabulary was designed with this extensibility in mind, i.e. although it covers the most common use cases, the comparison is not completely comprehensive.

Table 2. Comparison of built-in Constraint Types in OWL and SHACL

OWL	SHACL	Notes
Value Type Constraints		
owl:allValuesFrom	*sh:class or sh:datatype*	also checks well-formedness
-	*sh:nodeKind*	
Cardinality Constraints		
owl:maxCardinality	*sh:maxCount*	
owl:minCardinality	*sh:minCount*	
owl:cardinality	*sh:minCount + sh:maxCount*	SHACL is more verbose but less redundant for tools
owl:FunctionalProperty	*sh:maxCount 1*	In OWL: global axiom, in SHACL: local constraint
owl:InverseFunctionalProperty	*sh:maxCount 1 on a sh:inversePath*	In OWL: global axiom, in SHACL: local constraint
Value Range Constraints		
owl:onDatatype/owl:withRestrictions/xsd:minExclusive etc	*sh:minExclusive etc*	Datatype facets are considerably more verbose in OWL
String-based Constraints		
owl:onDatatype/owl:withRestrictions/xsd:minLength	*sh:minLength*	
owl:onDatatype/owl:withRestrictions/xsd:maxLength	*sh:maxLength*	
owl:onDatatype/owl:withRestrictions/xsd:length	*sh:minLength + sh:maxLength*	
owl:onDatatype/owl:withRestrictions/xsd:pattern	*sh:pattern*	OWL does not support sh:flags
owl:onDatatype/owl:withRestrictions/rdf:langRange	*sh:languageIn*	Different approach
-	*sh:uniqueLang*	
Property Pair Constraints		
owl:equivalentProperty	*sh:equals*	

OWL	SHACL	Notes
owl:propertyDisjointWith, owl:AllDisjointProperties	*sh:disjoint*	
owl:inverseOf	*sh:inversePath/sh:equals*	
-	*sh:lessThan*	
-	*sh:lessThanOrEquals*	
rdfs:subPropertyOf	*dash:subSetOf*	Or: combine SHACL with RDFS inferencing
owl:onProperty, owl:propertyChainAxiom	*sh:path*	SHACL supports arbitrary property paths, OWL does not
Logical Constraints		
owl:complementOf	*sh:not*	
owl:intersectionOf	*sh:and*	
owl:unionOf	*sh:or*	
owl:qualifiedMin/MaxCardinality	*1*	sh:xone
owl:disjointUnionOf	*sh:node/sh:or/sh:not*	This is verbose in SHACL
Shape-based (structural) Constraints		
rdfs:subClassOf, owl:equivalentClass	*sh:node*	
rdfs:subClassOf, owl:equivalentClass	*sh:property*	
owl:someValuesFrom	*sh:qualifiedMinCount 1 or dash:hasValueWithClass*	
owl:qualifiedMinCardinality etc	*sh:qualifiedMinCount etc*	
Property Characteristics		
owl:ReflexiveProperty	*sh:not/sh:disjoint in a node shape*	
owl:IrreflexiveProperty	*sh:disjoint in a node shape*	
owl:SymmetricProperty	-	
owl:AsymmetricProperty	-	
owl:TransitiveProperty	*sh:path with * operator*	
Other Constraints		
-	*sh:closed, sh:ignoredProperties*	
owl:hasValue	*sh:hasValue*	
owl:oneOf	*sh:in*	

OWL	SHACL	Notes
owl:hasKey	*dash:uriStart*	Approximation
owl:sameAs, owl:differentFrom, owl:AllDifferent	-	In SHACL every resource is distinct by default

Chapter 9. RDF Application Profiles

9.1. RDF Application Profiles versus RDF Validation Profiles

Based on the background information given in previous sections, two categories of profiles for RDF have been identified:

- **RDF Application Profiles**: profiles that support “complex analytic applications” profiles-of-ontologies that are **still ontologies** but whose utility is restricted to some particular domain-of-discourse (e.g., maritime safety-of-navigation, rather than all-of-geography). This is the **primary focus of this testbed**. RDF Application Profiles use ontologies declared elsewhere and merely specify how the terms of those vocabularies are constrained and packaged in matching sets of instance data. In RDF Application Profiles, constraints are expressed as logical constraints on data, not as data constraints on the underlying RDF vocabularies. Classes and properties can be reused in different application profiles without changing their underlying definition or affecting how they are used elsewhere. OWL Ontologies, in contrast, typically define logical constraints as an integral part of the vocabulary itself. To describe something using classes and properties from an OWL Ontology is to implicitly accept its underlying model of the universe.
- **RDF Validation Profiles**: Profiles that support “data validation/quality” describe the structure and constraints on the data content. The constraints are expressed as constraints on data, not as constraints on the underlying RDF vocabularies. Classes and properties can be reused in different application profiles without changing their underlying definition or affecting how they are used elsewhere. They can define constraints on data creation similar to those provided by XML schema or application-specific rules that are used for the validation of instance data. RDF Validation schemas can ensure the quality of data in ways not requiring inference schemas such as OWL Ontologies. Machine-actionable application profiles can document such rules both for quality control usage during data creation and data re-use, and for documenting shared data. While this is an important class of profiles that are useful for many applications, it is not the primary focus of this testbed. However, this class of profiles will require further investigation in future testbeds.

The following sections provide more details about the characteristics of RDF Application Profiles.

9.2. RDF Application Profile General Characteristics

RDF Application Profiles are useful as they allow the implementor to declare how they are using standard ontologies. In the context of working applications where there is often a difference between the ontology in use and the 'standard' namespace ontology, the following requirements need to be respected:

- may restrict the choice of options defined in base standards (OWL ontologies) to the extent necessary to achieve the objective of the profile. A profile may retain base standard options as options of the profile;
- shall not specify any requirements that would contradict or result in non-conformance to the base standards (OWL ontologies) to which it refers;

- may contain conformance requirements which are more specific and limited in scope than those of the base standard to which it refers.

Thus, by definition, conformance to a profile implies conformance to the set of base standards to which it refers. However, conformance to that set of base standards does not necessarily imply conformance to the profile [adapted from ISO/IEC TR 10000-1:1998]. That is, the profile may specifically exclude optional aspects of the base standards.

RDF Application Profiles are distinguished by a number of characteristics:

- May draw on one or more existing ontologies

The application profile may use elements from one or more different ontologies, but the application profile cannot create new elements not defined in existing ontologies. In the context of this testbed, NEO application profiles are defined as a strict subset of the NEO ontology.

- Introduce no new data elements

All elements in an application profile are drawn from elsewhere, from distinct ontologies. If an implementor wishes to create 'new' elements that do not exist elsewhere then (under this rubric) they must create their own ontology and take responsibility for 'declaring' and maintaining that ontology and its elements.

- May specify permitted ontologies and values

Often individual implementations wish to specify which range of values are permitted for a particular element, in other words they want to specify a particular restricted (or possibly extended) range for a property. The implementor may also want to specify mandatory schemes to be used for particular elements, for example particular date datatypes, particular format datatypes for personal names.

9.3. RDF Application Profile constraints

This section discusses the different types of RDF application profiles constraints and the most important characteristics of the RDF application profiles. When relevant, a comparison is done with characteristics of RDF Validation Profiles encoded in SHACL. A summary table of the OWL axioms used for simple Linked Data Application Profile and Complex Analytic Application is given at the end of this section.

9.3.1. Class Obligation

An RDF Application Profile is a specification that re-uses terms from one or more base ontologies, adding more specificity by identifying mandatory, recommended and optional classes and properties to be used for a particular application or domain-of-discourse, as well as recommendations for controlled vocabularies to be used.

The first set of constraints is about class obligation. Three types of class obligations are distinguished: mandatory, recommended and optional. In the following sections, the meaning of the terms MUST, MUST NOT, SHOULD and MAY are as defined in RFC 2119. In the given context, the term "processing" means that consumers must accept incoming data and transparently provide these data to applications and services.

- **Mandatory class**: a consumer of data MUST be able to process information about instances of the class; a producer of data MUST provide information about instances of the class.
- **Recommended class**: a producer of data SHOULD provide information about instances of the class; a producer of data MUST provide information about instances of the class, if such information is available; a consumer of data MUST be able to process information about instances of the class.
- **Optional class**: a consumer MUST be able to process information about instances of the class; a producer MAY provide the information but is not obliged to do so.

There is no built-in mechanism to distinguish mandatory, recommended and optional classes in OWL. A potential solution is to introduce annotation properties for mandatory, recommended and optional classes in the profile specification (potentially in ISO 19150-2).

9.3.2. Property Obligation

In the context of ontologies, **cardinality** refers to constraints on the appearance of a property on an instance of a domain class. Cardinality constraints specify restrictions on the minimum and maximum number of distinct value nodes that may appear with a given property of individuals of the class. They are used to define the **structure** of an individual of the given domain class. A property can be made optional, mandatory, conditional, recommended or appears with an exact, minimum and/or maximum number of occurrences of values.

The status of properties can be made more stringent in a given context. For instance, an optional property can be made mandatory in a particular application profile. A typical example would be a property that specifies the human language of a resource: such an element can be made mandatory in a multi-lingual community. Along the same lines, an application may make the status of an optional property conditional, or a conditional property mandatory.

As an application profile must operate within the interoperability constraints defined by the OWL standard, it cannot relax the status of data elements [7]. Violating these constraints will result in introduction of inconsistencies in logical axioms, which will lead to invalid inferences of the logical theory, making it unusable. For example, if a property is defined in an OWL ontology with a minimum cardinality of 1, it cannot be relaxed to make it optional in an application profile (minimum cardinality 0). An OWL reasoner will detect such an inconsistency.

Best Practice

To accommodate the largest number of application profiles and improve modularity ontologies should minimize the use cardinality restrictions to favor reusability of properties in application profiles. The use of rdfs:domain restriction is preferred when a property is tightly coupled to the domain class (example: geometry always applies to a spatial thing). However, not defining domain restriction will favor reusability of properties in many more application contexts (e.g. Dublin Core Properties).

While OWL provides constructs to define the range of occurrences of a property using *owl:exactCardinality*, *owl:minCardinality*, *owl:maxCardinality* restrictions to represent mandatory or optional properties, OWL does not provide mechanisms to express more nuanced cardinality constraints such as optional versus recommended properties. Many application profile documentations define cardinality constraints using the following categories based on the producer and consumer perspective (see DCAT-AP or GeoDCAT-AP profiles specifications for example):

- **Mandatory property**: a consumer MUST be able to process the information for that property; a producer MUST provide the information for that property.
- **Recommended property**: a consumer MUST be able to process the information for that property; a producer SHOULD provide the information for that property if it is available.
- **Optional property**: a consumer MUST be able to process the information for that property; a producer MAY provide the information for that property but is not obliged to do so.

The meaning of the terms MUST, MUST NOT, SHOULD and MAY are as defined in RFC 2119. In the given context, the term "processing" means that consumers must accept incoming data and transparently provide these data to applications and services. It neither implies nor prescribes what applications and services finally do with the data (parse, convert, store, make searchable, display to users, etc.). Classes are classified as **Mandatory (M)** if they appear as the range of one of the mandatory properties.

There is no built-in mechanism to distinguish mandatory, recommended and optional properties in OWL. A potential solution is to introduce annotation properties for mandatory, recommended and optional properties in the profile specification (potentially in ISO 19150-2).

9.3.3. Class definition

Named Class

Classes can be understood as sets of individuals. The addition of **named classes** (instances of rdfs:Class or owl:Class defined with a URI) should be added in an application profile in order to classify information and facilitate the search of individuals of a given class. By not adding classes in a profile, it would not be possible to find instances of a given category.

Logical class

OWL 2 provides three types of class descriptions to represent more advanced class constructors that are used in DL. They can be viewed as representing the AND, OR and NOT operators on classes. The three operators get the standard set-operator names: intersection, union and complement. These language constructs also share the characteristic that they contain nested class descriptions, either one (complement) or more (union, intersection). These classes are defined as anonymous classes and require the usage of an owl:sameAs axiom to an associated named class for a logical class definition. To leverage these classes expressions, a reasoner is required to classify these individuals according their definition. For this reason, these axioms are reserved for complex analytic applications. They should not be used in a simple RDF application profile.

9.3.4. Class Hierarchy

In modeling, a class hierarchy allows the definition of classes at different levels of generalization with properties at the appropriate levels, with specialized classes inheriting properties from general classes. A hierarchy allows the categorization of instances at the lowest level of specificity, while being able to retrieve them using more general queries.

Thus, when adding a class in an application, the superclasses definition should be added along with the axioms rdfs:subclassOf relating these classes. By adding this hierarchy, a transitive reasoner can be used to classify instances to more general concepts (superclasses). By not doing so, generalized query and property inheritance will not be possible.

9.3.5. Property hierarchy

In modeling, a property hierarchy allows the definition of properties (roles) at different levels of generalization. A hierarchy allows the categorization of properties at the lowest level of specificity, while being able to retrieve them using more general property queries.

Thus, when adding a property in an application, the super properties definition should be added along with the axioms rdfs:subPropertyOf relating these properties. By adding this hierarchy, a transitive reasoner can be used to classify properties to more general properties). By not doing so, generalized query on properties will not be possible.

Property hierarchies are not necessary for a simple RDF application profile, as the explicit definitions of properties associated with a class are enough to support instance description and validation.

9.3.6. Cardinality constraints encoding

There are two approaches that will need to be considered when encoding the cardinality constraints depending on whether cardinality restrictions are to be used for inferencing or for constraining data (validation).

OWL encoding

OWL provides *owl:exactCardinality, owl:minCardinality, owl:maxCardinality* predicates to capture cardinality restrictions. As long as the application profile operates within the interoperability constraints defined by the OWL standard and does not relax existing cardinality constraints in the ontology (e.g. changing mandatory to optional), additional cardinality-related axioms can be added in the profile if it is encoded in OWL. As described in the discussion OWL versus SHACL above and **Unique-Name Assumption**, OWL was designed for inferencing. OWL restrictions are not actually data constraints, but rather describe inferences to be applied based on them. For example, assuming there is an *owl:maxCardinality 1* restriction stating that a person can only have 1 value for *ex:hasFather* and there is an instance of *ex:Person* that has two *ex:hasFather* values, then an OWL processor will assume that these two values must in fact represent the same real-world entity, just with different URIs.

Pros:

- Can use OWL reasoner for inferencing, leveraging Unique-name assumption.

Cons:

- Axioms are defined globally in the namespace and it is hard to know which applications profile they apply.
- Extra annotations on restrictions (such as recommended, optional) are rare in OWL and harder to process.
- Duplication of axioms, cannot be merged globally
- Use of owl cardinality restriction cannot be used for data constraints/validation but it is often misused by ontologists.

SHACL Encoding

SHACL defines cardinality constraint components to specify restrictions on the minimum and maximum number of distinct value nodes. The default cardinality in SHACL for property shapes is {0,*unbounded*}. These restrictions are used to describe cardinality restrictions from a validation viewpoint, not a logical viewpoint (see previous section).

NOTE If no cardinality is declared, ShEx standard assumes the cardinality to be {1,1} while SHACL assumes {0,*}.

Table 3. SHACL cardinality constraint components

Operation	Description
sh:minCount	Restricts minimum number of value nodes. If not defined, there is no restriction (no minimum).
sh:maxCount	Restricts maximum number of value nodes. If not defined, there is no restriction (unbounded).

Pros:

- The capture of cardinality restrictions for validation purposes is intuitive for end-users and simpler to capture in SHACL.
- SHACL cardinality restrictions can be used to validate and build data capture forms.
- Annotation for property obligation (mandatory, recommended, optional) can be easily captured on SHACL property constraints.

Cons:

- Requires a separate RDF validation language outside of OWL to capture property restrictions.

Example

The following SHACL shape for *neo:Aircraft* defines a mandatory property neo:DeviceEntity.name by setting the *sh:minCount* and *sh:maxCount* to 1. Note that additional properties have been added on the property shape to overwrite the label and description of the property in the context of the application profile. An example of annotation for capturing obligation types (mandatory, recommended and optional) can also be captured.

SHACL shape defining mandatory property for neo:Aircraft

```
:AircraftShape a sh:NodeShape;
   sh:targetClass neo:Aircraft ;
   sh:property [
    sh:path     neo:DeviceEntity.name ;
    rdfs:label  "Aircraft name"
    rdfs:comment "The name of the aircraft";
    profile:obligation 'mandatory';
    sh:minCount 1;
    sh:maxCount 1;
  ] ;
```

9.3.7. Value Space Restriction

For some data elements, the value space can be made more restrictive than in the standard. This mechanism can apply when the standard is very loose about the values for a data element. The following value space constraints have been identified:

Closed Classes

In some cases, a class is composed of a finite set (a.k.a. closed set) of individuals (e.g. days of the week). OWL provides a straightforward way to describe a class by enumerating all its instances by using *owl:oneOf* axiom and *owl:equivalentClass* axiom. The usage of this axiom requires an OWL reasoner to validate the logical correctness of the assertion using individuals of these class. However, for a simple RDF application profile, **the instances of closed class could be copied in the application profile** to make them available to the end user without the need for reasoners. There is however no guarantee for a simple RDF application profile to prevent the introduction of new instances of the closed class that are outside the enumerated list of individuals.

NOTE The current version of NEO does not use *owl:oneOf* construct, but this use case is documented as it may be used in other ontologies.

Code List and Taxonomic Terms

In many cases, properties take instance values defined in code lists, taxonomies or thesauri. The standard ISO 19150-2 defines a design pattern for encoding code lists by defining subclasses of *skos:Concept* and *skos:ConceptScheme* and instances of *skos:Collection* for each code list. The actual values of the code lists or taxonomies are defined separately in a separate RDF document, so can be easily switched for different domains. This approach can be used for defining a simple RDF application profile. In this case, to favor reusability of the profile, an application profile should be composed of a bundle of RDF documents including the application profile ontology and a set of one or more RDF documents for the code lists/taxonomies.

Node Kind Constraints

Node Kind constraints specifies conditions to be satisfied by the RDF node kind of each value node. The node kind constraints lie into two categories: individual node kinds for value nodes of object properties, and literal node kinds for value nodes of datatype properties.

If an individual is not expected to be used outside a particular ontology, one can use an anonymous individual, which is identified by a local node ID rather than a global IRI. Anonymous individuals are analogous to blank nodes in RDF. Individuals expected to be used outside a particular ontology, should be identified by URIs.

WARNING	The use of blank node in RDF data is often problematic when data are accessed from remote services (such as SPARQL endpoint), as there is no guarantee that the same blank node will have the same identifier for different queries on the remote endpoint. It is highly recommended to use URI for any instances that required to be queried directly.

The class *owl:NamedIndividual* is new to OWL 2. It is used for declaring named (in contrast to anonymous) individuals (with IRI) in OWL 2 DL. However, these is no mechanism in OWL to indicate a node should be anonymous only.

SHACL provides a more flexible way to indicate the kind of node allowed in a model, by introducing *shacl:nodeKind* property. The node kind (IRI, blank node, literal or combinations of these) of all value nodes. The values of *sh:nodeKind* in a shape are one of the following six instances of the class *sh:NodeKind*: *sh:BlankNode, sh:IRI, sh:Literal sh:BlankNodeOrIRI, sh:BlankNodeOrLiteral* and *sh:IRIOrLiteral* described in Table 4 . A shape has at most one value for *sh:nodeKind.*

Table 4. Node kinds in SHACL

Nodekind	Description
sh:IRI	*Nodes must be IRIs*
sh:BlankNode	*Nodes must be Blank nodes*
sh:Literal	*Nodes must be Literals*
sh:BlankNodeOrLiteral	*Nodes must be Blank nodes or literals*
sh:BlankNodeOrIRI	*Nodes must be Blank nodes or IRIs*
sh:IRIOrLiteral	*Nodes must be IRIs or literals*

The following example states that all values of ex:knows need to be IRIs, at any subject.

SHACL shape to check that ex:knows values are IRIs

```
ex:NodeKindExampleShape
    a sh:NodeShape ;
    sh:targetObjectsOf ex:knows ;
    sh:nodeKind sh:IRI .
```

Instances with constraints

In some cases, properties of an entity need to accept only instances following certain constraints. Two categories of constraints for instances are distinguished:

- **logical constraints**: this type of constraints is addressed by OWL using class typing, composite classes, and use of restrictions.
- **syntactic constraints**: this type of constraints is mostly used for validation use case and cannot be expressed in OWL. SHACL provides extensible mechanisms such as SHACL_SPARQL extensions to express complex syntactic constraints such as URI being in a given namespace. This is out of scope for this testbed but should be investigated in future testbeds

For a simple RDF application profile, the only logical constraints allowed on individuals of the membership are to a named class. The use of logical classes (owl:unionOf, owl:intersectionOf, owl:complementOf) can only be leveraged by the use of DL reasoners, thus are reserved for complex analytical application profiles.

Literal Constraints

Datatypes, such as xsd:string or xsd:integer, and literals such as "1"^^xsd:integer, can be used to express data ranges. The simplest data ranges are datatypes. OWL provides a mechanism to define the type of a literal (number, string, date, datetime) using *rdfs:Datatype*. OWL uses XSD Schema Simple types to define the set of core datatypes supported by OWL. Recent additions to RDF 1.1 introduce the rdf:langString for literal with languages. OWL2 also introduces owl:DatatypeRestriction (modeled with facets restrictions) to define more precisely the constraints on a datatype (such as owl:minExclusive, owl:maxExclusive) and standard set-theoretic operations on data ranges using owl:DataIntersectionOf, owl:DataUnionOf, and owl:DataComplementOf data ranges; in logical languages these are usually called conjunction, disjunction, and negation, respectively. The owl:DataOneOf data range consists of exactly the specified set of literals (Figure 3).

For a simple RDF application profile, only the use of rdfs:Datatype with URI is allowed as the more advanced data range constructs in OWL 2 requires a datatype reasoner. To validate a value on the client-side, this approach requires a client to get the definition of the type in XML schema datatype encoding and to understand XML schema datatype specification and OWL 2 Data Range constructs. This implies significant overhead and complexity on the client that in practice very few clients support, as XML datatype validation is beyond the basic type definition.

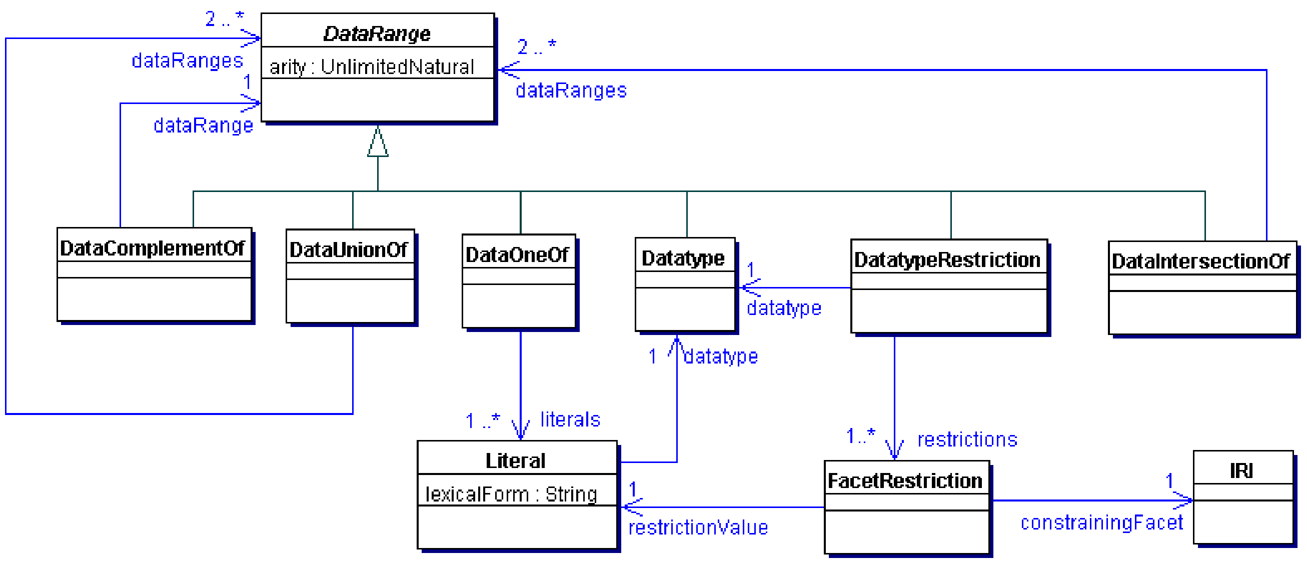


Figure 3. Data Ranges in OWL 2

9.3.8. Property Constraints

Property constraints enable the declaration of constraints on the outgoing and incoming properties of a focus node.

Domain restriction

rdfs:domain is an instance of *rdf:Property* that is used to state that any resource that has a given property is an instance of one or more classes. A domain of a property limits the individuals to which the property can be applied. The property hasChild may be stated to have the domain of Mammal. From this a reasoner can deduce that if Frank hasChild Anna, then Frank* must be a Mammal.

If a property relates an individual to another individual, and the property has a class as one of its domains, then the individual must belong to the class. Where a property P has more than one *rdfs:domain* property, then the resources denoted by subjects of triples with predicate P are instances of all the classes stated by the *rdfs:domain* properties.

rdfs:domain is allowed in simple RDF Application Schema. To accommodate the largest number of application profiles and improve modularity, ontologies should minimize the use of cardinality restrictions to favor reusability of properties in application profiles. The use of rdfs:domain restriction is preferred when a property is tightly coupled to the domain class (example: geometry always applies to a spatial thing). However, not defining domain restriction will favor reusability of properties in many more application contexts (e.g. Dublin Core Properties).

Range restriction

rdfs:range is an instance of *rdf:Property* that is used to state that the values of a property are instances of one or more classes. The range of a property limits the individuals that the property may have as its value. If a property relates an individual to another individual, and the property has a class as its range, then the other individual must belong to the range class. Where P has more than one *rdfs:range* property, then the resources denoted by the objects of triples with predicate P are instances of all the classes stated by the *rdfs:range* properties.

The use of *rdfs:range* is strongly recommended when adding properties in simple RDF application profile.

9.3.9. Property Value constraints

OWL property restrictions provide a group of properties to model value constraints: *owl:allValuesFrom*, *owl:someValuesFrom*, *owl:hasValue*. An OWL reasoner is required to perform logical consistency and validation, so these restrictions should only be reserved only for complex analytical application profiles. For a simple RDF application profile, the restriction *owl:allValuesFrom* could be replaced by *rdfs:range*.

Default Value

OWL ontologies are based on the OWA. An entity may have a value but still not know what the value is. That does not mean that the entity does not have the value. In a CWA, default values are often used when an entity does not provide a value for a property. **An OWL model does not**

support the definition of default values because of OWA, while this is a common occurrence in relational databases or in XML. SHACL and SHex provides mechanisms to assign default values for properties and complement OWL in this respect when it comes to modeling a closed world. This requirement is not about using default values as "inferred" triples at run-time (using *owl:hasValue* for example).

Example of usage of default value in SHACL for a length unit of measure

```
:LengthMeasureShape a sh:NodeShape ;
     sh:property [
        sh:path         ex:value ;
        sh:name        "value";
        sh:description "Value of the measure"
     ] ;
    sh:property [
        sh:path        ex:uom ;
        sh:name        "Unit of measure";
        sh:description "Unit of measure of a value (default meter)";
        sh:defaultValue "meter"
    ].
```

Property Characteristics

OWL2 provides property characteristics such as owl:TransitiveProperty, owl:SymmetricProperty,owl:AsymmetricProperty, owl:inverseOf, owl:FunctionalProperty, owl:ReflexiveProperty, owl:InverseFunctionalProperty, owl:IrreflexiveProperty that are extremely useful to perform advanced analytical tasks but requires an OWL reasoner to be leveraged. For this reason, these constructs are not allowed in an RDF Simple Application Profile.

Other Property constraints not supported by OWL

There are a number of property constraints that cannot be modeled in OWL that are mostly used for data validation. These constraints can be modeled using SHACL. Most of the use cases are related to interrelationships between data elements and their value spaces. For instance, the presence of one data element may impose the requirement that another element be present. Similarly, an ontology can restrict the value set of a data element, based on the value of another data element. A typical example would restrict the value space of the data type of a resource, based on its genre: for instance, a 'text' document cannot be of type MP3. Another set of use cases is the use of comparison between properties (Equals, LessThen, LessThanOrEquals, GreaterThan, GreaterThanOrEqual). For example:

- birthDate ⇐ deathDate is reasonable for most person databases
- birthDate-deathDate < 150 years is reasonable except for historic/art databases, where not-precisely known dates are often replaced with a wider interval to ensure search recall
- deathDate ⇐ today()

These types of constraints should be investigated in future testbeds.

9.3.10. Summary of profile characteristics

Table 5 summarizes which OWL axioms types are allowed for simple and complex analytic applications. The Simple Linked Data Application Profile supports axioms that do not require the use of DL reasoners. In order to allow inheritance of properties, the usage of rdfs:subclassOf axiom type in a simple RDF application profile is allowed.

Table 5. Summary of OWL Axioms types supported by simple and complex analytical applications

Axiom types	**Simple Application Profile**	**Complex Analytic Profile**
Class Axioms		
Named Class	Y	Y
owl:unionOf	N	Y
owl:intersectionOf	N	Y
owl:complementOf	N	Y
owl:oneOf	N	Y
owl:subclassOf	Y	Y
owl:equivalentClass	N	Y
owl:disjointWith	N	Y
Property Axioms		
rdfs:Property	Y	Y
owl:ObjectProperty	Y	Y
owl:DatatypeProperty	Y	Y
owl:AnnotationProperty	Y	Y
rdfs:subProperty	N	Y
owl:equivalentProperty	N	Y
owl:inverseOf	N	Y
owl:minCardinality	N	Y
owl:maxCardinality	N	Y
owl:cardinality	N	Y
rdfs:domain	Y	Y
rdfs:range	Y	Y
rdfs:sameAs	N	Y
owl:SymmetricProperty	N	Y
owl:TransitiveProperty	N	Y
owl:FunctionalProperty	N	Y
owl:InverseFunctionalProperty	N	Y
owl:ReflexiveProperty	N	Y
owl:IrreflexiveProperty	N	Y

Axiom types	**Simple Application Profile**	**Complex Analytic Profile**
owl:allValuesFrom	N	Y
owl:someValuesFrom	N	Y
owl:hasValue	N	Y
Individual Axioms		
Named Individual	Y	Y
owl:differentFrom	N	Y
owl:sameAs	N	Y
owl:AllDifferent	N	Y
Datatype Axioms		
primitive datatype	Y	Y
Enumerated datatype (owl:DataRange)	N	Y

Chapter 10. Application Profile Metadata description

This section attempts to define a formal model to describe metadata about application profiles. While the main focus of the model is to describe application profiles for linked-data-based applications. the model should be applicable also for application profiles using other schema languages (such as XML schema and JSON schema for example). The names of the classes of the model are chosen in such a way that it should remain applicable to different application profiles based on different schema languages. While this is out of scope for the testbed, a formal model was proposed for capturing application profile metadata to manage application profiles in vocabulary management system. This could be an area of investigation in future testbeds.

Of relevance to the design of this model, is the work done during OGC Testbed-12 [8] to model a semantic mediation service. The model introduced two main concepts:

- **Schema** defines the structure and constraints on a conceptual model. Schemas are modeled as a specialization of dcat:Dataset [9]. A schema can have zero or more distributions that define the encoding of the schema using different representation techniques (adms:representationTechnique) such as SHACL, ShEx, XML Schema or JSON Schema.
- **SchemaMapping** defines the transformation from one source schema to a target schema (using XSLT, Script, SPARQL rules, or other mapping languages). SchemaMappings are also modeled as specialization of dcat:Dataset. A schema mapping can have zero or more distributions that define the encoding of the schema mapping using different representation techniques (adms:representationTechnique) [10] such as XSLT or other schema mapping languages.

In our model, the concept of **Vocabulary** (which is a subclass of dcat:Dataset) is introduced to represent the concepts, properties, associations and axioms of a conceptual model. In our case, a vocabulary is associated with a namespace and uses a schema language such as RDF Schema, OWL, XML Schema or JSON Schema.

One or more subsets of vocabularies can be used to define an application profile. The concept of **Profile** is defined as a subclass of Dublin Core **dct:Standard**. A dataset used by a given application conforms (**dct:conformsTo**) to an application standard. A vocabulary can be used by multiple application profiles.

The Application Profile is typically associated with a Schema which defines the encoding of the structure and value constraints on the vocabulary terms.

The following figure illustrates the conceptual model for representing application profile metadata.

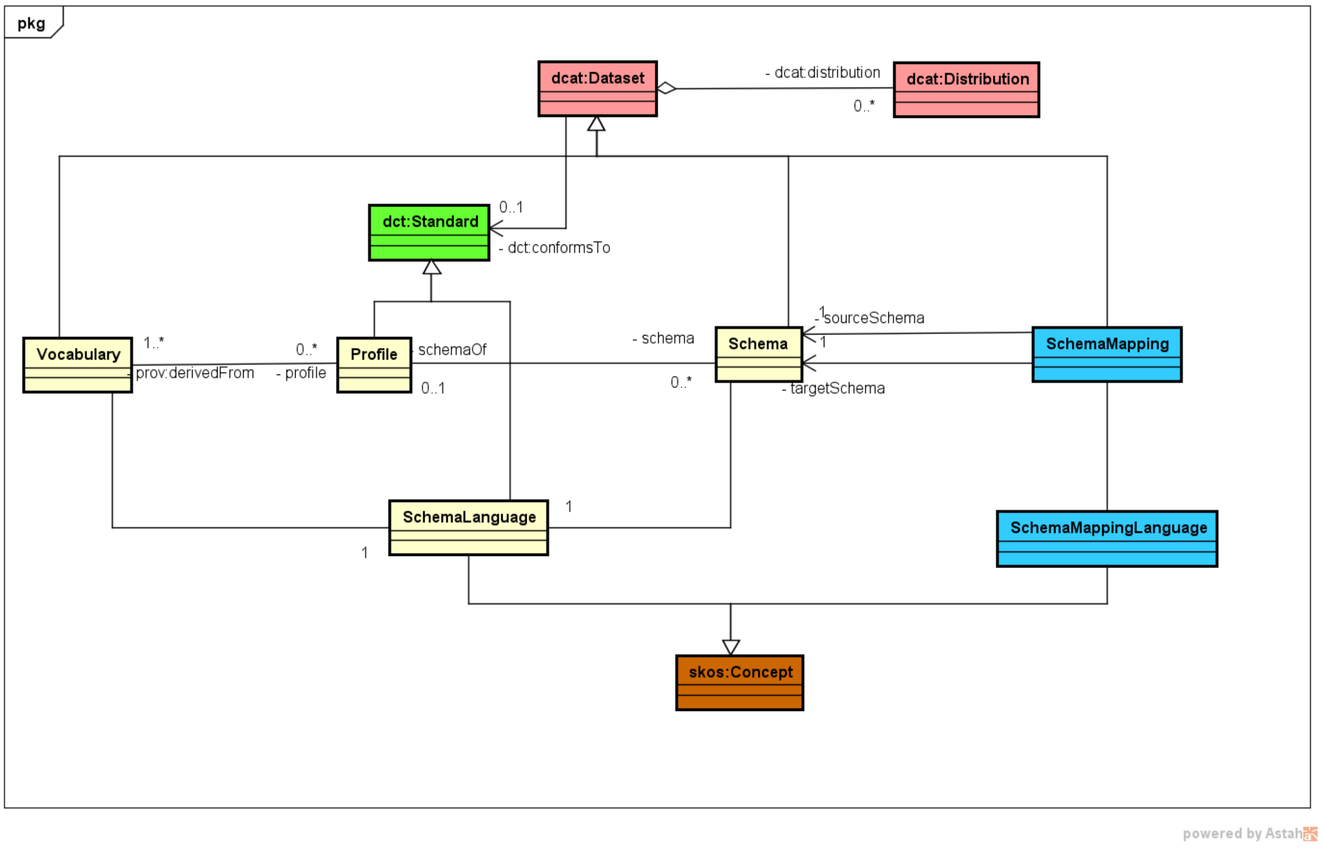


Figure 4. Application Profile Metadata Conceptual Model

At the time of this testbed, there is an ongoing activity in the W3C Dataset Exchange Working Group (DXWG). This working group is in charge of updating the DCAT ontology and defining best practices on how to define application profiles based on this ontology. There is also an attempt to define a conceptual model to represent metadata about application profiles, but it is still at a very early stage. It would be desirable to share the outcome of this testbed with the W3C working group in order to get an emerging standard to enable management, search, and discovery application profiles and validation of data against application profiles.

Appendix A: NEO Application Profiles and Sample Data

This section describes the process and artifacts used to generate the application profiles and sample data. The domain of the application profile was about Civilian Aerodrome Facilities. The reason for choosing this domain was because there were some overlaps with the Aviation Thread during the Testbed 14.

A.1. Processing Workflow description

An export of the NAS Model in ShapeChange-specific XML format (SXCML) was provided by the sponsor. The model contains all the information of the NAS UML model and OCL constraints defined in Enterprise Architect.

Figure 5 illustrates the workflow to process profile using the Profile Management Tool (PMT) and ShapeChange.

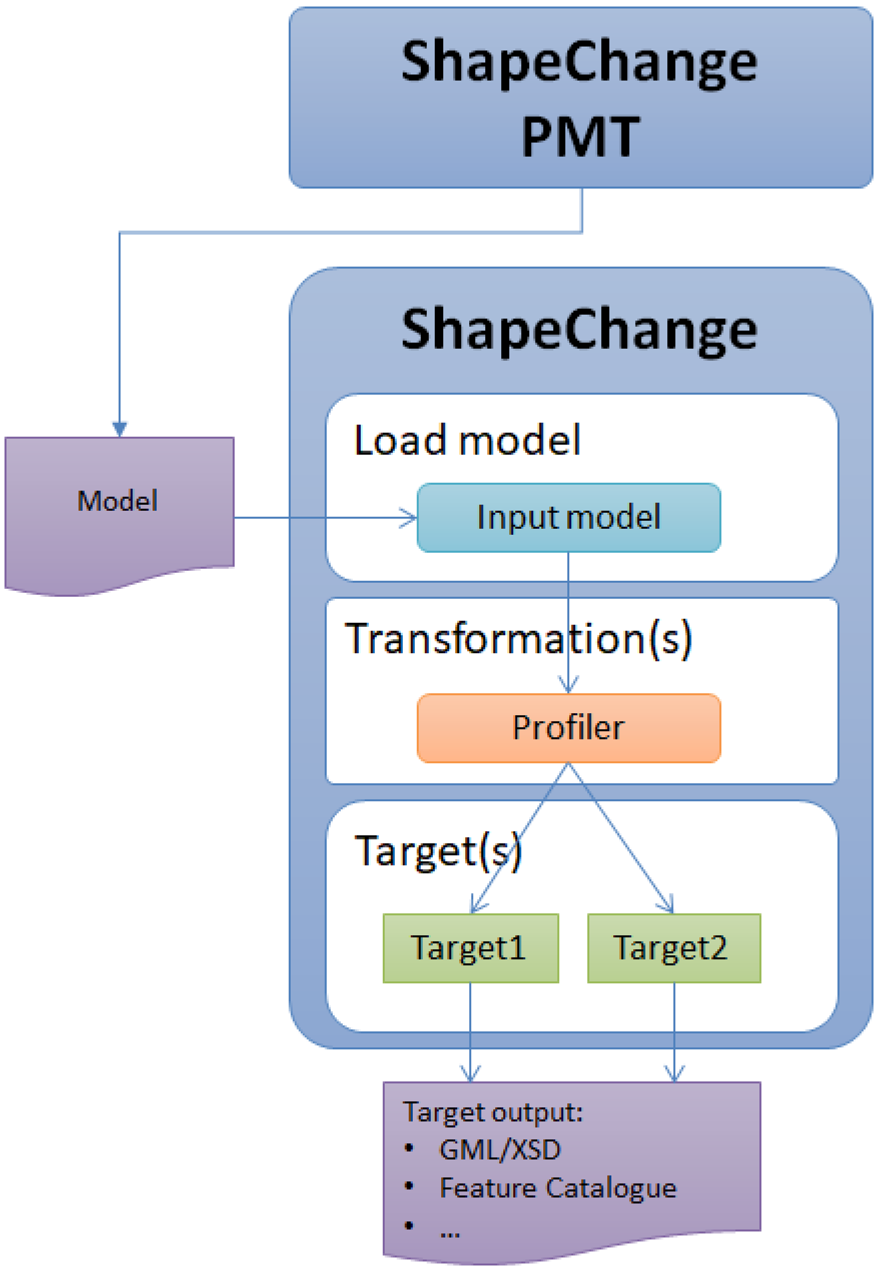
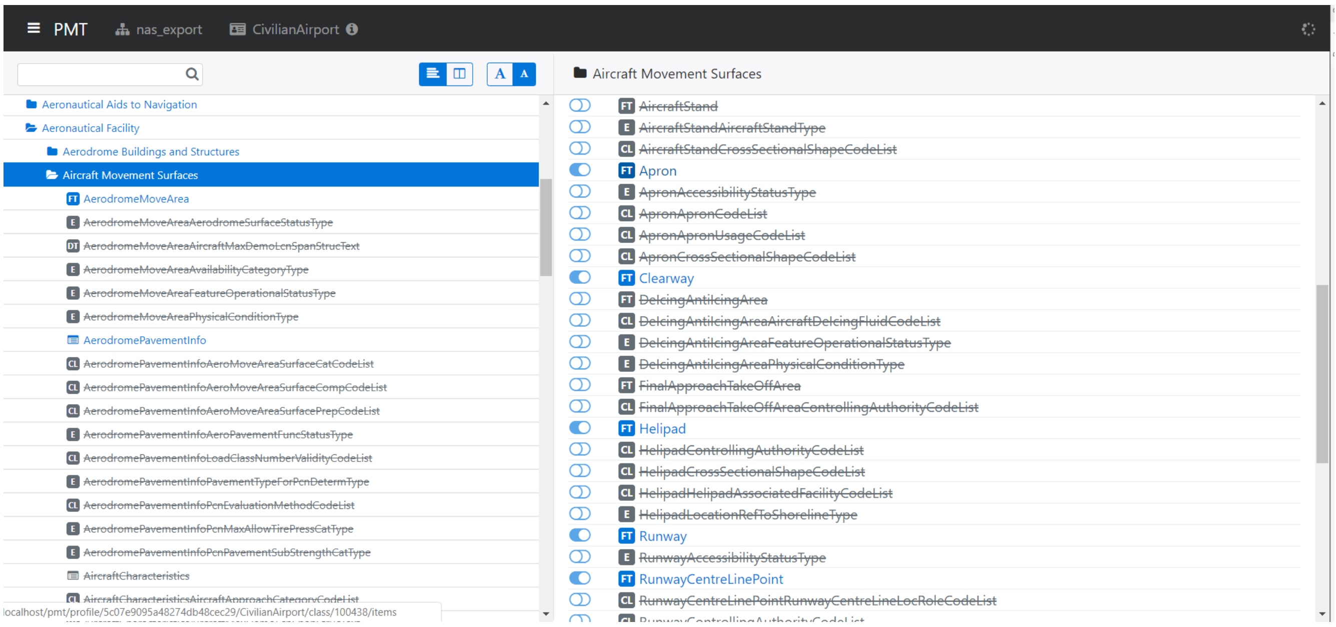


Figure 5. ShapeChange workflow - process profile created using the PMT

A.1.1. ShapeChange Profile Management Tool

The first step of the process was to the import the NAS export into the Profile Management Tool (PMT). PMT is a web application to edit application schema profiles. The definition of a consistent profile for a large application schema like the NAS is a non-trivial task. The PMT provides a number of features to support the user in this task. The PMT can load UML models from files in State Chart XML (SCXML) format. Profiles can be added to the model, edited, copied, and deleted. The tool provides a web client application to navigate through the model. A text-based search can be used to look up specific model elements. Packages, classes, and their properties can be added to or removed from a profile. The visual display of a model element changes accordingly. For selected packages and classes, the PMT offers actions for mass processing contained model elements (e.g. adding all properties of a class to the profile). In the case of Testbed-14, the participants manually selected classes related to Civilian Aerodrome facilities from Aerodrome Building and Structures, Aircraft Movement Surfgaces and General Aeronautical Ground Features packages (see Figure 6 and Figure 7).


PMT nas_export CivilianAirport

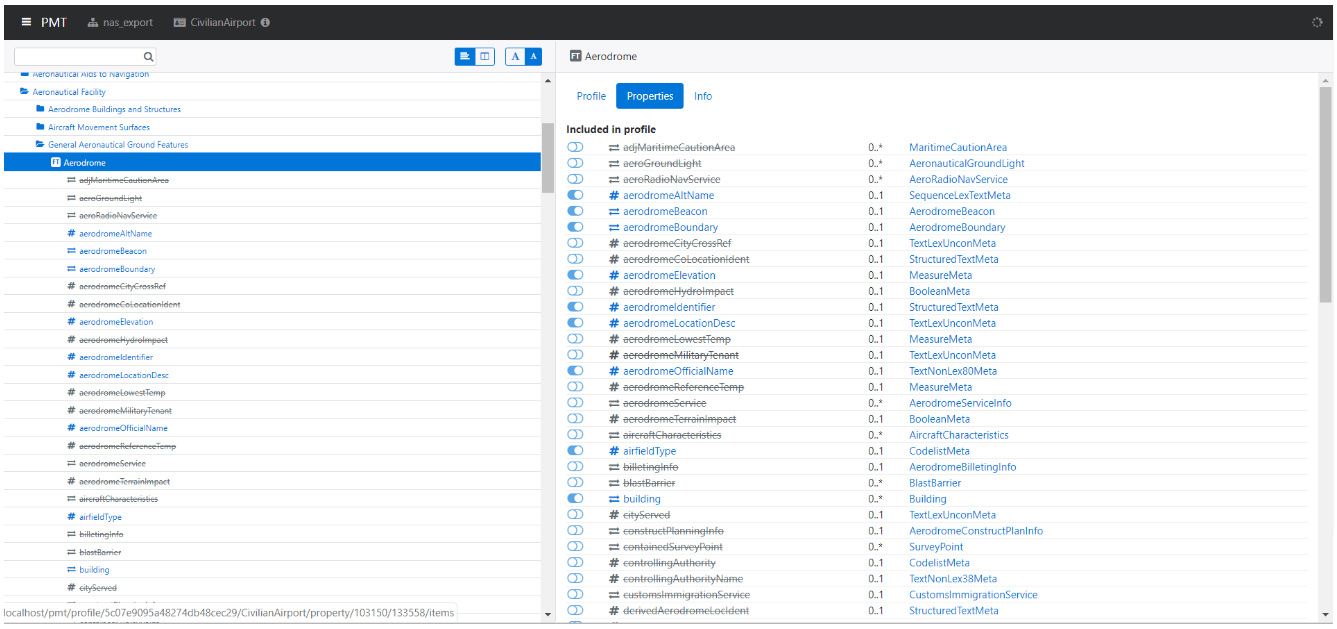
- Aeronautical Aids to Navigation
- Aeronautical Facility
  - Aerodrome Buildings and Structures
  - Aircraft Movement Surfaces
    - FT AerodromeMoveArea
    - E ~~AerodromeMoveAreaAerodromeSurfaceStatusType~~
    - DT ~~AerodromeMoveAreaAircraftMaxDemoLcnSpanStrucText~~
    - E ~~AerodromeMoveAreaAvailabilityCategoryType~~
    - E ~~AerodromeMoveAreaFeatureOperationalStatusType~~
    - E ~~AerodromeMoveAreaPhysicalConditionType~~
    - AerodromePavementInfo
    - CL ~~AerodromePavementInfoAeroMoveAreaSurfaceCatCodeList~~
    - CL ~~AerodromePavementInfoAeroMoveAreaSurfaceCompCodeList~~
    - CL ~~AerodromePavementInfoAeroMoveAreaSurfacePrepCodeList~~
    - E ~~AerodromePavementInfoAeroPavementFuncStatusType~~
    - CL ~~AerodromePavementInfoLoadClassNumberValidityCodeList~~
    - E ~~AerodromePavementInfoPavementTypeForPcnDetermType~~
    - CL ~~AerodromePavementInfoPcnEvaluationMethodCodeList~~
    - E ~~AerodromePavementInfoPcnMaxAllowTirePressCatType~~
    - E ~~AerodromePavementInfoPcnPavementSubStrengthCatType~~
    - ~~AircraftCharacteristics~~
    - CL ~~AircraftCharacteristicsAircraftApproachCategoryCodeList~~

localhost/pmt/profile/5c07e9095a48274db48cec29/CivilianAirport/class/100438/items

Aircraft Movement Surfaces

- FT ~~AircraftStand~~
- E ~~AircraftStandAircraftStandType~~
- CL ~~AircraftStandCrossSectionalShapeCodeList~~
- FT Apron
- E ~~ApronAccessibilityStatusType~~
- CL ~~ApronApronCodeList~~
- CL ~~ApronApronUsageCodeList~~
- CL ~~ApronCrossSectionalShapeCodeList~~
- FT Clearway
- FT ~~DeIcingAntiIcingArea~~
- CL ~~DeIcingAntiIcingAreaAircraftDeIcingFluidCodeList~~
- E ~~DeIcingAntiIcingAreaFeatureOperationalStatusType~~
- E ~~DeIcingAntiIcingAreaPhysicalConditionType~~
- FT ~~FinalApproachTakeOffArea~~
- CL ~~FinalApproachTakeOffAreaControllingAuthorityCodeList~~
- FT Helipad
- CL ~~HelipadControllingAuthorityCodeList~~
- CL ~~HelipadCrossSectionalShapeCodeList~~
- CL ~~HelipadHelipadAssociatedFacilityCodeList~~
- E ~~HelipadLocationRefToShorelineType~~
- FT Runway
- E ~~RunwayAccessibilityStatusType~~
- FT RunwayCentreLinePoint
- CL ~~RunwayCentreLinePointRunwayCentreLineLocRoleCodeList~~

Figure 6. Selecting classes for application profile using PMT


PMT nas_export CivilianAirport

- Aeronautical Aids to Navigation
- Aeronautical Facility
  - Aerodrome Buildings and Structures
  - Aircraft Movement Surfaces
  - General Aeronautical Ground Features
    - FT Aerodrome
      - ⇄ ~~adjMaritimeCautionArea~~
      - ⇄ ~~aeroGroundLight~~
      - ⇄ ~~aeroRadioNavService~~
      - # aerodromeAltName
      - ⇄ aerodromeBeacon
      - ⇄ aerodromeBoundary
      - # ~~aerodromeCityCrossRef~~
      - # ~~aerodromeCoLocationIdent~~
      - # aerodromeElevation
      - # ~~aerodromeHydroImpact~~
      - # aerodromeIdentifier
      - # aerodromeLocationDesc
      - # ~~aerodromeLowestTemp~~
      - # ~~aerodromeMilitaryTenant~~
      - # aerodromeOfficialName
      - # ~~aerodromeReferenceTemp~~
      - ⇄ ~~aerodromeService~~
      - # ~~aerodromeTerrainImpact~~
      - ⇄ ~~aircraftCharacteristics~~
      - # airfieldType
      - ⇄ ~~billetingInfo~~
      - ⇄ ~~blastBarrier~~
      - ⇄ building
      - # ~~cityServed~~

localhost/pmt/profile/5c07e9095a48274db48cec29/CivilianAirport/property/103150/133558/items

FT Aerodrome

Profile | Properties | Info

Included in profile

⇄ ~~adjMaritimeCautionArea~~	0..*	MaritimeCautionArea
⇄ ~~aeroGroundLight~~	0..*	AeronauticalGroundLight
⇄ ~~aeroRadioNavService~~	0..*	AeroRadioNavService
# aerodromeAltName	0..1	SequenceLexTextMeta
⇄ aerodromeBeacon	0..1	AerodromeBeacon
⇄ aerodromeBoundary	0..1	AerodromeBoundary
# ~~aerodromeCityCrossRef~~	0..1	TextLexUnconMeta
# ~~aerodromeCoLocationIdent~~	0..1	StructuredTextMeta
# aerodromeElevation	0..1	MeasureMeta
# ~~aerodromeHydroImpact~~	0..1	BooleanMeta
# aerodromeIdentifier	0..1	StructuredTextMeta
# aerodromeLocationDesc	0..1	TextLexUnconMeta
# ~~aerodromeLowestTemp~~	0..1	MeasureMeta
# ~~aerodromeMilitaryTenant~~	0..1	TextLexUnconMeta
# aerodromeOfficialName	0..1	TextNonLex80Meta
# ~~aerodromeReferenceTemp~~	0..1	MeasureMeta
⇄ ~~aerodromeService~~	0..*	AerodromeServiceInfo
# ~~aerodromeTerrainImpact~~	0..1	BooleanMeta
⇄ ~~aircraftCharacteristics~~	0..*	AircraftCharacteristics
# airfieldType	0..1	CodelistMeta
⇄ ~~billetingInfo~~	0..1	AerodromeBilletingInfo
⇄ ~~blastBarrier~~	0..*	BlastBarrier
⇄ building	0..*	Building
# ~~cityServed~~	0..1	TextLexUnconMeta
⇄ ~~constructPlanningInfo~~	0..1	AerodromeConstructPlanInfo
⇄ ~~containedSurveyPoint~~	0..*	SurveyPoint
# ~~controllingAuthority~~	0..1	CodelistMeta
# ~~controllingAuthorityName~~	0..1	TextNonLex38Meta
⇄ ~~customsImmigrationService~~	0..1	CustomsImmigrationService
# ~~derivedAerodromeLocIdent~~	0..1	StructuredTextMeta

Figure 7. Selecting properties from class using the PMT

Once the classes and properties were selected, an export of the application profile was done in SCXML format to be consumed by ShapeChange.

A.1.2. Configuring shape change

The next step was to build two configuration files for the Simple Linked Data Application and Complex Analytic Profile, so they can be used by the ShapeChange tool to produce the RDF application profiles needed for this Testbed.

Application Profile Configuration for Simple Linked Data Application

To perform the generation of the application profile for the Simple Linked Data Application, the configuration used to generated the NEO ontology was used as a starting point and was modified to handle the profile model by adding the Profiler Transformer and removing the mapping rules that were not compatible with the Simple Application Profile (commented in the configuration file source code). The target output was set to produce an OWL model using the RDF/XML encoding. The following is the configuration used for generating the application profile for Simple Linked Data Application.

simpleCivilianAirport.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ShapeChangeConfiguration
  xmlns="http://www.interactive-instruments.de/ShapeChange/Configuration/1.1"
  xmlns:sc="http://www.interactive-instruments.de/ShapeChange/Configuration/1.1"
  xmlns:xi="http://www.w3.org/2001/XInclude" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.interactive-
instruments.de/ShapeChange/Configuration/1.1
https://shapechange.net/resources/schema/ShapeChangeConfiguration.xsd">
  <input id="INPUT">
    <parameter name="addTaggedValues"
      value=
"ontologyName,ontologyDefinition,ontologyDescription,ontologyResourceURI,name,definiti
on,description,primaryCode,aliasList,mdbPK,skosConceptSchemeSubclassName,broaderListed
Value"/>

    <!--<parameter name="inputModelType" value="EA7"/>
   <parameter name="inputFile" value="C:/NAS/v2/TEST_v8.x.eap"/>-->
   <parameter name="inputModelType" value="SCXML"/>
   <parameter name="inputFile" value="C:/testbed14/nas_export.xml"/>
    <parameter name="appSchemaName" value="NSG Application Schema" />
    <!--
    <parameter name="taggedValueImplementation" value="array" />
    <parameter name="useStringInterning" value="true" />
    -->
    <parameter name="publicOnly" value="true"/>
    <!--<parameter name="checkingConstraints" value="enabled"/>-->
   <parameter name="checkingConstraints" value="disabled"/>
    <parameter name="sortedSchemaOutput" value="true"/>
    <parameter name="dontConstructAssociationNames" value="true"/>
    <!-- <xi:include href="src/main/resources/config/StandardAliases.xml"/> -->
    <descriptorSources>
      <DescriptorSource descriptor="documentation" source="tag" tag="documentation"/>
      <DescriptorSource descriptor="alias" source="none"/>
      <DescriptorSource descriptor="definition" source="tag" tag="definition"/>
      <DescriptorSource descriptor="description" source="tag" tag="description"/>
      <DescriptorSource descriptor="example" source="none"/>
      <DescriptorSource descriptor="legalBasis" source="none"/>
      <DescriptorSource descriptor="dataCaptureStatement" source="none"/>
      <DescriptorSource descriptor="primaryCode" source="tag" tag="primaryCode"/>
      <DescriptorSource descriptor="language" source="none"/>
    </descriptorSources>
  </input>

  <log>
```

```
    <parameter name="reportLevel" value="INFO"/>
   <parameter name="logFile" value="C:/testbed14/results/log.xml"/>
  </log>

  <transformers>
    <Transformer id="TRF1" input="INPUT" class=
"de.interactive_instruments.ShapeChange.Transformation.Profiling.Profiler" mode=
"enabled">
      <parameters>
        <ProcessParameter name="profiles" value="CivilianAirport"/>
      </parameters>
      <rules>
        <ProcessRuleSet name="profiler">
          <rule name="rule-trf-profiling-preprocessing-profilesValueConsistencyCheck
"/>
          <rule name="rule-trf-profiling-preprocessing-modelConsistencyCheck"/>
          <rule name="rule-trf-profiling-processing-explicitProfileSettings"/>
          <rule name="rule-trf-profiling-processing-classRemovalIncludesAllSubtypes"/>
        </ProcessRuleSet>
      </rules>
   </Transformer>

    <Transformer id="TRF2" input="TRF1" class=
"de.interactive_instruments.ShapeChange.Transformation.Flattening.AssociationClassMapp
er" mode="enabled">
      <rules>
        <ProcessRuleSet name="mapper">
          <!-- disable constraint validation, else the "comment" (nas:description) of
the OclConstraint is reset to what the SC parser finds in the constraint text
(nas:definition = OCL expression) -->
          <rule name="rule-trf-all-postprocess-skip-constraint-validation"/>
        </ProcessRuleSet>
      </rules>
    </Transformer>
    <Transformer id="TRF3" input="TRF2" class=
"de.interactive_instruments.ShapeChange.Transformation.Flattening.Flattener" mode=
"enabled">
      <parameters>
        <ProcessParameter name="includeUnionIdentifierTaggedValue" value="true"/>
        <ProcessParameter name="replaceUnionExcludeRegex" value=
"MaritimeBottomCharacter(MaterialCode|MaterialQualCode|SedimentColourCode)Reason"/>
      </parameters>
      <rules>
        <ProcessRuleSet name="flattener">
          <rule name="rule-trf-cls-replace-with-union-properties"/>
          <!-- OCL expressions that rely on property value(s)OrReason will be invalid
- to keep the log clean we've disabled constraint validation -->
          <rule name="rule-trf-all-postprocess-skip-constraint-validation"/>
        </ProcessRuleSet>
      </rules>
    </Transformer>
  </transformers>

  <targets>

    <TargetOwl inputs="TRF3" class=
"de.interactive_instruments.ShapeChange.Target.Ontology.OWLISO19150" mode="enabled">
     <targetParameter name="outputDirectory" value="C:/testbed14/results/ontology"/>
      <!-- Output format identifiers: turtle ntriples nquads rdfxml jsonld rdfjson
trig trix rdfthrift -->
      <targetParameter name="outputFormat" value="rdfxml"/>
      <targetParameter name="defaultEncodingRule" value="NEO"/>
      <targetParameter name="language" value="en"/>
      <targetParameter name="defaultTypeImplementation" value="owl:Class"/>

      <targetParameter name="ontologyNameCode" value="neo"/>
      <targetParameter name="source" value="NSG Enterprise Ontology (NEO) Standard
8.0"/>
      <targetParameter name="URIbase" value="http://api.nsgreg.nga.mil/ontology"/>

      <targetParameter name="skosConceptSchemeSuffix" value="_ConceptScheme"/>
      <targetParameter name="codeNamespaceForEnumerations"  value=
"http://api.nsgreg.nga.mil/ontology/neo-enum/8.0"/>
      <targetParameter name="codeListOwlClassNamespaceForEnumerations"  value=
"http://api.nsgreg.nga.mil/ontology/neo-enum/8.0"/>

      <!-- over-ride default "#" -->
      <!-- <targetParameter name="rdfNamespaceSeparator" value="/"/> -->

      <descriptorTargets>
        <DescriptorTarget appliesTo="ontology" target="rdfs:label" template=
"[[TV:ontologyName]]" format="langString"/>
        <DescriptorTarget appliesTo="ontology" target="skos:definition" template=
"![CDATA[Definition: [[TV:ontologyDefinition]]  Description:
[[TV:ontologyDescription]]]]" noValueText="[None Specified]" format="langString"/>
        <DescriptorTarget appliesTo="ontology" target="rdfs:isDefinedBy" template=
"[[TV:ontologyResourceURI]]" format="IRI"/>
        <DescriptorTarget appliesTo="ontology" target="skos:prefLabel" template=
"[[TV:ontologyName]]" format="langString"/>
        <DescriptorTarget appliesTo="ontology" target="skos:altLabel" template=
"[[TV(|):aliasList]]" multiValueBehavior="splitToMultipleTargets" format="langString
"/>

        <DescriptorTarget appliesTo="class" target="rdfs:label" template=
"[[TV:primaryCode]]" format="langString"/>
        <DescriptorTarget appliesTo="class" target="skos:definition" template=
"![CDATA[Definition: [[TV:definition]]  Description: [[TV:description]]]]"
noValueText="[None Specified]" format="langString"/>
        <DescriptorTarget appliesTo="class" target="rdfs:isDefinedBy" template=
"http://nsgreg.nga.mil/as/view?i=[[TV:mdbPK]]" format="IRI"/>
        <DescriptorTarget appliesTo="class" target="skos:prefLabel" template=
"[[TV:name]]" format="langString"/>
```

```
        <DescriptorTarget appliesTo="class" target="skos:altLabel" template=
"[[TV(|):aliasList]]" multiValueBehavior="splitToMultipleTargets" format="langString
"/>

        <DescriptorTarget appliesTo="property" target="rdfs:label" template=
"[[TV:primaryCode]]" format="langString"/>
        <DescriptorTarget appliesTo="property" target="skos:definition" template=
"![CDATA[Definition: [[TV:definition]]  Description: [[TV:description]]]]"
noValueText="[None Specified]" format="langString"/>
        <DescriptorTarget appliesTo="property" target="rdfs:isDefinedBy" template=
"http://nsgreg.nga.mil/as/view?i=[[TV:mdbPK]]" format="IRI"/>
        <DescriptorTarget appliesTo="property" target="skos:prefLabel" template=
"[[TV:name]]" format="langString"/>
        <DescriptorTarget appliesTo="property" target="skos:altLabel" template=
"[[TV(|):aliasList]]" multiValueBehavior="splitToMultipleTargets" format="langString
"/>

        <DescriptorTarget appliesTo="conceptscheme" target="rdfs:label" template=
"[[TV:primaryCode]]_ConceptScheme" format="langString"/>
        <DescriptorTarget appliesTo="conceptscheme" target="skos:definition" template
="![CDATA[Definition: [[TV:definition]]  Description: [[TV:description]]]]"
noValueText="[None Specified]" format="langString"/>
        <DescriptorTarget appliesTo="conceptscheme" target="rdfs:isDefinedBy"
template="http://nsgreg.nga.mil/as/view?i=[[TV:mdbPK]]" format="IRI"/>
        <DescriptorTarget appliesTo="conceptscheme" target="skos:prefLabel" template=
"[[TV:name]] - Concept Scheme" format="langString"/>
      </descriptorTargets>

     <xi:include href="C:/testbed14/config/StandardMapEntries-owl.xml"/>
      <rdfMapEntries>
        <!-- <RdfTypeMapEntry type="Binary" target="owl:Class"/> -->

        <RdfTypeMapEntry type="SecurityAttributesGroupType" target="ntk:RequiresType
"/>
        <RdfTypeMapEntry type="ISM_Notice" target="icism:NoticeType"/>
        <RdfTypeMapEntry type="NTKAccess" target="ntk:RequiresType"/>
        <RdfTypeMapEntry type="RevisionRecall" target="rr:RevisionRecallType"/>
<!--        <RdfTypeMapEntry type="VoidValueReason" target="test:VoidValueReason"/>-->
      </rdfMapEntries>

      <rdfConversionParameters>
        <!--        <StereotypeConversionParameter wellknown="FeatureType"
subClassOf="geo:Feature"/>  -->
      </rdfConversionParameters>

      <constraintMappings>
     <!--   <ConstraintMapping constraintType="OCL" template="[[name]]: [[comment]]
OCL expression: [[text]]" noValue="[None Specified]" format="langString"/> -->
      </constraintMappings>

     <xi:include href="C:/testbed14/config/StandardNamespaces-owl.xml"/>
```

```
      <namespaces>
<!--        <Namespace nsabr="test" ns="https://my.test/enums#"/>-->
        <Namespace nsabr="icism" ns=
"https://www.dni.gov/files/documents/CIO/ICEA/Juliet/ISM-Public#" location=
"https://www.dni.gov/files/documents/CIO/ICEA/Juliet/ISM-Public"/>
        <Namespace nsabr="ntk" ns=
"https://www.dni.gov/files/documents/CIO/ICEA/India/NTK-V10-Public#" location=
"https://www.dni.gov/files/documents/CIO/ICEA/India/NTK-V10-Public" />
        <Namespace nsabr="rr" ns=
"https://www.dni.gov/files/documents/CIO/ICEA/Juliet/RevRecall-Public#" location=
"https://www.dni.gov/files/documents/CIO/ICEA/Juliet/RevRecall-Public" />
      </namespaces>

      <rules>
        <EncodingRule name="NEO" extends="*">
          <rule name="rule-owl-pkg-singleOntologyPerSchema"/>
          <rule name="rule-owl-pkg-ontologyName-code"/>
          <rule name="rule-owl-pkg-ontologyName-appendVersion"/>
          <rule name="rule-owl-pkg-versionInfo"/>
          <rule name="rule-owl-pkg-versionIRI"/>
          <rule name="rule-owl-pkg-versionIRI-avoid-duplicate-version"/>
          <rule name="rule-owl-pkg-dctSourceTitle"/>
          <rule name="rule-owl-cls-iso191502IsAbstract"/>
          <rule name="rule-owl-cls-generalization"/>
      <!--    <rule name="rule-owl-cls-disjoint-classes"/> -->

          <rule name="rule-owl-cls-encode-featuretypes"/>
          <rule name="rule-owl-cls-encode-objecttypes"/>
          <rule name="rule-owl-cls-encode-mixintypes"/>
          <rule name="rule-owl-cls-encode-datatypes"/>
          <rule name="rule-owl-cls-encode-basictypes"/>

          <rule name="rule-owl-prop-general"/>
          <rule name="rule-owl-prop-range-global"/>
          <!-- <rule name="rule-owl-prop-range-local-withUniversalQuantification"/>
-->
          <rule name="rule-owl-prop-localScopeAll"/>
          <rule name="rule-owl-prop-multiplicityAsUnqualifiedCardinalityRestriction"/>
          <!-- <rule name="rule-owl-prop-
multiplicityAsQualifiedCardinalityRestriction"/> -->
          <rule name="rule-owl-prop-iso191502AssociationName"/>
      <!--    <rule name="rule-owl-prop-inverseOf"/> -->
          <rule name="rule-owl-prop-iso191502Aggregation"/>
          <!--
          <rule name="rule-owl-all-constraints-humanReadableTextOnly"/>
          <rule name="rule-owl-all-constraints-byConstraintMapping"/>
          <rule name="rule-owl-cls-union"/>
          <rule name="rule-owl-cls-unionSets"/>
          <rule name="rule-owl-cls-iso191502Enumeration"/>
          <rule name="rule-owl-cls-enumerationAsCodelist"/> -->
          <rule name="rule-owl-cls-codelist-external"/>
```

```
       <!--    <rule name="rule-owl-cls-codelist-19150-2"/> -->
          <rule name="rule-owl-cls-codelist-19150-2-skos-collection"/>
       <!--   <rule name="rule-owl-cls-codelist-19150-2-objectOneOfForEnumeration"/>
-->
      <!--    <rule name="rule-owl-cls-codelist-19150-2-differentIndividuals"/> -->
          <rule name="rule-owl-cls-codelist-19150-2-owlClassInDifferentNamespace"/>
          <!--  <rule name="rule-owl-cls-codelist-19150-2-conceptSchemeSubclass"/> -->
          <rule name="rule-owl-prop-code-broader-byBroaderListedValue"/>
        </EncodingRule>
      </rules>
    </TargetOwl>
   <!--<Target
class="de.interactive_instruments.ShapeChange.Target.ModelExport.ModelExport"
    mode="enabled" inputs="INPUT">
    <targetParameter name="outputDirectory" value="C:/NAS/v2/results/modelExport"/>
    <targetParameter name="outputFilename" value="nas_export"/>
    <targetParameter name="sortedOutput" value="true"/>
   </Target>-->
  </targets>
</ShapeChangeConfiguration>
```

Application Profile Configuration for Complex Analytic Application

The following configuration was used to define an RDF Application Profile for a Complex Analytic Application. The configuration file was adapted to consume the PMT exported model by adding the Profiler Transformer. All the mapping rules defined in the original NEO mapping were kept (except the generation of OCL constraints as text) as defined in the standard ISO 19150-2. The following is the configuration used for generating the application profile for Complex Analytic Application.

complexCivilianAirport.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ShapeChangeConfiguration
  xmlns="http://www.interactive-instruments.de/ShapeChange/Configuration/1.1"
  xmlns:sc="http://www.interactive-instruments.de/ShapeChange/Configuration/1.1"
  xmlns:xi="http://www.w3.org/2001/XInclude" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.interactive-
instruments.de/ShapeChange/Configuration/1.1
https://shapechange.net/resources/schema/ShapeChangeConfiguration.xsd">
  <input id="INPUT">
    <parameter name="addTaggedValues"
      value=
"ontologyName,ontologyDefinition,ontologyDescription,ontologyResourceURI,name,definiti
on,description,primaryCode,aliasList,mdbPK,skosConceptSchemeSubclassName,broaderListed
Value"/>

    <!--<parameter name="inputModelType" value="EA7"/>
   <parameter name="inputFile" value="C:/NAS/v2/TEST_v8.x.eap"/>-->
   <parameter name="inputModelType" value="SCXML"/>
```

```
    <parameter name="inputFile" value="C:/testbed14/nas_export.xml"/>
     <parameter name="appSchemaName" value="NSG Application Schema" />
     <!--
     <parameter name="taggedValueImplementation" value="array" />
     <parameter name="useStringInterning" value="true" />
     -->
     <parameter name="publicOnly" value="true"/>
     <!--<parameter name="checkingConstraints" value="enabled"/>-->
    <parameter name="checkingConstraints" value="disabled"/>
     <parameter name="sortedSchemaOutput" value="true"/>
     <parameter name="dontConstructAssociationNames" value="true"/>
     <!-- <xi:include href="src/main/resources/config/StandardAliases.xml"/> -->
     <descriptorSources>
       <DescriptorSource descriptor="documentation" source="tag" tag="documentation"/>
       <DescriptorSource descriptor="alias" source="none"/>
       <DescriptorSource descriptor="definition" source="tag" tag="definition"/>
       <DescriptorSource descriptor="description" source="tag" tag="description"/>
       <DescriptorSource descriptor="example" source="none"/>
       <DescriptorSource descriptor="legalBasis" source="none"/>
       <DescriptorSource descriptor="dataCaptureStatement" source="none"/>
       <DescriptorSource descriptor="primaryCode" source="tag" tag="primaryCode"/>
       <DescriptorSource descriptor="language" source="none"/>
     </descriptorSources>
   </input>

   <log>
     <parameter name="reportLevel" value="INFO"/>
    <parameter name="logFile" value="C:/testbed14/results/log.xml"/>
   </log>

   <transformers>
     <Transformer id="TRF1" input="INPUT" class=
 "de.interactive_instruments.ShapeChange.Transformation.Profiling.Profiler" mode=
 "enabled">
       <parameters>
         <ProcessParameter name="profiles" value="CivilianAirport"/>
       </parameters>
       <rules>
         <ProcessRuleSet name="profiler">
           <rule name="rule-trf-profiling-preprocessing-profilesValueConsistencyCheck
 "/>
           <rule name="rule-trf-profiling-preprocessing-modelConsistencyCheck"/>
           <rule name="rule-trf-profiling-processing-explicitProfileSettings"/>
           <rule name="rule-trf-profiling-processing-classRemovalIncludesAllSubtypes"/>
         </ProcessRuleSet>
       </rules>
    </Transformer>

     <Transformer id="TRF2" input="TRF1" class=
 "de.interactive_instruments.ShapeChange.Transformation.Flattening.AssociationClassMapp
 er" mode="enabled">
```

```
      <rules>
        <ProcessRuleSet name="mapper">
          <!-- disable constraint validation, else the "comment" (nas:description) of
the OclConstraint is reset to what the SC parser finds in the constraint text
(nas:definition = OCL expression) -->
          <rule name="rule-trf-all-postprocess-skip-constraint-validation"/>
        </ProcessRuleSet>
      </rules>
    </Transformer>
    <Transformer id="TRF3" input="TRF2" class=
"de.interactive_instruments.ShapeChange.Transformation.Flattening.Flattener" mode=
"enabled">
      <parameters>
        <ProcessParameter name="includeUnionIdentifierTaggedValue" value="true"/>
        <ProcessParameter name="replaceUnionExcludeRegex" value=
"MaritimeBottomCharacter(MaterialCode|MaterialQualCode|SedimentColourCode)Reason"/>
      </parameters>
      <rules>
        <ProcessRuleSet name="flattener">
          <rule name="rule-trf-cls-replace-with-union-properties"/>
          <!-- OCL expressions that rely on property value(s)OrReason will be invalid
- to keep the log clean we've disabled constraint validation -->
          <rule name="rule-trf-all-postprocess-skip-constraint-validation"/>
        </ProcessRuleSet>
      </rules>
    </Transformer>
  </transformers>

  <targets>

    <TargetOwl inputs="TRF3" class=
"de.interactive_instruments.ShapeChange.Target.Ontology.OWLISO19150" mode="enabled">
     <targetParameter name="outputDirectory" value="C:/testbed14/results/ontology"/>
      <!-- Output format identifiers: turtle ntriples nquads rdfxml jsonld rdfjson
trig trix rdfthrift -->
      <targetParameter name="outputFormat" value="rdfxml"/>
      <targetParameter name="defaultEncodingRule" value="NEO"/>
      <targetParameter name="language" value="en"/>
      <targetParameter name="defaultTypeImplementation" value="owl:Class"/>

      <targetParameter name="ontologyNameCode" value="neo"/>
      <targetParameter name="source" value="NSG Enterprise Ontology (NEO) Standard
8.0"/>
      <targetParameter name="URIbase" value="http://api.nsgreg.nga.mil/ontology"/>

      <targetParameter name="skosConceptSchemeSuffix" value="_ConceptScheme"/>
      <targetParameter name="codeNamespaceForEnumerations"  value=
"http://api.nsgreg.nga.mil/ontology/neo-enum/8.0"/>
      <targetParameter name="codeListOwlClassNamespaceForEnumerations"  value=
"http://api.nsgreg.nga.mil/ontology/neo-enum/8.0"/>
```

```
    <!-- over-ride default "#" -->
    <!-- <targetParameter name="rdfNamespaceSeparator" value="/"/> -->

    <descriptorTargets>
      <DescriptorTarget appliesTo="ontology" target="rdfs:label" template=
"[[TV:ontologyName]]" format="langString"/>
      <DescriptorTarget appliesTo="ontology" target="skos:definition" template=
"![CDATA[Definition: [[TV:ontologyDefinition]]  Description:
[[TV:ontologyDescription]]]]" noValueText="[None Specified]" format="langString"/>
      <DescriptorTarget appliesTo="ontology" target="rdfs:isDefinedBy" template=
"[[TV:ontologyResourceURI]]" format="IRI"/>
      <DescriptorTarget appliesTo="ontology" target="skos:prefLabel" template=
"[[TV:ontologyName]]" format="langString"/>
      <DescriptorTarget appliesTo="ontology" target="skos:altLabel" template=
"[[TV(|):aliasList]]" multiValueBehavior="splitToMultipleTargets" format="langString
"/>

      <DescriptorTarget appliesTo="class" target="rdfs:label" template=
"[[TV:primaryCode]]" format="langString"/>
      <DescriptorTarget appliesTo="class" target="skos:definition" template=
"![CDATA[Definition: [[TV:definition]]  Description: [[TV:description]]]]"
noValueText="[None Specified]" format="langString"/>
      <DescriptorTarget appliesTo="class" target="rdfs:isDefinedBy" template=
"http://nsgreg.nga.mil/as/view?i=[[TV:mdbPK]]" format="IRI"/>
      <DescriptorTarget appliesTo="class" target="skos:prefLabel" template=
"[[TV:name]]" format="langString"/>
      <DescriptorTarget appliesTo="class" target="skos:altLabel" template=
"[[TV(|):aliasList]]" multiValueBehavior="splitToMultipleTargets" format="langString
"/>

      <DescriptorTarget appliesTo="property" target="rdfs:label" template=
"[[TV:primaryCode]]" format="langString"/>
      <DescriptorTarget appliesTo="property" target="skos:definition" template=
"![CDATA[Definition: [[TV:definition]]  Description: [[TV:description]]]]"
noValueText="[None Specified]" format="langString"/>
      <DescriptorTarget appliesTo="property" target="rdfs:isDefinedBy" template=
"http://nsgreg.nga.mil/as/view?i=[[TV:mdbPK]]" format="IRI"/>
      <DescriptorTarget appliesTo="property" target="skos:prefLabel" template=
"[[TV:name]]" format="langString"/>
      <DescriptorTarget appliesTo="property" target="skos:altLabel" template=
"[[TV(|):aliasList]]" multiValueBehavior="splitToMultipleTargets" format="langString
"/>

      <DescriptorTarget appliesTo="conceptscheme" target="rdfs:label" template=
"[[TV:primaryCode]]_ConceptScheme" format="langString"/>
      <DescriptorTarget appliesTo="conceptscheme" target="skos:definition" template
="![CDATA[Definition: [[TV:definition]]  Description: [[TV:description]]]]"
noValueText="[None Specified]" format="langString"/>
      <DescriptorTarget appliesTo="conceptscheme" target="rdfs:isDefinedBy"
template="http://nsgreg.nga.mil/as/view?i=[[TV:mdbPK]]" format="IRI"/>
      <DescriptorTarget appliesTo="conceptscheme" target="skos:prefLabel" template=
"[[TV:name]] - Concept Scheme" format="langString"/>
      </descriptorTargets>

     <xi:include href="C:/testbed14/config/StandardMapEntries-owl.xml"/>
      <rdfMapEntries>
      <!-- <RdfTypeMapEntry type="Binary" target="owl:Class"/> -->

      <RdfTypeMapEntry type="SecurityAttributesGroupType" target="ntk:RequiresType
"/>
      <RdfTypeMapEntry type="ISM_Notice" target="icism:NoticeType"/>
      <RdfTypeMapEntry type="NTKAccess" target="ntk:RequiresType"/>
      <RdfTypeMapEntry type="RevisionRecall" target="rr:RevisionRecallType"/>
<!--        <RdfTypeMapEntry type="VoidValueReason" target="test:VoidValueReason"/>-->
      </rdfMapEntries>

      <rdfConversionParameters>
      <!--        <StereotypeConversionParameter wellknown="FeatureType"
subClassOf="geo:Feature"/>  -->
      </rdfConversionParameters>

      <constraintMappings>
     <!--   <ConstraintMapping constraintType="OCL" template="[[name]]: [[comment]]
OCL expression: [[text]]" noValue="[None Specified]" format="langString"/> -->
      </constraintMappings>

     <xi:include href="C:/testbed14/config/StandardNamespaces-owl.xml"/>
      <namespaces>
<!--        <Namespace nsabr="test" ns="https://my.test/enums#"/>-->
      <Namespace nsabr="icism" ns=
"https://www.dni.gov/files/documents/CIO/ICEA/Juliet/ISM-Public#" location=
"https://www.dni.gov/files/documents/CIO/ICEA/Juliet/ISM-Public"/>
      <Namespace nsabr="ntk" ns=
"https://www.dni.gov/files/documents/CIO/ICEA/India/NTK-V10-Public#" location=
"https://www.dni.gov/files/documents/CIO/ICEA/India/NTK-V10-Public" />
      <Namespace nsabr="rr" ns=
"https://www.dni.gov/files/documents/CIO/ICEA/Juliet/RevRecall-Public#" location=
"https://www.dni.gov/files/documents/CIO/ICEA/Juliet/RevRecall-Public" />
      </namespaces>

      <rules>
      <EncodingRule name="NEO" extends="*">
        <rule name="rule-owl-pkg-singleOntologyPerSchema"/>
        <rule name="rule-owl-pkg-ontologyName-code"/>
        <rule name="rule-owl-pkg-ontologyName-appendVersion"/>
        <rule name="rule-owl-pkg-versionInfo"/>
        <rule name="rule-owl-pkg-versionIRI"/>
        <rule name="rule-owl-pkg-versionIRI-avoid-duplicate-version"/>
        <rule name="rule-owl-pkg-dctSourceTitle"/>
        <rule name="rule-owl-cls-iso191502IsAbstract"/>
        <rule name="rule-owl-cls-generalization"/>
        <rule name="rule-owl-cls-disjoint-classes"/>
        <rule name="rule-owl-cls-encode-featuretypes"/>
        <rule name="rule-owl-cls-encode-objecttypes"/>
        <rule name="rule-owl-cls-encode-mixintypes"/>
        <rule name="rule-owl-cls-encode-datatypes"/>
        <rule name="rule-owl-cls-encode-basictypes"/>

        <rule name="rule-owl-prop-general"/>
        <rule name="rule-owl-prop-range-global"/>
        <!-- <rule name="rule-owl-prop-range-local-withUniversalQuantification"/>
-->
        <rule name="rule-owl-prop-localScopeAll"/>
        <rule name="rule-owl-prop-multiplicityAsUnqualifiedCardinalityRestriction"/>
        <!-- <rule name="rule-owl-prop-
multiplicityAsQualifiedCardinalityRestriction"/> -->
        <rule name="rule-owl-prop-iso191502AssociationName"/>
        <rule name="rule-owl-prop-inverseOf"/>
        <rule name="rule-owl-prop-iso191502Aggregation"/>
        <!-- <rule name="rule-owl-all-constraints-humanReadableTextOnly"/> -->
     <!--    <rule name="rule-owl-all-constraints-byConstraintMapping"/> -->
        <rule name="rule-owl-cls-union"/>
        <rule name="rule-owl-cls-unionSets"/>
<!--           <rule name="rule-owl-cls-iso191502Enumeration"/> -->
        <rule name="rule-owl-cls-enumerationAsCodelist"/>
        <rule name="rule-owl-cls-codelist-external"/>
        <rule name="rule-owl-cls-codelist-19150-2"/>
        <!--  <rule name="rule-owl-cls-codelist-19150-2-skos-collection"/> -->
        <rule name="rule-owl-cls-codelist-19150-2-objectOneOfForEnumeration"/>
        <rule name="rule-owl-cls-codelist-19150-2-differentIndividuals"/>
        <rule name="rule-owl-cls-codelist-19150-2-owlClassInDifferentNamespace"/>
        <!--  <rule name="rule-owl-cls-codelist-19150-2-conceptSchemeSubclass"/> -->
        <rule name="rule-owl-prop-code-broader-byBroaderListedValue"/>
      </EncodingRule>
      </rules>
    </TargetOwl>

  </targets>
</ShapeChangeConfiguration>
```

A.1.3. Generating the profiles

ShapeChange 2.6.1-SNAPSHOT was used to generate the application profiles by running the following commands:

Command for Simple Linked Data Application Profile Generation

```
java -jar ShapeChange-2.6.1-SNAPSHOT.jar -Dfile.encoding=UTF-8 -c
simpleCivilianAirport.xml

Output:

I ---------- Semantic validation of ShapeChange configuration: START ----------
I --- Validating transformer with @id 'TRF3' ...
I --- Validating transformer with @id 'TRF1' ...
I --- Validating target with @class
'de.interactive_instruments.ShapeChange.Target.Ontology.OWLISO19150' and @inputs
'TRF3' ...
I ---------- Semantic validation of ShapeChange configuration: COMPLETE ----------
I Now processing transformation 'TRF1' for input ID: 'INPUT'.
I ---------- TransformationManager postprocessing: validating constraints ----------
I Performed transformation for transformer ID 'TRF1' for input ID: 'INPUT'.
-------------------------------------------------
I Now processing transformation 'TRF2' for input ID: 'TRF1'.
I Performed transformation for transformer ID 'TRF2' for input ID: 'TRF1'.
-------------------------------------------------
I Now processing transformation 'TRF3' for input ID: 'TRF2'.
I ========== processing phase ==========
I ---------- now processing: rule-trf-cls-replace-with-union-properties ----------
I ========== postprocessing phase ==========
I Performed transformation for transformer ID 'TRF3' for input ID: 'TRF2'.
-------------------------------------------------
I Application schema found, package name: 'NSG Application Schema', target namespace:
'http://api.nsgreg.nga.mil/ontology'
I Now processing target 'ISO 19150-2 OWL Ontology' for input 'TRF3'.
I The default type implementation is 'owl:Class'.
I Executed target class
'de.interactive_instruments.ShapeChange.Target.Ontology.OWLISO19150' for input ID:
'TRF3'.
```

Command for Complex Analytic Application Profile Generation

```
java -jar ShapeChange-2.6.1-SNAPSHOT.jar -Dfile.encoding=UTF-8 -c
complexCivilianAirport.xml

Output:

I ---------- Semantic validation of ShapeChange configuration: START ----------
I --- Validating transformer with @id 'TRF3' ...
I --- Validating transformer with @id 'TRF1' ...
I --- Validating target with @class
'de.interactive_instruments.ShapeChange.Target.Ontology.OWLISO19150' and @inputs
'TRF3' ...
I ---------- Semantic validation of ShapeChange configuration: COMPLETE ----------
I Now processing transformation 'TRF1' for input ID: 'INPUT'.
I ---------- TransformationManager postprocessing: validating constraints ----------
I Performed transformation for transformer ID 'TRF1' for input ID: 'INPUT'.
-------------------------------------------------
I Now processing transformation 'TRF2' for input ID: 'TRF1'.
I Performed transformation for transformer ID 'TRF2' for input ID: 'TRF1'.
-------------------------------------------------
I Now processing transformation 'TRF3' for input ID: 'TRF2'.
I ========== processing phase ==========
I ---------- now processing: rule-trf-cls-replace-with-union-properties ----------
I ========== postprocessing phase ==========
I Performed transformation for transformer ID 'TRF3' for input ID: 'TRF2'.
-------------------------------------------------
I Application schema found, package name: 'NSG Application Schema', target namespace:
'http://api.nsgreg.nga.mil/ontology'
I Now processing target 'ISO 19150-2 OWL Ontology' for input 'TRF3'.
I The default type implementation is 'owl:Class'.
I Executed target class
'de.interactive_instruments.ShapeChange.Target.Ontology.OWLISO19150' for input ID:
'TRF3'.
```

NOTE The application profile generated for Complex Analytic Application was pretty limited in expresiveness due to the lack of axioms in NEO to represent property hierarchies and property characteristics (transitive, functional, etc).

A.2. Sample Data

To generate the sample data (dolon.rdf), the simple RDF application profile was imported into Topbraid and sample data were created manually by creating a fictitious Airport and Heliport in Dolon. The dataset was inspired from sample data encoded in AIXM, which contains similar elements (http://github.com/aixm/donlon).

Only one sample dataset was generated, as they can be used by both application profiles (as they share the same namespace). The result of the inferences was richer in the case of the complex analytic application profile as it contained more expressive axioms.

The following is a sample of the dataset:

```
dolon:DolonAirport
  rdf:type neo-ent:LandAerodrome ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.aerodromeBeacon>
dolon:DolonAirportBeacon1 ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.aerodromeBoundary>
dolon:DolonAirportBoundary ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.aerodromeElevation> [
      rdf:type neo-ent:MeasureMeta ;
      <http://api.nsgreg.nga.mil/ontology/neo/1-8#MeasureMeta.value> "253
m"^^sc:Measure ;
      rdfs:label "253 meter" ;
    ] ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.aerodromeLocationDesc> [
      rdf:type neo-ent:TextLexUnconMeta ;
      <http://api.nsgreg.nga.mil/ontology/neo/1-8#TextLexUnconMeta.value> "Dolon" ;
      rdfs:label "Dolon" ;
    ] ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.aerodromeOfficialName> [
      rdf:type neo-ent:TextNonLex80Meta ;
      rdfs:label "Dolon Airport" ;
    ] ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.airfieldType>
<http://api.nsgreg.nga.mil/vocabulary/ncv/AirfieldTypeTermSet/major> ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.faaIdentifier>
dolon:StructuredTextMeta_1 ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.highestElevation> [
      rdf:type neo-ent:MeasureMeta ;
      <http://api.nsgreg.nga.mil/ontology/neo/1-8#MeasureMeta.value> "256
m"^^sc:Measure ;
      rdfs:label "256 m" ;
    ] ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.installationLocation>
dolon:DolonAerodromeInstLocation ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.movementArea>
dolon:DolonAirportApron1 ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.movementArea>
dolon:DolonAirportApron2 ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.movementArea>
dolon:DolonAirportApron3 ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.movementArea>
dolon:DolonAirportTaxiway_1 ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.movementArea>
dolon:DolonAirportTaxiway_2 ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.movementArea> dolon:Helipad_1
;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.movementArea> dolon:Helipad_2
;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.movementArea> dolon:Runway_1 ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.movementArea> dolon:Runway_2 ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Aerodrome.referencePoint> [
      rdf:type neo-ent:PointPositionInfo ;
      rdfs:label "" ;
    ] ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Entity.uniqueEntityIdentifier>
<urn:uuid.dd062d88-3e64-4a5d-bebd-89476db9ebea> ;
  rdfs:label "Dolon Airport" ;
.


dolon:DolonAirportApron1
  rdf:type neo-ent:Apron ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#AerodromeMoveArea.atAerodrome>
dolon:DolonAirport ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-
8#AerodromeMoveArea.featureOperationalStatus>
<http://api.nsgreg.nga.mil/vocabulary/ncv/FeatureOperationalStatusTermSet/operational>
;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#AerodromeMoveArea.highestElevation> [
      rdf:type neo-ent:MeasureMeta ;
      <http://api.nsgreg.nga.mil/ontology/neo/1-8#MeasureMeta.value> "254
m"^^sc:Measure ;
      rdfs:label "254 m" ;
    ] ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Apron.apronType>
<http://api.nsgreg.nga.mil/vocabulary/ncv/ApronTypeTermSet.cargo> ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Apron.apronUsage>
<http://api.nsgreg.nga.mil/vocabulary/ncv/ApronUsageTermSet.parking> ;
  <http://api.nsgreg.nga.mil/ontology/neo/1-8#Apron.featureDesignator> [
      rdf:type neo-ent:TextNonLexUnconMeta ;
      <http://api.nsgreg.nga.mil/ontology/neo/1-8#TextNonLexUnconMeta.value> "A1" ;
      rdfs:label "" ;
    ] ;
  rdfs:label "Apron A1" ;
.
```

A.3. Generated Artifacts

The following artifacts were generated for this project and commited in GitHub at: https://github.com/opengeospatial/D022-1_Characterization_of_RDF_Application_Profiles_Engineering_Report/tree/master/profiles

Table 6. Artifacts generated by ShapeChange process flow.

Name	Description
pmp-export/nas-export	PMT export of the Application Profile from NAS
configuration/simpleCivilianAirport.xml	ShapeChange configuration for Simple Linked Data Application

Name	Description
configuration/complexCivilianAirport.xml	ShapeChange configuration for Complex Analytic Application
simple/neo.rdf	RDF Application profile for Simple Linked Data Application
simple/enums.rdf	Code lists for Simple Linked Data Application
simple/neo-codes.rdf	Codelist_ConceptScheme for Simple Linked Data Application
complex/neo.rdf	RDF Application profile for Complex Analytic Application
complex/enums.rdf	Code lists for omplex Analytic Application
complex/neo-codes.rdf	Codelist_ConceptScheme for Complex Analytic Application
samples/dolon.ttl	Sample dataset compliant with the application profiles.

Appendix B: Revision History

Table 7. Revision History

Date	**Editor**	**Release**	**Primary clauses modified**	**Descriptions**
Oct. 27, 2018	Stephane Fellah	.1	all	Splitting of original ER

Appendix C: Bibliography

1. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific american. 284, 34–43 (2001).
2. Lanier, J.: The complexity ceiling chapter in Next Fifty Years : Science In The First Half Of The Twenty-First Century. Vintage (2002).
3. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge acquisition. 5, 199–220 (1993).
4. Obrst, L., Chase, P., Markeloff, R.: Developing an Ontology of the Cyber Security Domain. In: Costa, P.C.G. da and Laskey, K.B. (eds.) STIDS. pp. 49–56. CEUR-WS.org (2012).
5. Baader, F., Calvanese, D., McGuinness, D., Patel-Schneider, P., Nardi, D.: The description logic handbook: Theory, implementation and applications. Cambridge university press (2003).
6. Knublauch, H.: SHACL and OWL Compared. (2017).
7. Kurilovas, E.: Interoperability, standards and metadata for e-Learning. In: Intelligent Distributed Computing III. pp. 121–130. Springer (2009).
8. Fellah, S.: OGC Testbed-12 Semantic Portrayal, Registry and Mediation. OGC 16-059,Open Geospatial Consortium, http://docs.opengeospatial.org/per/16-059.html (2017).
9. Maali, F., Erickson, J., Archer, P.: Data catalog vocabulary (DCAT). W3C Recommendation. 16, (2014).
10. Dekkers, M.: Asset description metadata schema (adms). W3C Working Group. (2013).