4

ALIGNED MetaModel Overview

Rob Brennan[1], Bojan Bozic[1], Odhran Gavin[1] and Monika Solanki[2]

[1]Trinity College Dublin, Ireland
[2]University of Oxford, UK

The foundation of our ALIGNED methodology is an RDF-based semantic metamodel or language to describe software and data life cycles, inter-life cycle events, design intent, and domain models. This common framework for software and data engineering enables the following techniques for managing complexity: (1) Model-driven software engineering of data-intensive systems based on Linked Data; (2) Integrating expert-based data curation workflows into the software and data quality cycles; and (3) Providing unified views and governance of both software and data engineering activities when developing data-intensive systems;

This common metamodel for software and data engineering describes data-intensive systems both at a system specification level and in terms of the engineering activities, actors and artefacts.

Figure 4.1 illustrates the ALIGNED metamodels. At the top layer (the generic metamodel), it documents the common concepts used in data-intensive systems as a set of Linked Data vocabularies. The next ALIGNED layer covers the domain-specific metamodels that constitute a vocabulary and constraints for operating in a specific domain. This layer constrains the types of data-intensive systems that can be built in terms of architecture and tools, best practices for data collection and curation and common data assets (e.g., Linked Data datasets to be consumed by applications in this domain). ALIGNED has developed four domain-specific metamodels based on each of our use cases: enterprise information processing (JURION), e-research in the Social Sciences and Humanities (Seshat), crowdsourced public datasets (DBpedia), and enterprise software development (PoolParty).

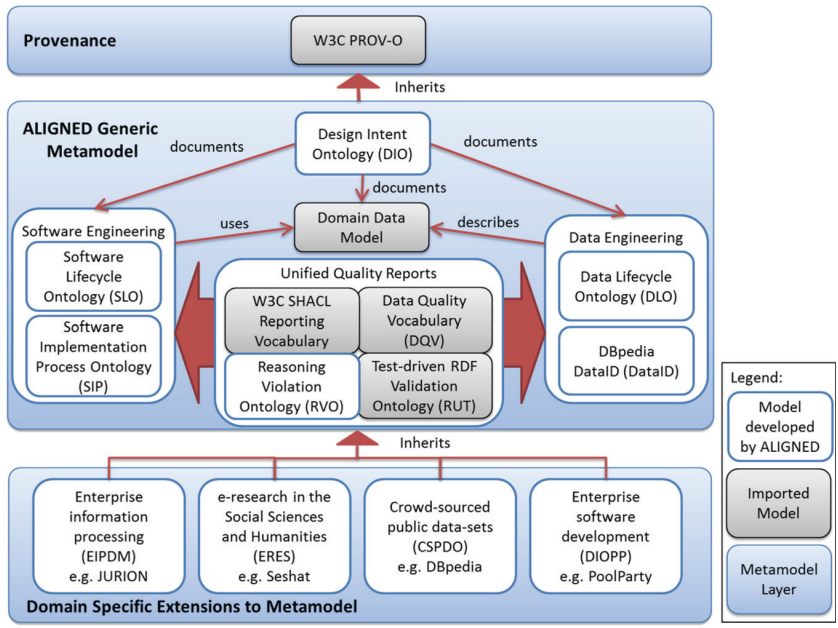


Figure 4.1 The ALIGNED metamodel layers.

Both the generic metamodel and domain-specific model layers are further specified in the following sections of this document.

4.1 Generic Metamodel

As specified in the last section, the ALIGNED metamodel is split into two major layers: the upper or generic layer is described in this section.

4.1.1 Basic Approach

The ALIGNED generic metamodel is structured as a set of complementary vocabularies that can be used to document the development and design of a data-intensive system throughout its life cycle. It extends the W3C PROV Ontology (PROV-O) to define software and data engineering agents, activities and entities. This facilitates the creation of provenance records describing software and data engineering.

The constituent vocabularies defined are as follows:

Software Life cycle Vocabulary (SLO and SIP): This is split into two components: the Software Life cycle Ontology (SLO) and the Software Implementation Process Ontology (SIP). SLO defines a top-level ontology

for describing life cycle processes. SIP uses SLO to define the major agents (project roles, classes of software tools, etc.), activities (life cycle stages) and entities (models, code, test cases, etc.) involved in a software engineering project and their relations. SIP is implemented as a set of RDF modules implementing specific processes in the software development and implementation life cycle.

Data Life Cycle Ontology (DLO): Defines the major agents (project roles, classes of software tools, etc.), activities (life cycle stages) and entities (schema, datasets, code, test cases, etc.) involved in a data engineering project and their relations with a special focus on capturing the engineering life cycle.

Design Intent Ontology (DIO): Used to document the design decisions about data-intensive system artefacts such as software components or datasets. The purpose of the DIO ontology is to model the design intent or design rationale while undertaking the design of any artefact. A design intent or design rationale is an explicit documentation of the reasons behind decisions made when designing a system or artefact.

Domain Vocabulary: Describes the domain(s) of a data-intensive system. It is the specific data model or knowledge model used within the data-intensive system. The SLO, DLO, and DIO vocabularies are used to document additional context or constraints for the domain vocabulary to support semantics-driven software engineering, data quality engineering, engineering project governance, and tool integration.

In the subsections below, some basic details about our specification approach are described. Then we provide an overview of the structure and contents of each vocabulary.

4.1.2 Namespaces and URIs

Table 4.1 lists the standard prefixes used for each vocabulary. All have been checked for clashes with prefix.cc.

Table 4.1 Generic metamodel namespace declarations

Generic Metamodel Vocabulary Name	Prefix
Data Life Cycle Ontology	dlo
Design Intent Ontology	dio
Domain Vocabulary	This is defined by the specific data-intensive system rather than by the ALIGNED metamodel.
Software Life Cycle Vocabularies	slo, sip

Each prefix has been registered as a persistent URL (PURL) with purl.org or the W3C community persistent name service. These namespaces will be maintained by TCD servers.

4.1.3 Expressivity of Vocabularies

Since these generic vocabularies are designed to have the widest possible reuse, they only require the use of RDFS semantics. However, full utilisation of the model also requires the use of the W3C PROV ontology and in line with that specification the OWL2 RL profile is used for advanced features of the model.

4.1.4 Reference Style for External Terms

The ALIGNED metamodel vocabularies (DIO, SLO, DLO) must reference terms from each other and from externally defined vocabularies or ontologies. This necessitates an ontology implementation style decision that ranges from full OWL import statements to free-flowing Linked Data with no defined style or structure. For ALIGNED, the consortium has decided to adopt the MIREOT (Minimum information to reference an external ontology term) implementation style guide.[1] This avoids the practical problems with OWL imports and yet provides some structure around the reuse of existing resources.

4.1.5 Links with W3C PROV

The basic strategy for the ALIGNED metamodel is to specialise the W3C PROV ontology to describe software and data engineering activities (processes, tasks), entities (engineering artefacts or concepts) and agents (roles or software tools). Examples include:

- prov:Activity – sub-types defined to describe data or software engineering life cycle stages
- prov:Plan – used to describe engineering workflows
- prov:Entity – to describe software or data engineering artefacts – test case, design, test results, and so on
- prov:SoftwareAgent – to describe software engineering tools
- prov:Role – for software and data engineering roles

This approach means that PROV acts as a common upper ontology for all of our metamodel vocabularies and binds them together into a coherent whole.

[1] http://obi-ontology.org/page/MIREOT

It also facilities the creation of provenance records describing software and data engineering. The software and data engineering tools created in ALIGNED generate these PROV records as a way of logging their activities using enterprise Linked Data. This common representation of the domain facilitates tool integration and the creation of unified governance tools for combined software and data engineering.

4.2 ALIGNED Generic Metamodel

4.2.1 Design Intent Ontology (DIO)

The purpose of the DIO ontology is to model the design intent or design rationale while undertaking the design of any artefact. A design intent or design rationale is an explicit documentation of the reasons behind decisions made when designing a system or artefact.

The Design Intent Ontology (DIO)[2] is a generic ontology that provides the conceptualisation needed to capture the knowledge generated during various phases of the overall design life cycle. It provides definitions for design artefacts such as requirements, designs, design issues, solutions, justifications, and evidence and relationships between them to represent the design process and how these things lead to design outcomes. It draws upon the paradigms of IBIS (Interactive Intent-Based Illustration),[3] argumentation, and design rationale. It is linked to W3C PROV by defining the actors in the design process as PROV agents and the design artefacts themselves are PROV entities. It makes few assumptions about the design process used as the definitions of these activities properly belongs in the software life cycle and data life cycle models. Figure 4.2 illustrates the conceptual entities in DIO and their relationships.

4.3 Software Engineering

4.3.1 Software Life Cycle Ontology

The purpose of the SLO is to provide a top-level ontology for describing a process in the life cycle of a software. The ontology conforms to the ISO/IEC 12207 standard for Systems and software engineering – Software life cycle processes. The terminology used in the ontology conforms to ISO/IEC TR 24774:2010(E). All subprocesses will require to import this module.

[2] http://purl.org/dio/

[3] http://www.cs.columbia.edu/~doree/IBIS/thesis.html

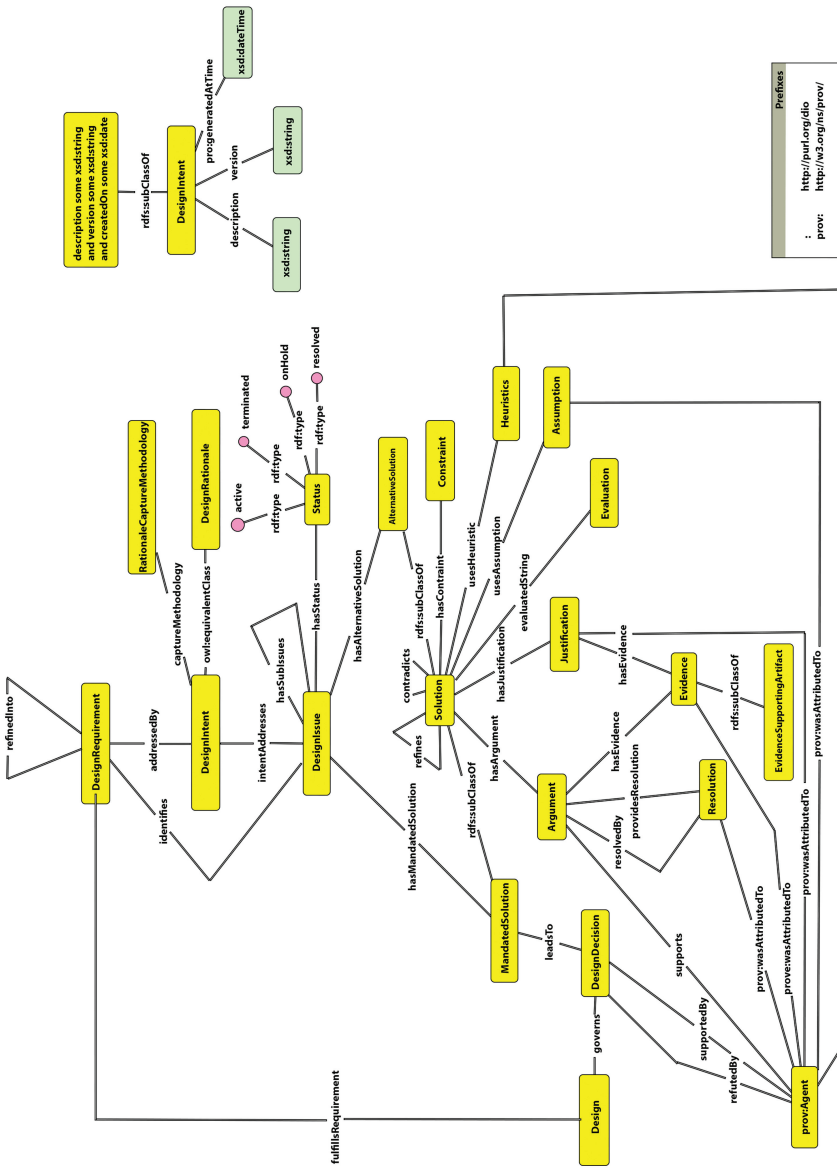


Figure 4.2 The Design Intent Ontology (DIO).

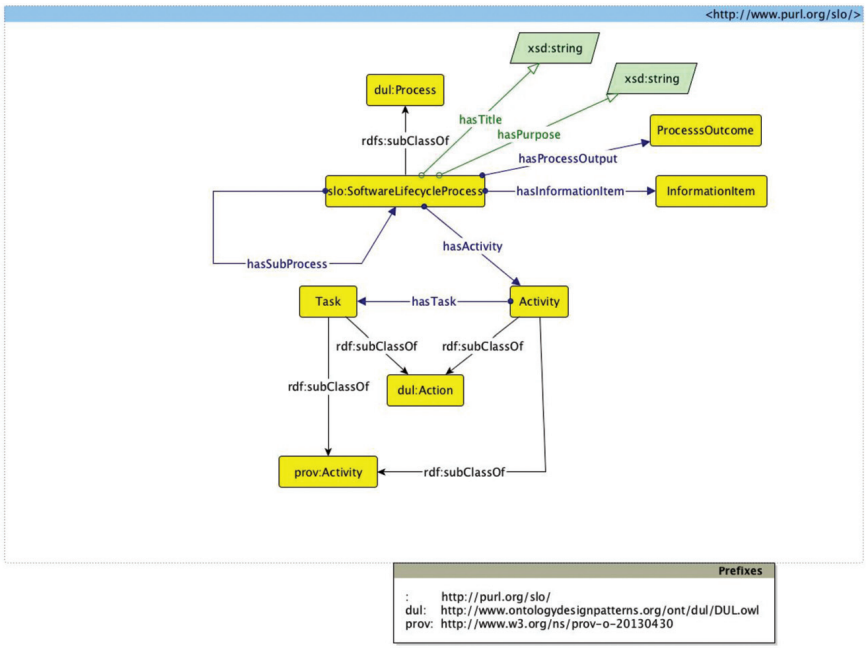


Figure 4.3 The Software Life cycle Ontology.

Figure 4.3 illustrates the conceptual entities in SLO. The core concept is a SoftwareLifecyleProcess, which can be decomposed into sub-processes, tasks and activities. The SIP ontology (see below) builds on this basic framework to describe standard software engineering processes e.g., requirements analysis and architectural design.

4.3.2 Software Implementation Process Ontology (SIP)

The purpose of the SIP is to provide a set of conceptual entities to represent a specified system element implemented as a software product or service.

This ontology imports and builds upon the ALIGNED SLO as the basic description of a process. It also utilises concepts defined in the SEON (Software Evolution ONtologies)[4] and the Software Ontology (SWO).[5]

The basic concepts of the SIP ontology are illustrated in Figure 4.4. It shows the definition of basic software engineering processes and activities

[4]http://www.se-on.org/
[5]http://purl.obolibrary.org/obo/swo.owl

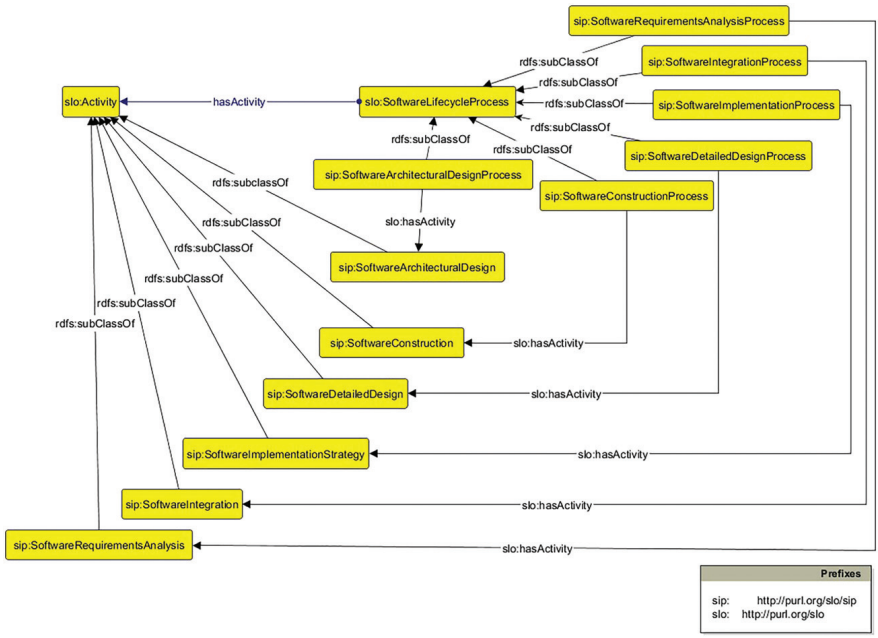


Figure 4.4 Core Concepts of the Software Implementation Process (SIP) Ontology.

such as requirements analysis, design, implementation, integration in terms of SLO activities and processes.

4.4 Data Engineering

4.4.1 Data Life Cycle Ontology

The purpose of the DLO is to provide a set of conceptual entities, agents, activities, and roles to represent the general data engineering process. Furthermore, it is the basis for deriving specific domain ontologies which represent life cycles of concrete data engineering projects such as DBpedia or Seshat.

Figure 4.5 shows the main classes of the data life cycle model. We have used the W3C PROV ontology, in this example represented by the classes Role, Person, Entity, and Activity. We use the Process class which is derived from Activity to implement the Linked Data Stack life cycle stages as subclasses. This allows us to represent LOD activities in our data life cycle metamodel. In addition, we have modelled datasets, data sources, and

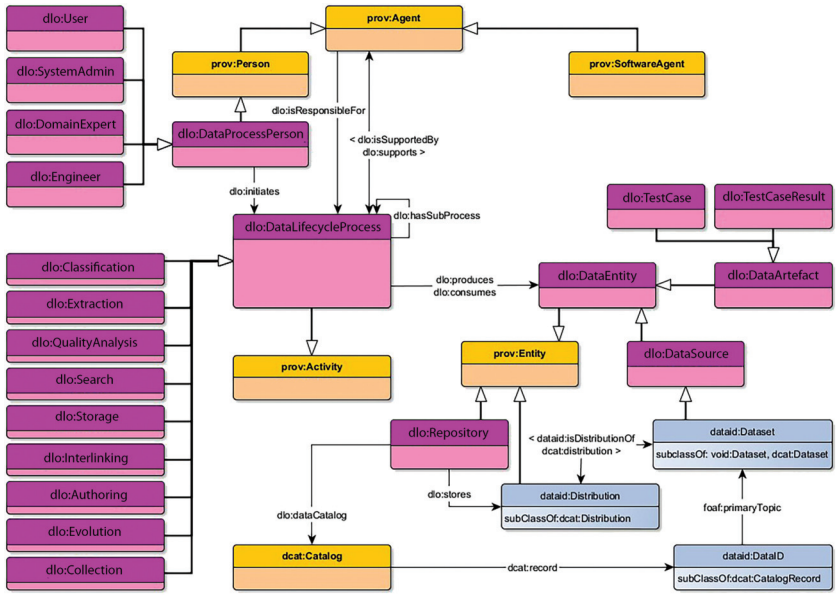


Figure 4.5 Generic data life cycle metamodel (DLO).

data repositories. For datasets, we import the W3C Data Catalog Vocabulary (DCAT)[6] definition of a dataset as it is a broad definition that goes beyond representing only RDF-based datasets.

The full documentation and OWL ontology file of the ALIGNED data life cycle model can be downloaded from http://www.essepuntato.it/lode/owlapi/https://w3id.org/dlo.

4.5 DBpedia DataID (DataID)

DataID is a multi-layered metadata system, extending both the DCAT and PROV Ontology to provide more specific dataset metadata. Depending on context, type of data and use case, this core ontology can be augmented by multiple existing extensions (e.g., Linked Data, repository descriptions, etc.).

DataID core, as the kernel element of this ecosystem, describes datasets and their different manifestations, as well as relations to agents like persons or organisations, in regard to their rights and responsibilities. Together with

[6] http://www.w3.org/TR/vocab-dcat/

DLO, DataID core constitutes the data management side of the ALIGNED Suite of Ontologies.[7]

The DBpedia DataID core vocabulary is a metadata system for detailed descriptions of datasets and their different manifestations. Established vocabularies like DCAT, VoID, PROV-O and FOAF are reused for maximum compatibility, in order to establish a uniform and accepted way to describe and deliver dataset metadata for arbitrary datasets and to put existing standards into practice. In addition, DataID can describe the relations of Agents (like persons or organisations) to datasets with regard to their rights and responsibilities.

Due to the growing complexity and different usage purposes, the DataID ontology was modularised into a core ontology and multiple mid-layer ontologies. While the core ontology is mandatory for any of the mid-level ontologies presented, none of those are required for describing data. That being said, in many use cases, some or all the mid-level ontologies will be a useful extension.

The DataID core vocabulary (Figure 4.6) describes datasets (based heavily on the DCAT ontology), as well as their relation to agents like persons or organisations with regard to their rights and responsibilities.

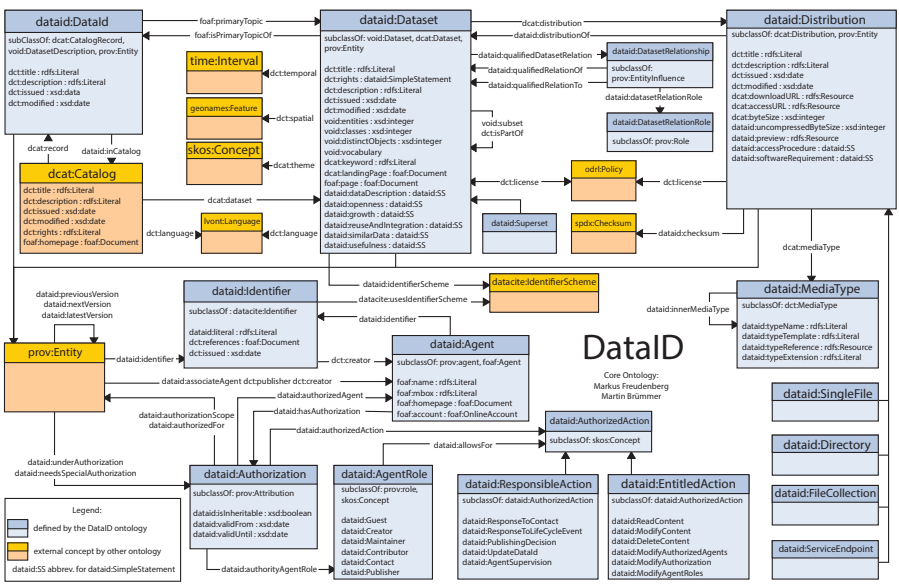


Figure 4.6 The DataID Ontology.

[7] http://aligned-project.eu/data-and-models

The full documentation and OWL ontology file of DataID can be downloaded from DBpedia,[8] the DataID landing page at DBpedia,[9] and through the ALIGNED website.

4.6 Unified Quality Reports

4.6.1 Reasoning Violation Ontology (RVO) Overview

The purpose of RVO is to enable a reasoner to describe reasoning errors detected in an input ontology, in order to facilitate the integration of reasoners into semantic Web tool chains.

It is defined as a simple OWL 2 ontology that is amenable to RDFS-based interpretations or use as a Linked Data vocabulary without any dependence on reasoning. A permanent identifier for the ontology has been registered with the W3C permanent identifier community group. The full source of the ontology is published online. This ontology is used to describe RDF and OWL reasoning violation messages in the Dacura Quality Service. These are generated by running an RDF/RDFS/OWL-DL reasoner over an RDF-based ontology model and allowing the Dacura quality service to report any integrity violations detected at schema or instance level. These violations report areas where the input model is logically inconsistent or breaks RDFS/OWL semantics or axioms. Violations may be reported as based on open world or closed world assumptions. The open world is the default OWL semantics and can typically only detect a limited number of problems due to incomplete knowledge. The closed world interpretation assumes that you have provided all relevant aspects of the model and is able to detect a much wider range of violations, e.g., missing or misspelled term definitions. This is often useful during ontology development or in a system that interprets OWL as a constraint language.

RVO will allow machine-readability and interpretation of detailed reasoning error messages. Furthermore, this would enable building tools to verify the OWL DL compliance of an ontology, find out which best practice requirements the ontology meets or violates, track the impact of interpreting the ontology in open and closed world contexts, identify the exact position of violations, and support intelligent visualisation of errors. The structure of the base RVO classes is shown in Figure 4.7.

[8] http://dataid.dbpedia.org/ns/core.html
[9] http://dbpedia.org/projects/dbpedia-dataid#Data%20model

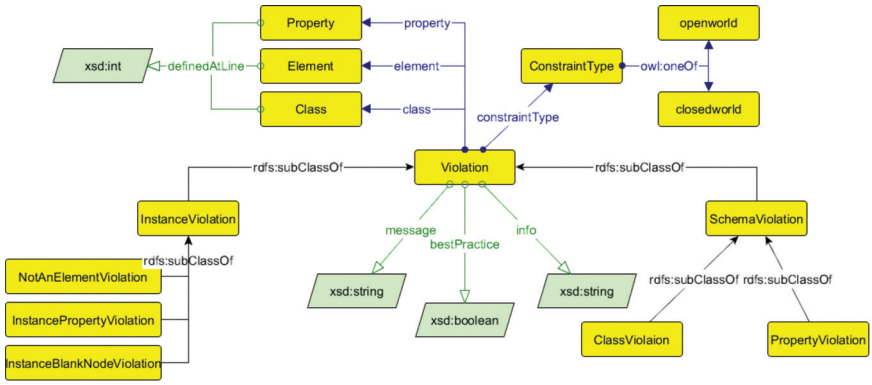


Figure 4.7 Reasoning Violation Ontology (RVO) Base Classes.

RVO class and instance violations are shown in Figure 4.8. Class violations are used for reporting issues regarding the TBox and instance violations ABox in general. Therefore, class violations are reported when e.g., property domains are missing, subsumption errors are detected, or class and property cycles are found. Instance violations show instances which are not elements of valid classes, cardinalities which are incorrect, property constraints that are violated, literals and objects which are confused, and so on.

The full documentation and OWL ontology file for RVO can be downloaded using the LODE documentation service and the persistent URI for the ontology.[10]

Example

This example shows a ClassViolation which is a SchemaViolation and more specifically a ClassCycleViolation. Such specific violation detection results make it possible to provide exact suggestions to ontology developers or repair agents and trigger ontology improvements. Figure 4.9 shows the errors produced by this violation.

Ontology Snippet Producing the Violation:

seshat:Territory seshat:hasValue xsd:DateTime.

[10]http://www.essepuntato.it/lode/owlapi/https://w3id.org/rvo

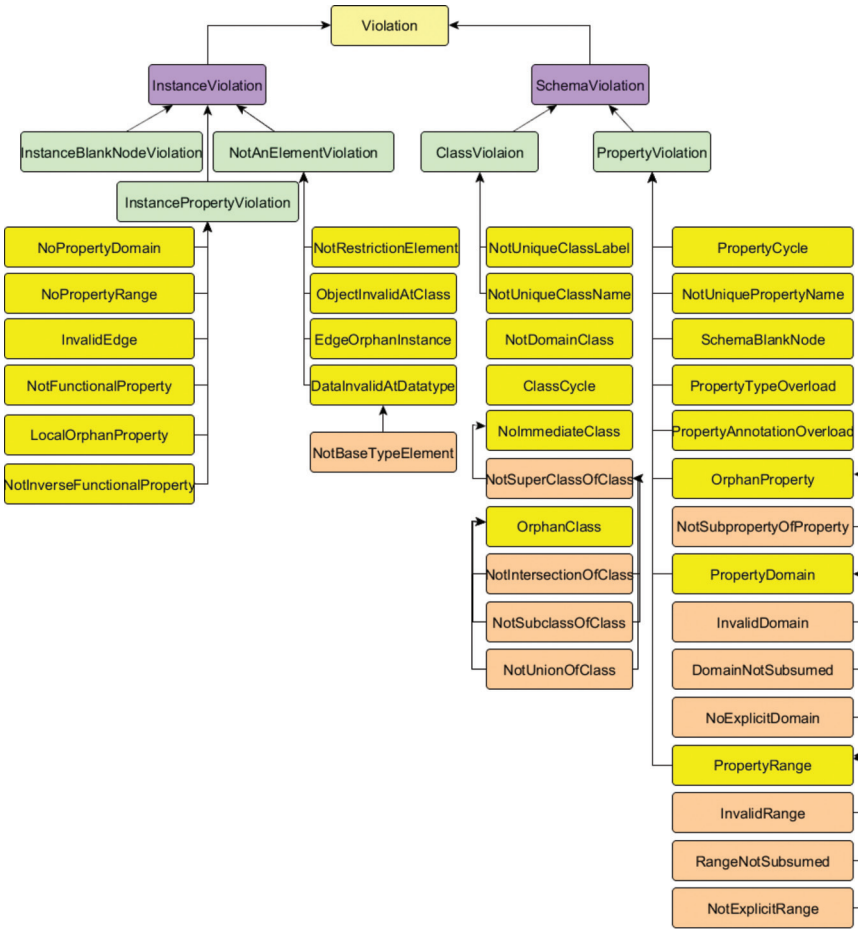


Figure 4.8 RVO Instance and Schema Violation Classes.

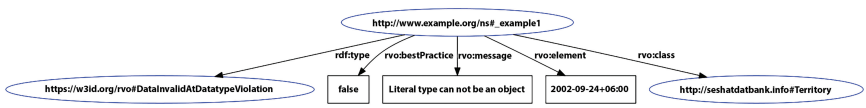


Figure 4.9 Resulting RDF Graph after Validation.

4.6.2 W3C SHACL Reporting Vocabulary

The Shapes Constraint Language is a language to validate RDF graphs against a set of constraints. These constraints are formalised as shapes and other constructs expressed in the form of an RDF graph. The language features and approaches occurring in the current specification of SHACL were in

part inspired by the SPIN[11] and Shape Expressions (ShEx). The current revision of the specification for SHACL is published by the W3C[12] with complementary material available in a GitHub repository.[13]

SHACL Core defines frequently needed features to formulate common constraints for RDF graphs. SHACL Core Constraints are defined by parameterising Constraint Components that are templates for checks for a specific required property of an RDF nodes (e.g., unique occurrence of a property value associated with a specific property, for instance only one foaf:age value for a given foaf:Person). One or several of such constraints are associated with target RDF nodes to validate against in a SHACL Shape. SHACL shapes are expressed as RDF resources and aggregated in a Shapes Graph. An RDF graph to be checked for conformance against a Shapes Graph (the Data Graph) is provided to a Validation Engine that produces a Validation Report. The Validation Report states whether the Data Graph conforms to the Shapes Graph, listing violations of individual RDF nodes against shapes detected during the validation process in case of non-conformance.

SHACL Example

The following example data graph contains three SHACL instances of the class ex:Person. It is taken from the SHACL documentation.

```
ex:Alice
       a ex:Person ;
       ex:ssn "987-65-432A" .
ex:Bob
       a ex:Person ;
       ex:ssn "123-45-6789" ;
       ex:ssn "124-35-6789" .
ex:Calvin
       a ex:Person ;
       ex:birthDate "1971-07-07"^^xsd:date ;
       ex:worksFor ex:UntypedCompany .
```

The following conditions are shown in the example:

A SHACL instance of ex:Person can have at most one value for the property ex:ssn, and this value is a literal with the datatype xsd:string that matches a specified regular expression.

A SHACL instance of ex:Person can have unlimited values for the property ex:worksFor, and these values are IRIs and SHACL instances of ex:Company.

[11] http://spinrdf.org/

[12] https://www.w3.org/TR/shacl/

[13] https://github.com/w3c/data-shapes

A SHACL instance of ex:Person cannot have values for any other property apart from ex:ssn, ex:worksFor and rdf:type.

These conditions can be represented as shapes and constraints in the following shapes graph:

```
ex:PersonShape
       a sh:NodeShape ;
       sh:targetClass ex:Person ;     # Applies to all persons
       sh:property [                  # _:b1
              sh:path ex:ssn ;            # constrains the values of ex:ssn
              sh:maxCount 1 ;
              sh:datatype xsd:string ;
              sh:pattern "^\\d{3}-\\d{2}-\\d{4}$" ;
       ] ;
       sh:property [                  # _:b2
              sh:path ex:worksFor ;
              sh:class ex:Company ;
              sh:nodeKind sh:IRI ;
       ] ;
       sh:closed true ;
       sh:ignoredProperties ( rdf:type ) .
```

The shape declaration above illustrates some of the key terminology used by SHACL. The target for the shape ex:PersonShape is the set of all SHACL instances of the class ex:Person. This is specified using the property sh:targetClass. During the validation, these target nodes become focus nodes for the shape. The shape ex:PersonShape is a node shape, which means that it applies to the focus nodes. It declares constraints on the focus nodes, for example using the parameters sh:closed and sh:ignoredProperties. The node shape also declares two other constraints with the property sh:property, and each of these is backed by a property shape. These property shapes declare additional constraints using parameters such as sh:datatype and sh:maxCount.

Some of the property shapes specify parameters from multiple constraint components in order to restrict multiple aspects of the property values. For example, in the property shape for ex:ssn, parameters from three constraint components are used. The parameters of these constraint components are sh:datatype, sh:pattern and sh:maxCount. For each focus node the property values of ex:ssn will be validated against all three components.

4.6.3 Data Quality Vocabulary

The Data Quality Vocabulary (DQV) is an extension to the DCAT vocabulary which covers data quality, frequency of updates, user correction, persistence, and other properties of the dataset in question. It is designed to improve trust in data. It does not provide a determination of what quality is, but instead

seeks to allow data consumers to judge whether the data in a dataset is suitable for their uses, and to publish their opinions and annotations about the dataset and its quality. The vocabulary seeks to do this by making it easier to publish, exchange, and consume metadata at every step of the dataset life cycle. Figure 4.10 shows the DQV ontology.

The quality of a dataset is assessed via certain observed properties. To express these properties, an instance of a dcat:Dataset or dcat:Distribution can be related to five different types of quality information represented by the following classes:

- dqv:QualityAnnotation represents feedback and quality certificates given about the dataset or its distribution.
- dcterms:Standard represents a standard the dataset or its distribution conforms to.
- dqv:QualityPolicy represents a policy or agreement that is chiefly governed by data quality concerns.
- dqv:QualityMeasurement represents a metric value providing quantitative or qualitative information about the dataset or distribution.

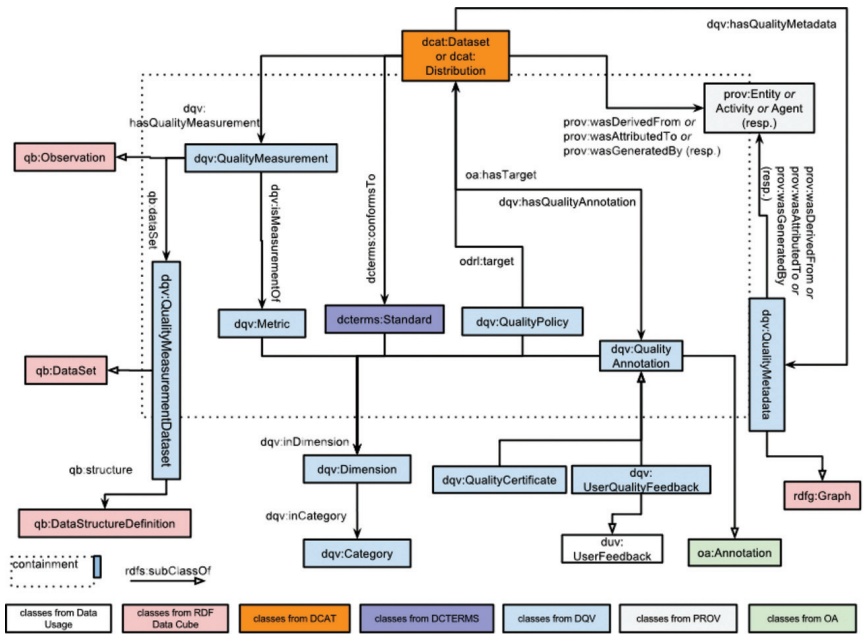


Figure 4.10 Data model showing the main relevant classes and their relations.

- prov:Entity represents an entity involved in the provenance of the dataset or distribution.

DQV defines quality measures as specific instances of Quality Measurements, adapting the daQ quality framework. It relies on quality dimensions and quality metrics. Figure 4.11 shows the interrelation of statements about data quality.

A Quality Dimension (dqv:Dimension) is a quality-related characteristic of a dataset relevant to the consumer (e.g., the availability of a dataset).

A Quality Metric (dqv:Metric) gives a procedure for measuring a data quality dimension, which is abstract, by observing a concrete quality indicator. There are usually multiple metrics per dimension; e.g., availability can be indicated by the accessibility of a SPARQL endpoint, or that of an RDF dump. The value of a metric can be numeric (e.g., for the metric "human-readable labeling of classes, properties and entities", the percentage of entities having an rdfs:label or rdfs:comment) or Boolean (e.g., whether or not a SPARQL endpoint is accessible).

Besides quality measurements, DQV considers certificates, standards, and quality policies, which can also be organised according to dimensions. Quality metadata containers (dqv:QualityMetadata) can group together different quality statements, so that their provenance can be tracked jointly.

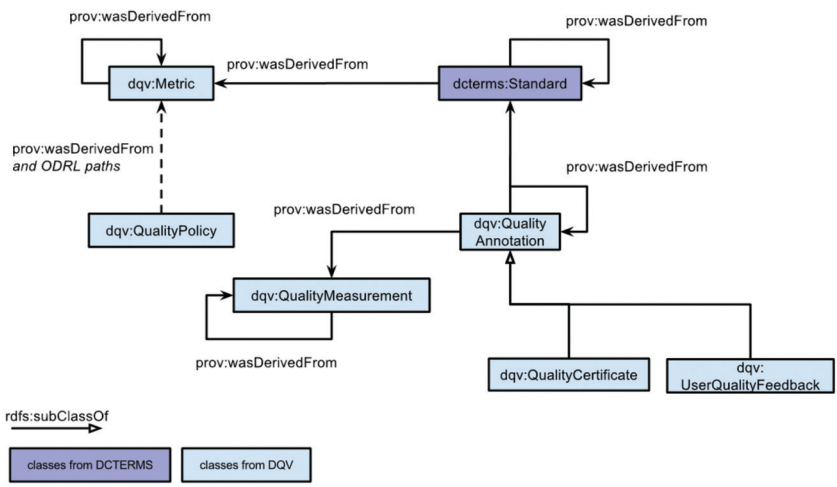
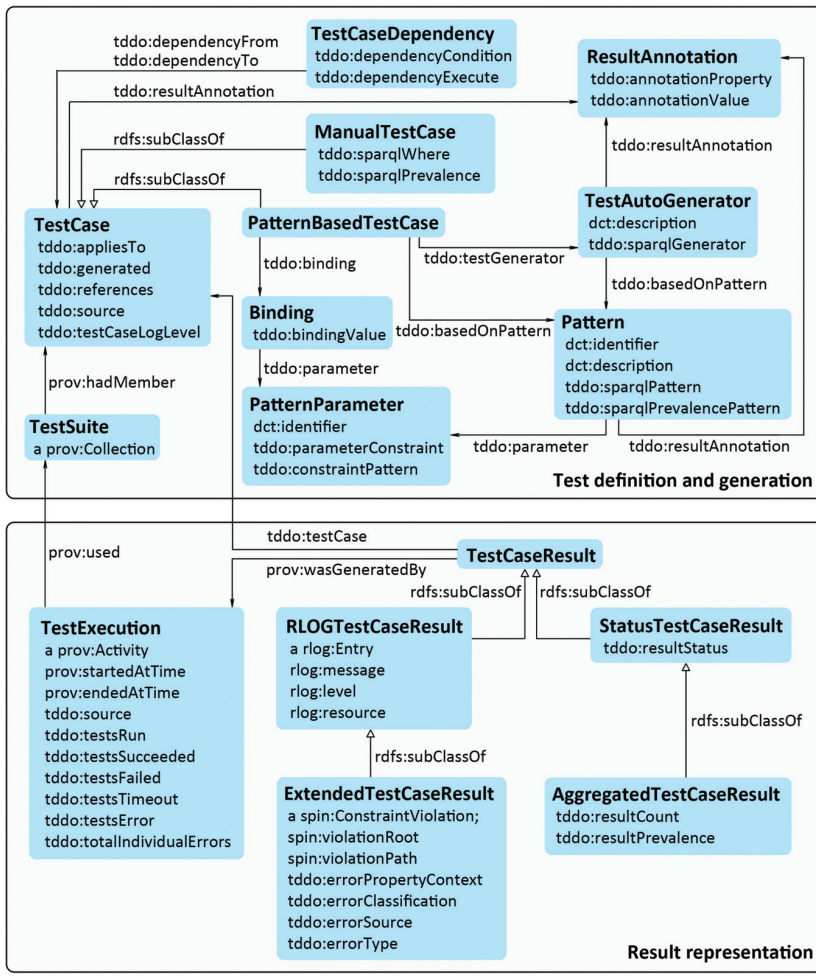


Figure 4.11 Using the property prov:wasDerivedFrom to interrelate quality metrics and other quality statements.

4.6.4 Test-Driven RDF Validation Ontology (RUT)

The RDFUnit ontology describes concepts used in RDFUnit, a test-driven RDF Validation framework that can run automatically generated (based on a schema) and manually generated test cases against an endpoint.[14]



Prefix prov: http://www.w3.org/ns/prov#
Prefix spin: http://spinrdf.org/spin#
Prefix dct: http://purl.org/dc/terms/
Prefix tddo: http://databugger.aksw.org/ns/core#
Prefix rlog: http://persistence.uni-leipzig.org/nlp2rdf/ontologies/rlog#

[14] "NLP data cleansing based on Linguistic Ontology constraints" pp. 5–7, http://jens-lehmann.org/files/2014/eswc_rdfunit_nlp.pdf, ESWC, 2014.

Table 4.2 Domain-specific metamodel namespace declarations

Domain-specific Metamodel Vocabulary Name	Prefix
Enterprise information processing	eip
E-research in the Social Sciences and Humanities	sdo
Crowd-sourced public datasets	pds
Enterprise software development	sdev

Domain-Specific Extensions

Namespaces

Table 4.2 lists the standard prefixes used for each vocabulary. All have been checked for clashes with prefix.cc. Each prefix has been registered as a persistent URL (PURL) with purl.org or the W3C community persistent name service. These namespaces will be maintained by TCD servers.

Enterprise Information Processing

The purpose of the Enterprise Information Processing Domain-specific Meta-Model (EIPDM) is to provide a set of concrete entities, agents, activities, and roles to represent the data engineering process. It is based on the general DLO. The initial information gathered to build the domain-specific enterprise information processing metamodel is based on the JURION use case. The JURION use case includes both processes for data and software development and therefore uses the DLO and the SLO.

As the JURION use case includes the both processes of data (D) and software development (S), the model information are marked with their respective process type.

The actors identified in JURION are listed in Table 4.3. The entities identified in JURION are listed in Table 4.4. The activities are listed in Table 4.5.

For functionalities, we have a number of existing models for different kinds of documents. Depending on the document type, there is different mandatory metadata and additional information.

e-research in the Social Sciences and Humanities

The purpose of the ALIGNED E-research in the Social Sciences and Humanities domain-specific metamodel is to provide a set of concrete entities, agents, activities, and roles to represent the specific data engineering process for e-research in the social sciences and humanities. It is based on the Seshat use case within ALIGNED. It specialises the ALIGNED generic DLO and imports the W3C PROV ontology.

Table 4.3 JURION actors

Actor	Description
CMS Expert	Responsible for the technical correctness of process and data
Content Architect	Responsible for the overall process and schemas
Legal Domain Expert	Responsible for ensuring that legal data are correct
Legal Editor	Responsible for editing legal information
Product Owner	Wants the best possible product
Schema Expert	Responsible for executing and documenting schema changes
Software Developer	Review requirements, suggest possible solutions, estimate cost of certain features and bugfixing actions and implement them.
Software Manager	Coordinates all software development teams and projects
Software Testers	Perform manual testing, issue and observe automated test runs
Software Analyst	Studies the application domain and defines requirements based on his experience the software on the one hand, and the domain and customers on the other hand
Customers	Partners and testers

Table 4.4 JURION entities

Entity	Description
Schema Changes	Schema changes are done at regular intervals
Test Cases	Data tests
Text files	In XML, data in Ontowiki, databases – specific with constraints
Controlled Vocabularies	Several controlled vocabularies are maintained in PoolParty
Data Sources	External data sources
Testing Suites	Java unit tests, Jenkins, Performance Tests, Integration Test, Sonarqube
Source Code	Git and SVN repositories
Server Infrastructure	Servers that support the development process
Data/Software Requirements Documents	Mostly unstructured and free-text description of new features

This model adds support for specific external data sources for datasets like wikis, Web pages, and academic paper repositories. It adds new entities to represent candidate data for inclusion in a dataset, reports of historical events and historical interpretations created by domain experts. It extends the set of data life cycle processes to include data curation activities such as data collection and data publishing. Finally, new roles are defined for data consumer, processor and producer tools that help maintain semi-automated data curation pipelines or workflows.

Table 4.5 JURION activities

Activity	Description
Specify and model data	e.g., definition of base URI and schema mapping
Transform data	Transformation process from XML to RDF format
Integrate/Upload data	Integrate new datasets, entities, and so on
Maintain data	Enrich, delete, change, curate
Link data	Mapping with internal or external sources, link sources
Extract data	Generate test data, configure, test, review, e.g., for classification purposes
Use data	e.g., for visualisations, search, and so on
Quality analysis of data	Checking for consistency, integrity, and so on
Plan Software	Requirements planning, application evolution, data requirements for the data development team
Analyse Software	Requirement validation – requirements changes, version tracking, schema/data-based software evolution analysis
Design Software	Design verification, query design, design evolution via mapping evaluation
Implement Software	Code generation and transformation, application verification
Software maintenance	Schema and instance change impact evaluation, bug classification
Publish data	Converting a dataset to a release

Figure 4.12 illustrates the concepts found in the ontology. A full specification of the model is available online at http://www.essepuntato.it/lode/owlapi/https://w3id.org/sdo.

Seshat Domain Ontologies

The Seshat Domain Ontology Set consists of the following specific ontologies: seshat, xdd, and dacura.

seshat

This ontology describes human societies throughout time. It is used by the Evolution Institute and its partners to describe time-series data collected about all human societies. Figure 4.13 shows the seshat ontology in graphic form.

The most important classes are:

- Polity: A polity is defined as an independent political unit. Kinds of polities range from villages (local communities) through simple and

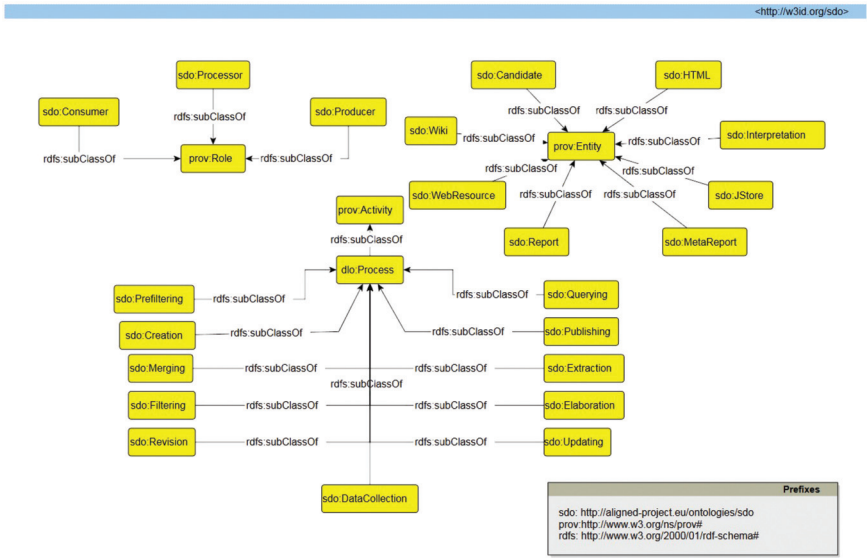


Figure 4.12 The ALIGNED domain-specific ontology for E-research in the Social Sciences and Humanities.

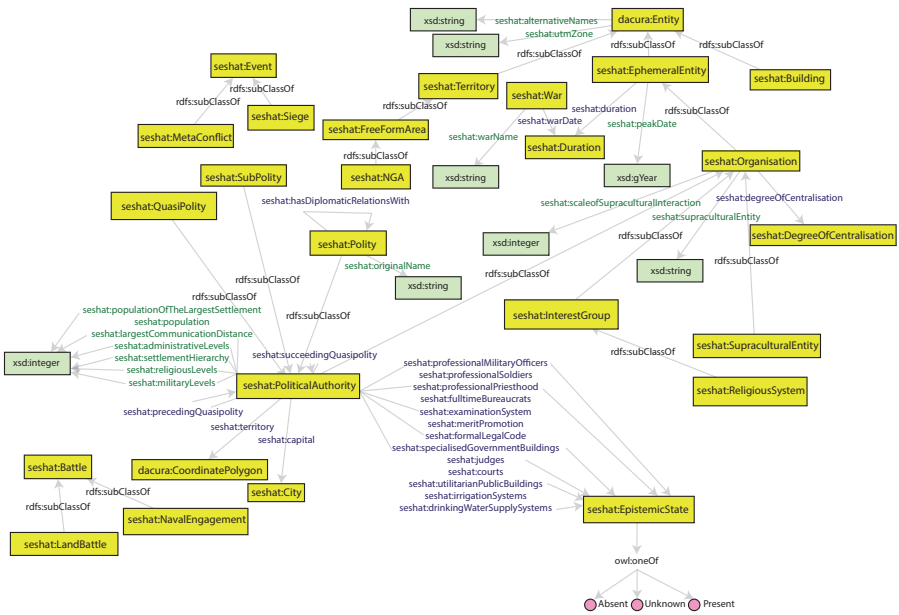


Figure 4.13 The Seshat ontology.

complex chiefdoms to states and empires. A polity can be either centralised or not (e.g., organised as a confederation). What distinguishes a polity from other human groupings and organisations is that it is politically independent of any overarching authority; it possesses sovereignty. Polities are defined spatially by the area enclosed within a boundary on the world map. There may be more than one such areas. Polities are dynamical entities, and thus their geographical extent may change with time. Thus, typically each polity will be defined by a set of multiple boundaries, each for a specified period of time. For prehistoric periods and for geographical areas populated by a multitude of small-scale polities, we use a variant called quasi-polity.

- TemporalEntity: An abstract concept describing anything that must have temporal bounds.
- PointInSpace: This is an abstract class for all points in space.
- Box: Class for boxing datatypes in order to add annotations.

Which have the following properties:

- alternativeName: The name of a seshat Entity. Generally same as the name of the wiki page.
- population: Estimated population of the polity; can change as a result of both adding/losing new territories or by population growth/decline within a region.
- name: The name of a seshat Entity. Generally same as the name of the wiki page.
- peakDate: A property used to define the temporal bounds of a seshatbox:TemporalEntity. For example, corresponds to the Duration for a Polity from the Seshat code book.
- longitude and latitude: In numeric form.
- capitalCityLocation: The latitude and longitude of the capital city.
- type: The xsd datatype of a Box.

xdd

The xdd ontology describes complex datatypes such as polygon, polyline and range types.

```
@prefix xdd: <http://dacura.scss.tcd.ie/ontology/xdd#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

xdd:coordinatePolygon   a rdfs:Datatype ;
  rdfs:label "Coordinate Polygon"@en ;
  rdfs:comment "A closed JSON list of coordinates."@en .
xdd:coordinatePolyline   a rdfs:Datatype ;
```

```
  rdfs:label "Coordinate Polyline"@en ;
  rdfs:comment "A JSON list of coordinates."@en .
xdd:gYearRange   a rdfs:Datatype ;
  rdfs:label "Year"@en ;
  rdfs:comment "Either a year or a range of years."@en .
xdd:integerRange   a rdfs:Datatype ;
  rdfs:label "Integer"@en ;
  rdfs:comment "Either an integer or a range of integers."@en .
xdd:decimalRange   a rdfs:Datatype ;
  rdfs:label "Decimal"@en ;
  rdfs:comment "A number with an arbitrary number of decimal places, or
a numberrange"@en .
```

dacura

The dacura ontology covers all xsd datatypes, rdf and rdfs literal types used in other ontologies on the platform. The dacura ontology can be seen in Figure 4.14.

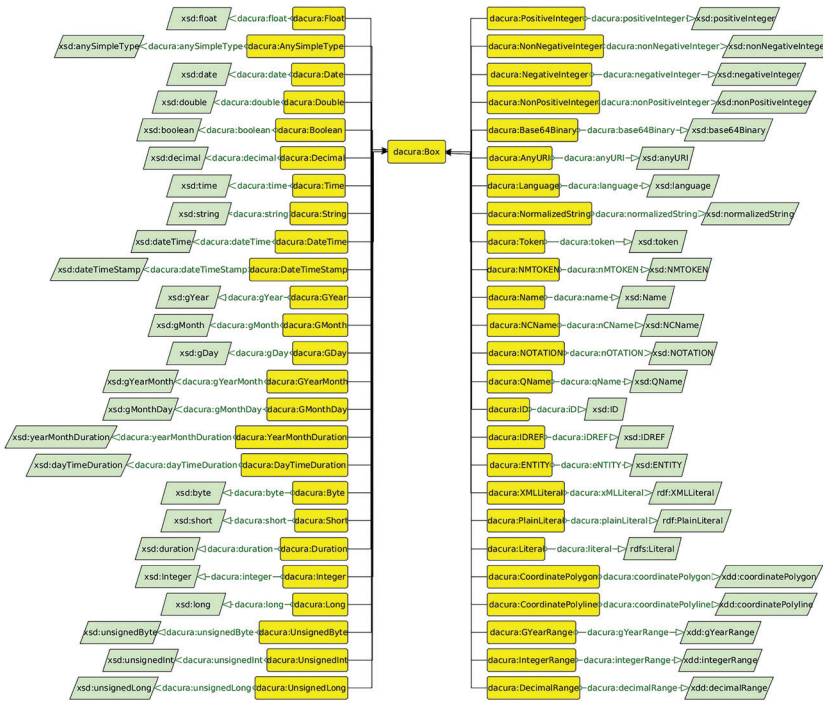


Figure 4.14 The Dacura ontology.

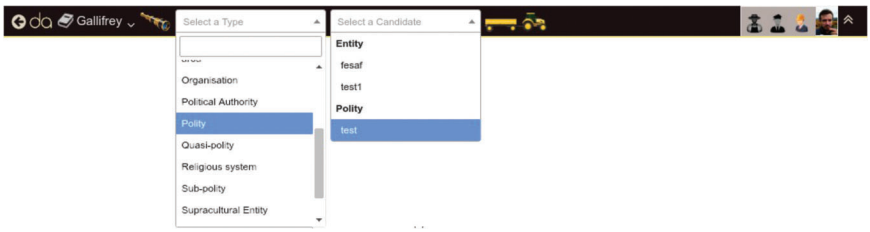


Figure 4.15 Dacura console usage example.

Usage

The set of seshat domain ontologies is designed to be generic in order to keep the usage as broad as possible. An example of the usage of the seshat domain ontology is the dacura console which is available as a browser plugin and enables the user to harvest social sciences and digital humanities data from websites and store them as RDF triples in a knowledge base.

Figure 4.15 shows the dacura console listing types for a candidate from the seshat domain ontology and available candidates for entities and polities. The user selects an existing collection on the left side of the browser bar and depending on her user role, which can be Data Harvester, Expert Annotator or Architect, she can browse and create new candidates of a certain type and edit existing candidates in addition to automatically harvested candidates from a website.

The usage of the dacura and xdd ontologies is best shown on the backend of the dacura platform as represented in Figures 4.16 and 4.17. The screenshots show the creation of a new candidate of the type Polity. Dacura and xdd ontologies are used to describe the used datatypes for properties of a polity (in this example e.g., original name, polity territory, religious levels, polity population, military levels, etc.). Used datatypes can be strings in open text fields as in "Original name", classes in dropdown boxes as in "Has diplomatic relations with", polygons in google maps as in "Polity territory" and many more datatypes.

Crowd-sourced Public Datasets (CSPDO)

This ontology is used to describe the domain-specific extensions to the ALIGNED data life cycle model ontology (DLO) and SLO for crowd-sourced public datasets based on the DBpedia use case within the ALIGNED

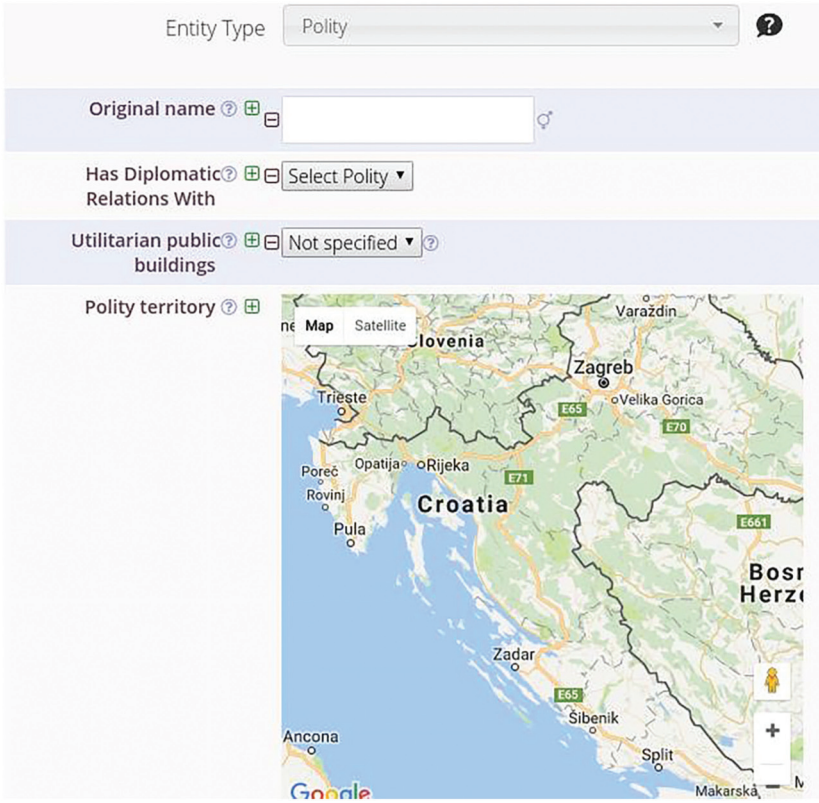


Figure 4.16 New candidate example part 1.

project. Over time, generic features may be migrated to the upper ontology. Figure 4.18 shows the CSPDO ontology.

This ontology is used to describe the domain-specific extensions to the ALIGNED data life cycle model ontology (DLO) and SLO for crowd-sourced public datasets based on the DBpedia use case within the ALIGNED project. Over time, generic features may be migrated to the upper ontology.

This ontology supports extensions needed for DBpedia. Thus, there is a focus on the validation activities. DBpedia is a large-scale extraction project of unstructured and semi-structured data from different Wikipedia language editions to RDF. This extraction is achieved from a modular extraction framework that is customised to handle multilingualism and structural differences

Figure 4.17 New candidate example part 2.

between different Wikipedia language editions. The latest DBpedia release (v. 2016) generated a total of three billion facts from 125 localised versions. As Wikipedia evolves over time, the code should be able to adapt to these changes. However, identifying errors at this data scale becomes very hard and validation workflows must be established that will ensure the quality of the extracted data.

ALIGNED tackled these challenges with data validation and interlink validation tools that communicate their results though the ALIGNED vocabularies.

The actors identified in DBpedia are listed in Table 4.6. The entities identified in DBpedia are listed in Table 4.7. The activities are listed in Table 4.8.

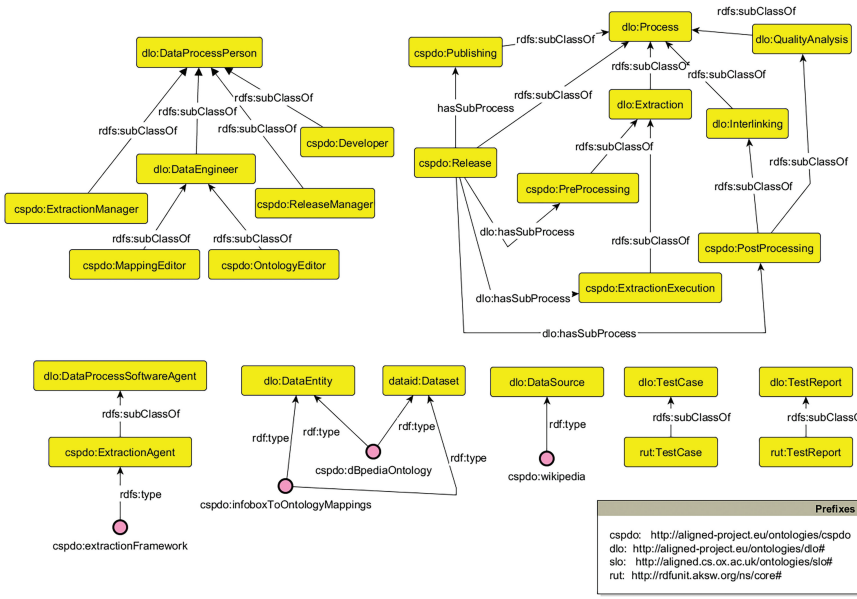


Figure 4.18 The Crowd-sourced Public Datasets ontology.

The DBpedia software and data engineering development process involves the following **actors**:

Table 4.6 DBpedia actors

Actor	Description
Extraction manager	DBpedia team members who run the extraction process for a given DBpedia release
Extraction Agent	Software agents that perform the extraction such as DBpedia live
Mapping editor	Community members who edit the DBpedia mapping wiki
Ontology Editor	DBpedia foundation members that edit the DBpedia ontology
Release manager	DBpedia team members that are responsible for the actions leading to a given release of DBpedia
Developer	DBpedia team members or community who write code for the **extraction framework** or **tools**
User	Users of DBpedia

We can identify the following **entities** (i.e., tools and technologies) that support the DBpedia development workflow:

Table 4.7 DBpedia entities

Entity	Description
Wikipedia	Input source for DBpedia
Extraction Framework	The source code used to extract knowledge from **Wikipedia**
Server	The physical/virtual server where an **extraction agent** is running
DBpedia Ontology	Crowdsourced OWL ontology describing DBpedia concepts and properties
Infobox to Ontology mappings	Crowdsourced mappings between the DBpedia ontology and Wikipedia infoboxes
Dataset static dataset (dump) External dataset Live feed	The output that comes after an extraction manager or release manager runs an extraction agent based on the extraction framework on a **Wikipedia** input. The output can be a static dataset, an external dataset (such as links to other datasets) or the DBpedia Live feed
Tools	Scripts or applications that work on DBpedia data
Issue or support question	New feature or support requests and bug reports are filed as tickets in the **extraction framework** Github issue tracker or reported in the DBpedia-related mailing lists

Actors and Entities are connected by the following **activities**:

Table 4.8 DBpedia activities

Activity	Description
Coding	Involves resolution of **issues/error reports** (i.e., bug fixing, feature development), but also refactoring. Done by **developers**, working on the **extraction framework**.
Release Pre-processing step Extraction Post-processing step Publishing	Releasing a DBpedia **dataset** is a complex procedure that involves a lot of pre-processing steps, the actual extraction, additional postprocessing steps and finally the dataset publishing step. This activity is performed by the **release manager** using the **extraction framework** and DBpedia **tools**
Maintain dbpedia.org	The act of maintaining the information website of DBpedia
Support (mailing lists or bugs)	Acting on a user support or new feature request or tackling a bug report

DBpedia Ontology (DBO)

The structure of the DBpedia knowledge base is maintained by the DBpedia user community. Most importantly, the community creates mappings

from Wikipedia information representation structures to the DBpedia ontology. This ontology unifies different template structures, both within single Wikipedia language editions and across currently 27 different languages. The complete DBpedia ontology can be browsed online at http://mappings.dbpedia.org/server/ontology/classes/.

DBO is used to describe the data that are extracted with the DBpedia information extraction framework.

Usage

Model Mapper tool

The prototype Model Mapper tool (D3.4) uses CSPDO to record interlink validation processing on the DBpedia release candidate. This enables its activities to be shown in the Unified Governance tool (D5.2), and for other data engineering tools to co-ordinate with it in a toolchain. For example, as shown here, for the exchange of which mappings failed the validation test. This allows another tool to take corrective action on these mappings or to present them to a user.

The RDF shows the description of an interlink validation run which identifies the specific tool used for validation, the three datasets consumed (the linkset, DBpedia and Geonames) and the validation report produced (ex:interlink_validation_report_1). The datasets are identified as DataID data-sources and thus could have a large amount of metadata recorded about them. The actor who initiated the interlink validation is recorded and classified as a SysAdmin. The interlink validation report itself identifies two invalid mappings in the mapping set analysed, in the first case both ends of the mapping are incorrect (probably missing from the mapped datasets) and the second mapping error identifies only one mal-formed resource.

```
ex:interlink_val_1 a cspdo:InterlinkValidation ;
     dlo:isSupportedBy ex:interlink_validator ;
     dlo:consumes ex:dbpedia_geonames_interlinks_2015 ;
     dlo:consumes ex:dbpedia_dataset_2015 ;
     dlo:consumes ex:geonames_dataset_20151010 ;
     dlo:produces ex:interlink_validation_report_1 .
ex:dbpedia_dataset_2015 a dlo:DataSource .
ex:geonames_dataset_20151010 a dlo:DataSource .
ex:person_1 a dlo:SystemAdmin ;
     dlo:initiates ex:interlink_val_1 .
```

```
ex:model_mapper a dlo:DataSoftwareAgent ;
     dlo:supports ex:interlink_val_1 .
ex:interlink_validation_report_1 a cspdo:InterlinkValidationReport ;
     prov:wasGeneratedBy ex:interlink_val_1 ;
     ex:invalidMapping1 [ ex:mapId ex:mapping_1 ;
     ex:invalidResource <resource_1> ;
     ex:invalidResource <resource_2> ] ;
     ex:invalidMapping2 [ ex:mapId ex:mapping_2 ;
     ex:invalidResource <resource_3> ] ;
     prov:generatedAtTime ''20151010''^^xsd:date .
```

DBpedia release description

Since 2015, DBpedia releases are described with the DataID ontology. This created the opportunity for application on top of the machine readable dataset metadata. These DataID descriptions are used to automatically generate the DBpedia release download page as well as automate the creation of a triple store loaded with the release data using the Docker container technology.[15]

DBpedia workflow description (planned)

As a future work, we plan to integrate DataID, DLO and PROV to describe DBpedia extraction workflows and keep track of origin and pre-processing steps of each dataset.

4.6.5 Enterprise Software Development (DIOPP)

The aim of the ontology is to integrate the datasets generated through requirements specification and the issues raised during their implementation. This ontology covers the mappings defined between the PoolParty conceptualisation and the DIO ontology. The mappings are further supported by the figures illustrated here. An example illustrating the mapping can be found here.

In the following, we describe PoolParty's requirements for the ALIGNED domain-specific metamodel for enterprise software development.

The actors identified in PoolParty are listed in Table 4.9. The entities identified in PoolParty are listed in Table 4.10. The activities are listed in Table 4.11.

[15] https://github.com/dbpedia/Dockerized-DBpedia

For the software life cycle and design intent, the development process involves the following **actors**:

Table 4.9 PoolParty actors

Actor	Description
Project Manager	Responsible for resource planning
Requirements Editor	Specifies requirements for a specific feature in a way that it fits to the application's design (functional and UI)
Product Owner	Knows the market and customers, identifies new features, (informally) specifies requirements, continuous and final inspection of new features
Consultant	Knows the customers and their needs, provide support for existing and training for new customers. May act as Project Managers, Requirements Editors, and Testers
Developer	Review requirements, suggest possible solutions, estimate cost of certain features and bugfixing actions and implement them.
Tester	Perform manual testing, issue and observe automated test runs
Customer	Partners, Integrators

We can identify the following **entities** (i.e., tools and technologies) that support the PoolParty development workflow:

Table 4.10 PoolParty entities

Entity	Description
Issue Ticket	New feature requests and bug reports are filed as tickets in Atlassian Jira. They have assigned, e.g., a creator (a **consultant** in most cases), an actor responsible for resolution (a **developer** in most cases), a **cost** estimation (in days), and **version** information (e.g., which version it occurred) and other metadata like description, dates, comments. Can be organised in Epics, Stories and Issues. Each of these may cover a Requirements Document (see below).
Requirements Documents	Are written using Atlassian Confluence Wiki. Mostly unstructured and freetext description of new features. Are proofread by **product owner** and **developers**.
Source Code	Git and SVN repositories
Server Infrastructure	Servers that support the development process, e.g., **testing** PoolParty or performing demos, scheduled builds for continuous integration, hosting developer chat/continuous integration notifications
Testing Suite	Java unit tests, Selenium Web Browser automation tests, API tests, operated by **testers**
Communication Resource	Skype, GotoMeeting, Chat clients, email

Actors and Entities are connected by the following **activities**:

Table 4.11 PoolParty activities

Activity	Description
Resource planning	Meetings where project managers and product owners decide (based on the issue ticket cost estimations) what features and bug requests will be scheduled for a **sprint** with what **priority**
Sprint	Certain period of time during which a specified set of **issue tickets** should be resolved
Coding	Involves resolution of **issue tickets** (i.e., bug fixing, feature development) but also refactoring. Done by **developers**, creating **source code**.
Staging	Preparing a release version of the software, i.e., creating installation packages and installing them at customer **server infrastructure**
Requirements writing	The activity of creating requirement documents and issue tickets
Communication to customers	Informal **communication** between **Consultant** and **Customer** for initiating requirements writing

4.6.6 Unified Governance Domain Ontologies

The motivation for our work was the current setup at SWC, where Atlassian Confluence wiki-like team collaboration software is used to support requirements engineering, feature specification and discussion, providing documentation of research projects and publishing of technical information. Atlassian JIRA is a ticket system used for issue and change tracking, organising ideas from team members as well as collecting from customers. These loosely coupled tools form the basis for a requirements engineering system.

Following the agile methodology of software development, the data are recorded in Confluence under headings such as “Requirements”, “Goal”,“User Story”, “Epic” and “Stakeholders”. Additional fields such as “Precondition”, “Detailed description”, “Acceptance criteria & Test scenario” are included to provide further context to the requirements. A single field, “Comment” captures the opinions/discussion carried out by human agents. The JIRA interface is used without any major modification.

SWC collects the requirements for each version of PPT in the PoolParty development space. Requirements are then linked to pages containing epics and user stories. Most of these pages are structured based on standard

templates defined by SWC. The outputs from these template-based pages are largely document-centric and require extensive human intervention to synthesise and synchronise them with PoolParty development tasks.

By using DIO, DIOPP and bespoke mappings to annotate and provide metadata to the content extracted from Confluence and JIRA, SWC is able to create merged repositories of requirements, customer feedback, bug reports and project documentation thereby consolidating PoolParty experiences, customer ideas and market needs in order to integrate them into products. This is a key factor for successful development of SWC products and for raising customer satisfaction and enterprise agility. Questions asked by customers will flow faster into the requirement engineering system. The process will help to generate concise reports on distributed business objects and entities relevant for the development processes, and to coordinate the data management and development workflows required to deliver new versions of the evolving PoolParty product. The serendipitous mining of design intents from requirements and issues will therefore have a significant impact on the full life cycle of PoolParty products from requirements through to development and maintenance.

4.6.7 Semantic Booster and Model Catalogue Domain Ontology

4.6.7.1 Model catalogue

The Model Catalogue can be used to document models and metamodels – adding descriptions and descriptive metadata to concepts and relationships. Search and comparison tools allow modellers and data engineers to understand concepts in the model and better understand the underlying data. In the ALIGNED project, we have been building a repository of the metamodels and domain-specific models for external users.

The catalogue can also be used to provide, and reason about, links between concepts in different domains. For example, showing how a software model reuses and extends concepts from the data life cycle (DLO) will help the users of data understand how data can be linked and compared.

The Model Catalogue may be used as a development platform for metamodels or domain-specific models – a collaborative editing platform enables the easy development of new versions of models, permitting discourse and iteration, controlling versioning and user access. The catalogue is tightly integrated with pipelines for MDE: enabling export of software components – alternate representations, sources or configurations for data entry such as XForms, data transfer such as XSD and XML, or data storage, such as relational database schemas or Booster specifications.

The catalogue can also use the ALIGNED ontologies to capture metadata about the models themselves – for example using PROV to capture provenance information about a dataset, or DIO to capture design intent behind software modelling decisions. The metamodel for the catalogue itself – that constrains the way that models are represented –is being extended in the next phase of the ALIGNED project to incorporate more concepts from the generic ALIGNED models.

4.6.7.2 Booster

As a MDE tool, Booster uses domain-specific models to build systems. Booster can aid the development of tools that build interoperable datasets by extending the ALIGNED metamodels. For example, an abstract model of DLV in Booster may be extended and specialised in a domain-specific model. This will ensure that any data captured and stored in the Booster-generated system will be semantically interoperable with data collected in other systems based on the DLV ontology. By understanding how these domain-specific models extend or instantiate parts of the ALIGNED metamodels, the tool can be configured to specialise the software. For example, data captured and stored in Booster might be automatically linked to public external datasets corresponding to compatible ontologies.

Booster has its own metamodel: instances of which are Booster specifications. The Booster metamodel may be linked to concepts in the ALIGNED ontologies – in particular "design decisions", and parts of a "software life cycle". Currently the textual notation for Booster does not easily support the linking of these concepts, but the design and development of a more advanced metamodel for Booster is underway, allowing explicit links to external ontologies, with support for maintaining and using these links within the generated software components.

4.6.8 PROV[16]

The provenance of digital objects represents their origins. PROV is a specification to express provenance records, which contain descriptions of the entities and activities involved in producing and delivering or otherwise influencing a given object. Provenance can be used for many purposes, such as understanding how data were collected so it can be meaningfully used, determining ownership and rights over an object, making judgements

[16]This section contains material derived from "PROV-Overview An Overview of the PROV Family of Documents", https://www.w3.org/TR/prov-overview/ © 2013 W3C.

about information to determine whether to trust it, verifying that the process and steps used to obtain a result complies with given requirements, and reproducing how something was generated.

As a specification for provenance, PROV accommodates all those different uses of provenance. Different people may have different perspectives on provenance, and as a result, different types of information might be captured in provenance records.

- One perspective might focus on agent-centred provenance, that is, what people or organisations were involved in generating or manipulating the information in question. For example, in the provenance of a picture in a news article we might capture the photographer who took it, the person that edited it, and the newspaper that published it.
- A second perspective might focus on object-centred provenance, by tracing the origins of portions of a document to other documents. An example is having a Web page that was assembled from content from a news article, quotes of interviews with experts, and a chart that plots data from a government agency.
- A third perspective one might take is on process-centred provenance, capturing the actions and steps taken to generate the information in question. For example, a chart may have been generated by invoking a service to retrieve data from a database, then extracting certain statistics from the data using some statistics package, and finally processing these results with a graphing tool.

The goal of PROV is to enable the wide publication and interchange of provenance on the Web and other information systems. PROV enables one to represent and interchange provenance information using widely available formats such as RDF and XML. In addition, it provides definitions for accessing provenance information, validating it, and mapping to Dublin Core.

The design of PROV stems from the recommendations of the Provenance Incubator Group which performed an extensive information gathering process including use case cataloging, requirements elicitation and a literature survey. From this process, the following eight recommendations were made:

1. the core concepts of identifying an object, attributing the object to person or entity, and representing processing steps;
2. accessing provenance-related information expressed in other standards;
3. accessing provenance;
4. the provenance of provenance;
5. reproducibility;

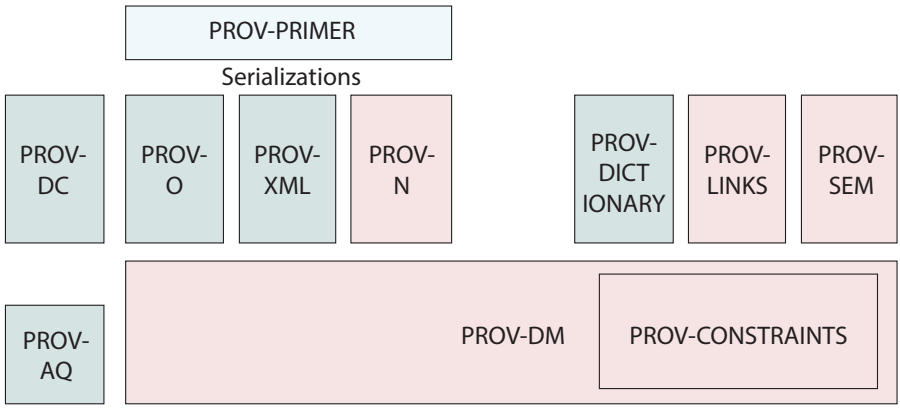


Figure 4.19 The Organisation of PROV.

6. versioning;
7. representing procedures;
8. representing derivation.

Figure 4.19 shows the organisation of PROV and how the documents (roughly) depend on each other. At its core is a conceptual data model (PROV-DM), which defines a common vocabulary used to describe provenance. This is instantiated by various serialisations. These serialisations are used by implementations to interchange provenance. To help developers and users express valid provenance, a set of constraints (PROV-Constraints) are defined, which can be used to implement provenance validators. This is complimented by a formal semantics (PROV-SEM). Finally, to further support the interchange of provenance, additional specifications are provided for protocols to locate and access provenance (PROV-AQ), connect bundles of provenance descriptions (PROV-Links), represent dictionary style collections (PROV-Dictionary) and define how to interoperate with the widely used Dublin Core vocabulary (PROV-DC).

4.6.9 SKOS[17]

The SKOS is a data-sharing standard, bridging several different fields of knowledge, technology and practice. In the library and information sciences,

[17]This section contains material derived from "SKOS Simple Knowledge Organization System Reference", https://www.w3.org/TR/skos-reference/ © 2009 W3C.

a long and distinguished heritage is devoted to developing tools for organising large collections of objects such as books or museum artefacts. These tools are known generally as "knowledge organization systems" (KOS) or sometimes as "controlled structured vocabularies". Several similar yet distinct traditions have emerged over time, each supported by a community of practice and set of agreed standards. Different families of knowledge organisation systems, including thesauri, classification schemes, subject heading systems, and taxonomies are widely recognised and applied in both modern and traditional information systems. In practice, it can be hard to draw an absolute distinction between thesauri and classification schemes or taxonomies, although some properties can be used to broadly characterise these different families. The important point for SKOS is that, in addition to their unique features, each of these families shares much in common and can often be used in similar ways. However, there is currently no widely deployed standard for representing these knowledge organisation systems as data and exchanging them between computer systems.

The W3C's Semantic Web Activity has stimulated a new field of integrative research and technology development, at the boundaries between database systems, formal logic and the World Wide Web. This work has led to the development of foundational standards for the Semantic Web. The RDF provides a common data abstraction and syntax for the Web. The RDF Vocabulary Description language (RDFS) and the OWL together provide a common data modelling (schema) language for data in the Web. The SPARQL Query Language and Protocol provide a standard means for interacting with data in the Web.

These technologies are being applied across diverse applications because many applications require a common framework for publishing, sharing, exchanging and integrating ("joining up") data from different sources. The ability to link data from different sources is motivating many projects, as different communities seek to exploit the hidden value in data previously spread across isolated sources.

The SKOS therefore aims to provide a bridge between different communities of practice within the library and information sciences involved in the design and application of knowledge organisation systems. In addition, SKOS aims to provide a bridge between these communities and the Semantic Web, by transferring existing models of knowledge organisation to the Semantic Web technology context, and by providing a low-cost migration path for porting existing knowledge organisation systems to RDF.

The SKOS is a common data model for knowledge organisation systems such as thesauri, classification schemes, subject heading systems and taxonomies. Using SKOS, a knowledge organisation system can be expressed as machine-readable data. It can then be exchanged between computer applications and published in a machine-readable format in the Web. The SKOS data model is formally defined as an OWL Full ontology. SKOS data are expressed as RDF triples and may be encoded using any concrete RDF syntax (such as RDF/XML or Turtle). The SKOS data model views a knowledge organisation system as a concept scheme comprising a set of concepts. These SKOS concept schemes and SKOS concepts are identified by URIs, enabling anyone to refer to them unambiguously from any context, and making them a part of the World Wide Web. SKOS concepts can be labelled with any number of strings, in any given natural language. One of these labels in any given language can be indicated as the preferred label for that language, and the others as alternative labels.

4.6.10 OWL[18]

The OWL is a language for defining and instantiating Web ontologies. Ontology is a term borrowed from philosophy that refers to the science of describing the kinds of entities in the world and how they are related. An OWL ontology may include descriptions of classes, properties, and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e., facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms.

One question that comes up when describing yet another XML/Web standard is "What does this buy me that XML and XML Schema don't?" There are two answers to this question.

- An ontology differs from an XML schema in that it is a knowledge representation, not a message format. Most industry-based Web standards consist of a combination of message formats and protocol specifications. These formats have been given an operational semantics, such as, "Upon receipt of this PurchaseOrder message, transfer Amount dollars from AccountFrom to AccountTo and ship Product". But the specification is

[18]This section contains material derived from "OWL Web Ontology Language Overview", https://www.w3.org/TR/owl-features/ © 2004 W3C.

not designed to support reasoning outside the transaction context. For example, we will not in general have a mechanism to conclude that because the Product is a type of Chardonnay it must also be a white wine.

- One advantage of OWL ontologies will be the availability of tools that can reason about them. Tools will provide generic support that is not specific to the particular subject domain, which would be the case if one were to build a system to reason about a specific industry-standard XML schema. Building a sound and useful reasoning system is not a simple effort. Constructing an ontology is much more tractable. It is our expectation that many groups will embark on ontology construction. They will benefit from third-party tools based on the formal properties of the OWL language, tools that will deliver an assortment of capabilities that most organisations would be hard pressed to duplicate.

The OWL language provides three increasingly expressive sublanguages designed for use by specific communities of implementers and users.

OWL Lite supports those users primarily needing a classification hierarchy and simple constraint features. For example, while OWL Lite supports cardinality constraints, it only permits cardinality values of 0 or 1. It should be simpler to provide tool support for OWL Lite than its more expressive relatives, and provide a quick migration path for thesauri and other taxonomies.

OWL DL supports those users who want the maximum expressiveness without losing computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time) of reasoning systems. OWL DL includes all OWL language constructs with restrictions such as type separation (a class cannot also be an individual or property, a property cannot also be an individual or class). OWL DL is so named due to its correspondence with description logics, a field of research that has studied a particular decidable fragment of FOL. OWL DL was designed to support the existing Description Logic business segment and has desirable computational properties for reasoning systems.

OWL Full is meant for users who want maximum expressiveness and the syntactic freedom of RDF with no computational guarantees. For example, in OWL Full, a class can be treated simultaneously as a collection of individuals and as an individual in its own right. Another significant difference from OWL DL is that a owl:DatatypeProperty can be marked as an owl:InverseFunctionalProperty. OWL Full allows an ontology to augment the meaning of the pre-defined (RDF or OWL) vocabulary. It is unlikely that any reasoning software will be able to support every feature of OWL Full.

4.6.11 RDFS[19]

The RDF is a framework for expressing information about resources. Resources can be anything, including documents, people, physical objects, and abstract concepts.

RDF is intended for situations in which information on the Web needs to be processed by applications, rather than being only displayed to people. RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parsers and processing tools. The ability to exchange information between different applications means that the information may be made available to applications other than those for which it was originally created.

In particular, RDF can be used to publish and interlink data on the Web. For example, retrieving http://www.example.org/bob#me could provide data about Bob, including the fact that he knows Alice, as identified by her IRI (International Resource Identifier). Retrieving Alice's IRI could then provide more data about her, including links to other datasets for her friends, interests, and so on. A person or an automated process can then follow such links and aggregate data about these various things. Such uses of RDF are often qualified as Linked Data.

Triples

RDF allows us to make statements about resources. The format of these statements is simple. A statement always has the following structure:

<subject> <predicate> <object>

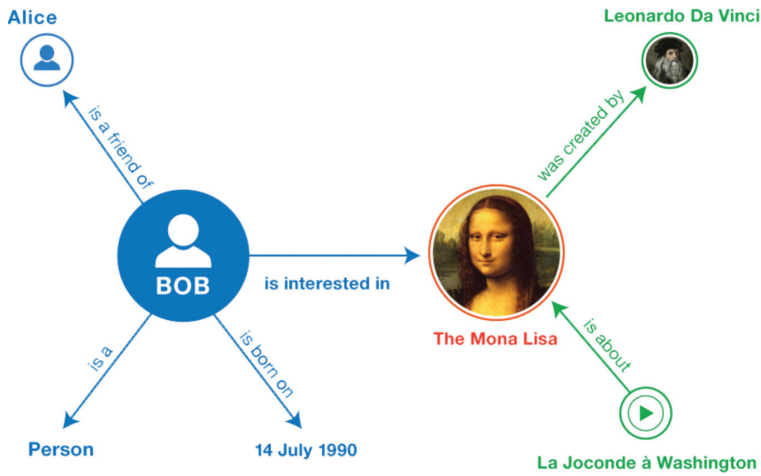
An RDF statement expresses a relationship between two resources. The subject and the object represent the two resources being related; the predicate represents the nature of their relationship. The relationship is phrased in a directional way (from subject to object) and is called in RDF a property. Because RDF statements consist of three elements, they are called triples.

Here are examples of RDF triples (informally expressed in pseudocode):

Example 1: Sample triples (informal)

<Bob> <is a> <person>.
<Bob> <is a friend of> <Alice>.

[19]This section contains material derived from "RDF Schema 1.1", https://www.w3.org/TR/rdf-schema/ © 2004–2014 W3C.



<Bob> <is born on> <the 4th of July 1990>.
<Bob> <is interested in> <the Mona Lisa>.
<the Mona Lisa> <was created by> <Leonardo da Vinci>.
<the video “La Joconde Washington”> <is about> <the Mona Lisa>

The same resource is often referenced in multiple triples. In the example above, Bob is the subject of four triples, and the Mona Lisa is the subject of one and the object of two triples. This ability to have the same resource be in the subject position of one triple and the object position of another makes it possible to find connections between triples, which is an important part of RDF’s power.

We can visualise triples as a connected graph. Graphs consist of nodes and arcs. The subjects and objects of the triples make up the nodes in the graph; the predicates form the arcs. Figure 4.20 shows the graph resulting from the sample triples.

Once you have a graph like this you can use SPARQL to query for e.g., people interested in paintings by Leonardo da Vinci.

The RDF Data Model is described in this section in the form of an “abstract syntax”, i.e., a data model that is independent of a particular concrete syntax (the syntax used to represent triples stored in text files). Different concrete syntaxes may produce exactly the same graph from the perspective of the abstract syntax. The semantics of RDF graphs are defined in terms of this abstract syntax.

4.6.12 RDF[20]

The RDF is a language for representing information about resources in the World Wide Web. It is particularly intended for representing metadata about Web resources, such as the title, author, and modification date of a web page, copyright and licensing information about a Web document, or the availability schedule for some shared resource. However, by generalising the concept of a "Web resource", RDF can also be used to represent information about things that can be identified on the Web, even when they cannot be directly retrieved on the Web. Examples include information about items available from online shopping facilities (e.g., information about specifications, prices, and availability), or the description of a Web user's preferences for information delivery.

RDF is intended for situations in which this information needs to be processed by applications, rather than being only displayed to people. RDF provides a common framework for expressing this information so it can be exchanged between applications without loss of meaning. Since it is a common framework, application designers can leverage the availability of common RDF parsers and processing tools. The ability to exchange information between different applications means that the information may be made available to applications other than those for which it was originally created.

RDF is based on the idea of identifying things using Web identifiers (called Uniform Resource Identifiers, or URIs), and describing resources in terms of simple properties and property values. This enables RDF to represent simple statements about resources as a graph of nodes and arcs representing the resources, and their properties and values. To make this discussion somewhat more concrete as soon as possible, the group of statements "there is a Person identified by http://www.w3.org/People/EM/contact#me, whose name is Eric Miller, whose email address is em@w3.org, and whose title is Dr." could be represented as the RDF graph below:

This illustrates that RDF uses URIs to identify:

- individuals, e.g., Eric Miller, identified by http://www.w3.org/People/EM/contact#me

[20]This section contains material derived from "RDF Primer", https://www.w3.org/TR/rdf-primer/ © 2004 W3C.

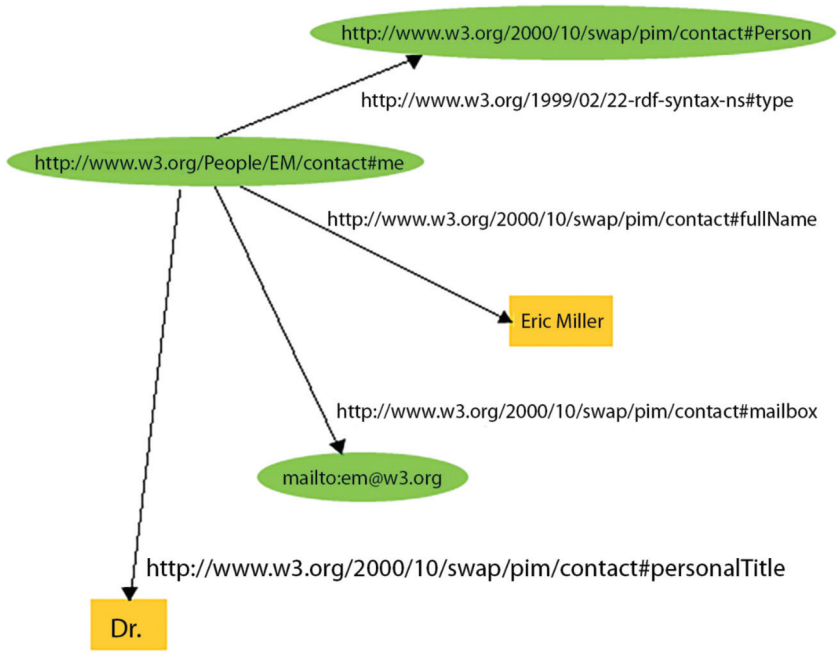


Figure 4.20 An RDF Graph Describing Eric Miller.

- kinds of things, e.g., Person, identified by http://www.w3.org/2000/10/swap/pim/contact#Person
- properties of those things, e.g., mailbox, identified by http://www.w3.org/2000/10/swap/pim/contact#mailbox
- values of those properties, e.g., mailto:em@w3.org as the value of the mailbox property (RDF also uses character strings such as "Eric Miller", and values from other datatypes such as integers and dates, as the values of properties)

RDF also provides an XML-based syntax (called RDF/XML) for recording and exchanging these graphs. Example 2 is a small chunk of RDF in RDF/XML corresponding to the graph in Figure 4.20:

Example 2: RDF/XML Describing Eric Miller

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">

<contact:Person rdf:about="http://www.w3.org/People/EM/contact\#me">
<contact:fullName>Eric Miller</contact:fullName>
```

```
<contact:mailbox rdf:resource="mailto:em@w3.org"/>
<contact:personalTitle>Dr.</contact:personalTitle>
</contact:Person>

</rdf:RDF>
```

Note that this RDF/XML also contains URIs, as well as properties like mailbox and fullName (in an abbreviated form), and their respective values em@w3.org, and Eric Miller.

Like HTML, this RDF/XML is machine processable and, using URIs, can link pieces of information across the Web. However, unlike conventional hypertext, RDF URIs can refer to any identifiable thing, including things that may not be directly retrievable on the Web (such as the person Eric Miller). The result is that in addition to describing such things as Web pages, RDF can also describe cars, businesses, people, news events, and so on. In addition, RDF properties themselves have URIs, to precisely identify the relationships that exist between the linked items.