Date: Monday 18th May, 2015

The Distributed Ontology, Model, and Specification Language (DOL)

Version 1.0

OMG Document Number: ad/2015-05-03
Normative reference:
Machine readable files(s): ad/2015-05-05, ad/2015-05-05, ..., ad/2015-05-12, ad/2015-05-14
Normative:
Non-normative:

COMPLIANCE

OMG's Issue Reporting Procedure

All OMG specifications are subject to continuous review and improvement. As part of this process we encourage readers to report any ambiguities, inconsistencies, or inaccuracies they may find by completing the Issue Reporting Form listed on the main web page `http://www.omg.org`, under Documents, Report a Bug/Issue (`http://www.omg.org/technology/agreement.htm`).

Table of Contents

Preface

OMG

Founded in 1989, the Object Management Group, Inc. (OMG) is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable, and reusable enterprise applications in distributed, heterogeneous environments. Membership includes Information Technology vendors, end users, government agencies, and academia.

OMG member companies write, adopt, and maintain its specifications following a mature, open process. OMG's specifications implement the Model Driven Architecture® (MDA®), maximizing ROI through a full-lifecycle approach to enterprise integration that covers multiple operating systems, programming languages, middleware and networking infrastructures, and software development environments. OMG's specifications include: UML® (Unified Modeling Language™); CORBA® (Common Object Request Broker Architecture); CWM™ (Common Warehouse Metamodel); and industry-specific standards for dozens of vertical markets.

More information on the OMG is available at http://www.omg.org/.

OMG Specifications

As noted, OMG specifications address middleware, modeling and vertical domain frameworks. All OMG Specifications are available from the OMG website at:

`http://www.omg.org/spec`

Specifications are organized by the following categories:

- Business Modeling Specifications
- Middleware Specifications
  - CORBA/IIOP
  - Data Distribution Services
  - Specialized CORBA
- IDL/Language Mapping Specifications
- Modeling and Metadata Specifications
  - UML, MOF, CWM, XMI
  - UML Profile
- Modernization Specifications
- Platform Independent Model (PIM), Platform Specific Model (PSM), Interface Specifications
  - CORBAServices
  - CORBAFacilities

- OMG Domain Specifications
- CORBA Embedded Intelligence Specifications
- CORBA Security Specifications

All of OMG's formal specifications may be downloaded without charge from our website. (Products implementing OMG specifications are available from individual suppliers.) Copies of specifications, available in PostScript and PDF format, may be obtained from the Specifications Catalog cited above or by contacting the Object Management Group, Inc. at:

OMG Headquarters
140 Kendrick Street
Building A, Suite 300
Needham, MA 02494
USA
Tel: +1-781-444-0404
Fax: +1-781-444-0320
Email: `pubs@omg.org`

Certain OMG specifications are also available as ISO standards. Please consult `http://www.iso.org`.

Typographical Conventions

The type styles shown below are used in this document to distinguish programming statements from ordinary English. However, these conventions are not used in tables or section headings where no distinction is necessary.

Times/Times New Roman - 10 pt.: Standard body text

Helvetica/Arial - 10 pt. Bold: OMG Interface Definition Language (OMG IDL) and syntax elements.

`Courier - 10 pt. Bold:` Programming language elements.

Helvetica/Arial - 10 pt.: Exceptions

NOTE: Italic text represents names defined in the specification or the name of a document, specification, or other publication.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification to `http://www.omg.org/report_issue.htm`.

0. Submission-Specific Material

0.1. Submission Preface

Fraunhofer FOKUS, MITRE, and Thematix Partners LLC are pleased to submit this joint proposal in response to the Ontology, Model and Specification Integration and Interoperability (OntoIOp) RFP (OMG document ad/2013-12-02). The submitter contacts for this submission are:

- Fraunhofer FOKUS, Andreas Hoffmann, andreas.hoffmann@fokus.fraunhofer.de
- MITRE, Leo Obrst, lobrst@mitre.org
- Thematix Partners LLC, Elisa Kendall, ekendall@thematix.com

Clause 0 of this document contains information specific to the OMG submission process and is not part of the proposed specification. The proposed specification starts with Clause 1 "Scope".

0.2. Mandatory Requirements

ID	RFP requirement	How this proposal addresses requirement
6.5.1(a)	Proposals shall provide a specification of a metalanguage for relationships between the components of logically heterogeneous OMS, particularly, given a language translation from a language L1 to another language L2, the application of the language translation to an OMS that is written in the language L1.	DOL provides the required translation construct using syntax `O with translation t`, see 9.4 and 10.2.3. Moreover, DOL provides heterogeneous interpretations between OMS, see 9.5 and 10.2.4.
6.5.1(b)	Proposals shall provide a specification of a metalanguage for the union of OMS written in different languages, which implicitly involves the application of suitable default translations in order to reach a common target language.	The syntax for unions is `O1 and O2`, see 9.4 and 10.2.3. Default translations are discussed in 9.4, and DOL's notion of heterogeneous logical environment explicitly specifies default translations, see 11.2.
6.5.1(c)	Proposals shall provide a specification of a metalanguage for importation in modular OMS.	DOL allows the import of OMS by their IRI, see 9.4 and 10.2.3.

Continued on next page

Table 0.1 – *Continued from previous page*

ID	**RFP requirement**	**How this proposal addresses requirement**
6.5.1(d)	Proposals shall provide a specification of a metalanguage for relationships between OMS and their extracted modules e.g. the whole theory is a conservative extension of the module.	DOL provides such a construct with syntax `module m : o1 of o2 for sig`, see 9.5 and 10.2.4.
6.5.1(e)	Proposals shall provide a specification of a metalanguage for relationships between OMS and their approximation in less expressive languages such that the approximation is logically implied by the original theory, where the approximation generally has to be maximal in some suitable sense.	DOL provides such a construct with syntax `o keep logic`, see 9.4 and 10.2.3.
6.5.1(f)	Proposals shall provide a specification of a metalanguage for links such as imports, interpretations, refinements, and alignments between OMS/modules.	DOL covers several metalogical relationships, namely entailments, interpretations, equivalences, refinements, alignments and module relations, see 9.5 and 10.2.4.
6.5.1(g)	Proposals shall provide a specification of a metalanguage for combination of OMS along links.	DOL provides such a construct with syntax `combine n`, where `n` is a network of OMS and mappings (links), see 9.4 and 10.2.3.
6.5.2(a)	The constructs of the metalanguage shall be applicable to different logics.	The semantics of DOL is based on a heterogeneous logical environment, which can contain arbitrary logics, see 11.2.
6.5.2(b)	The metalanguage shall neither be restricted to OMS in a specific domain, nor to OMS represented in a specific logical language.	The semantics of DOL is based on a heterogeneous logical environment, which can contain arbitrary logics, see 11.2.
6.5.2(c)	The metalanguage shall not replace the object language constructs of the conforming logical languages.	The syntax of a `NativeOMS` is left unspecified in this standard. Rather, here this standard relies on other standards and language definitions. See 9.4 and 10.2.3.
6.5.2(d)	The metalanguage shall provide syntactic constructs for (i) structuring OMS regardless of the logic in which their sentences are formalized and (ii) basic and structured OMS and facilities to identify them in a globally unique way.	The structuring constructs for OMS in 9.4 and 10.2.3 can be used for any logic, see the semantics in 11.2. DOL uses IRIs for referencing both basic and structured OMS, see 9.7.1.

Continued on next page

Table 0.1 – *Continued from previous page*

ID	**RFP requirement**	**How this proposal addresses requirement**
6.5.3(a)	An abstract syntax specified as an SMOF compliant meta model.	Currently, the abstract syntax is specified using EBNF, see clause 9. The SMOF meta model is given in annex M.
6.5.3(b)	A human-readable lexical concrete syntax in EBNF and serialization in XML, for the latter XMI shall be used.	The concrete syntax (in EBNF) is specified in clause 10. The XMI representation is automatically derived from the SMOF meta model.
6.5.3(c)	Complete round-trip mappings from the human-readable concrete syntax to the abstract syntax and vice versa.	Both abstract syntax (clause 9) and concrete syntax (clause 10) use the same non-terminal symbols in their EBNF grammar; this makes a round-trip mapping between both straightforward. Moreover, the round-trip mapping has been implemented in form of a parser and a printer as part of the heterogeneous tool set (see `http:hets.eu`).
6.5.3(d)	A formal semantics for the abstract syntax.	The formal semantics is given in clause 11.
6.5.4(a)	Existing OMS in existing serializations shall validate as OMS in the metalanguage with a minimum amount of syntactic adaptation.	Any document providing an OMS in a serialization of a DOL conformant language can be used as-is in DOL, by reference to its IRI. See 10.5.
6.5.4(b)	It shall be possible to refer to existing files/documents from an OMS implemented in the metalanguage without the need for modifying these files/documents.	Documents can be referenced by IRIs, see 9.7.1.
6.5.4(c)	Translations between logical languages shall preserve (possibly to different degrees) the semantics of the logical languages. Between a given pair of logical languages, several translations are possible.	The semantics of DOL is based on a heterogeneous logical environment, which contains institution comorphisms as translations, see 11.2. Institution comorphisms preserve semantics in a weak form through their satisfaction condition. The LoLa ontology specifies properties of translations (comorphisms) preserving more and more of the semantics, see annex A.
6.5.5(a)	Informative annexes shall establish the conformance of a number of relevant logical languages. An initial set of language translations may be part of an informative annex.	For conformance of logical languages, see 6.5.5(b) below. Conformance of some translations is established in annex I.

Continued on next page

Table 0.1 – *Continued from previous page*

ID	RFP requirement	How this proposal addresses requirement
6.5.5(b)	Conformance of the following subset of logical languages shall be established: OWL2 (with profiles EL, RL, QL), CLIF, RDF, UML class diagrams.	We establish conformance of OWL 2 (annex B), CLIF (annex C), RDF and RDF Schema (annex D) and UML class diagrams (annex E) with DOL.
6.5.5(c)	Conformance of a suitable set of translations among the languages mentioned in the previous bullet point shall be established.	Conformance of some translations is established in annex I.
6.5.6	Existing standards and best practices for allocating globally unique identifiers shall be reused. The same standards and best practices shall also be applied to associate different representations of the same content to one unique identifier.	DOL uses IRIs to reference documents (both DOL documents, as well as documents written in some conforming language). See 9.7.1.

0.3. Optional Requirements

ID	RFP requirement	How this proposal addresses requirement
6.6.1	Submissions may include additional languages without a standardized model theory.	This is left for future work.
6.6.2	Proposals may provide constructs for non-monotonic logics.	Currently, only monotonic logics are supported. However, DOL provides a circumscription-like non-monotonic structuring construct with syntax `o1 then %minimize o2`, see 9.4 and 10.2.3.
6.6.3	A characterization of the trade-offs among different translations.	This is left for future work.

0.4. Issues to be Discussed

ID	Discussion item	Resolution
6.7.(a)	Do existing language standards need to be extended or adapted in order to make them OntoIOp conforming.	The goal of DOL is to support existing languages without any adaptations, see also 6.5.4(a). However, in order to meet requirement 6.5.6, DOL-conforming languages should support the use of IRIs. If they do not, there is a mechanism for assigning IRIs to (fragments of) language documents even if the language itself does not support this, see 2.2. Moreover, there is a mechanism for injecting IRIs in existing language serializations, see 10.5 and 8.8.
6.7.(b)	Proposals should discuss whether the semantics of the metalanguage shall be included into the standard	We have included the DOL metalanguage semantics in this specification. The reasons are discussed in the introduction of clause 11.
6.7.(c)	Proposals should discuss the chosen list of logics and translations.	The chosen list of logics and translations is discussed in the introduction of annex I.
6.7.(d)	Proposals should discuss a meta-ontology of logical languages and theories.	The LoLa ontology is discussed in annex A.
6.7.(e)	Proposals should discuss the use of QVT for expressing logic translations.	This has been left for future work.
6.7.(f)	Proposals should discuss the role of APIs.	The role of APIs is discussed in section N.3.
6.7.(g)	Proposals should discuss availability and use of tools.	Tools for DOL are discussed in annex N.
6.7.(h)	Proposals should discuss a registry of logical languages.	A registry is discussed in clause 2.

0.5. Evaluation Criteria

ID	Criterion	Comment
6.8(a)	Proposals covering a broader range of features and of use cases will be favored. As a minimum, proposals shall define conformance criteria for logical languages and translations, and their proposed metalanguage shall cover some metalogical relationships and shall be applicable to multiple logics.	Based on the notion of institution, conformance criteria for logical languages are defined in 2.1 and those for translations in 2.1.1. DOL covers several metalogical relationships, namely entailments, interpretations, equivalences, refinements, alignments and module relations, see 9.5 and 10.2.4. DOL is applicable to multiple logics (see also 6.8(c) and 8.5 below).
6.8(b)	Proposals covering existing language standards without (or with fewer) modifications will be favored.	Any document providing an OMS in a serialization of a DOL conforming language can be used as-is in DOL, by reference to its IRI. See 10.5.
6.8(c)	Proposals establishing actually (or making this at least possible in theory) OntoIOp conformance of more logical languages and translations will be favored.	We establish conformance of OWL 2 (annex B), Common Logic (annex C), RDF and RDF Schema (annex D), UML class diagrams (annex E) and CASL (annex H) with DOL.

0.6. Proof of Concept

Prototypical open source tools for DOL are already available, see annex N. It is expected that they will reach industrial strength within two or three years.

0.7. Changes to Adopted OMG Specifications

This specification proposes no changes to adopted OMG specifications.

1. Scope

This OMG Specification specifies the Distributed Ontology, Model and Specification Language (DOL). DOL is designed to achieve integration and interoperability of ontologies, specifications and models (OMS for short). DOL is a language for distributed knowledge representation, system specification and model-driven development across multiple OMS, particularly OMS that have been formalized in different OMS languages. This OMG Specification responds to the OntoIOp Request for Proposals [25].

1.1. Background Information

Logical languages are used in several fields of computing for the development of formal, machine-processable texts that carry a formal semantics. Among those fields are 1) **O**ntologies formalizing domain knowledge, 2) (formal) **M**odels of systems, and 3) the formal **S**pecification of systems. Ontologies, models and specifications will (for the purpose of this document) henceforth be abbreviated as **OMS**.

An OMS provides formal descriptions, which range in scope from domain knowledge and activities (ontologies, models) to properties and behaviors of hardware and software systems (models, specifications). These formal descriptions can be used for the analysis and verification of domain models, system models and systems themselves, using rigorous and effective reasoning tools. As systems increase in complexity, it becomes concomitantly less practical to provide a monolithic logical cover for all. Instead various models are developed to represent different viewpoints or perspectives on a domain or system. Hence, interoperability becomes a crucial issue, in particular, formal interoperability, i.e. interoperability that is based on the formal semantics of the different viewpoints. Interoperability is both about the ability to interface different domains and systems and the ability to use several OMS in a common application scenario. Further, interoperability is about coherence and consistency, ensuring at an early stage of the development that a coherent system can be reached.

In complex applications, which involve multiple OMS with overlapping concept spaces, it is often necessary to identify correspondences between concepts in the different OMS; this is called OMS alignment. While OMS alignment is most commonly studied for OMS formalized in the same OMS language, the different OMS used by complex applications may also be written in different OMS languages, which may even vary in their expressiveness. This OMG Specification faces this diversity not by proposing yet another OMS language that would subsume all the others. Instead, it accepts the diverse reality and formulates means (on a sound and formal semantic basis) to compare and integrate OMS that are written in different formalisms. It specifies DOL, a formal language for expressing not only OMS but also mappings between OMS formalized in different OMS languages.

Thus, DOL gives interoperability a formal grounding and makes heterogeneous OMS and services based on them amenable to checking of coherence (e.g. consistency, conservativity, intended consequences, and compliance).

1.2. Features Within Scope

The following are within the scope of this OMG Specification:

1. homogeneous OMS as well as heterogeneous OMS (OMS that consist of parts, which are written in different languages);
2. mappings between OMS (mapping OMS symbols to OMS symbols);
3. OMS networks (they involve several OMS and mappings between them);
4. translations between different OMS languages conforming with DOL (translating whole OMS to another language);
5. annotation and documentation of OMS, mappings between OMS, symbols, and sentences;
6. recommendations of vocabularies for annotating and documenting OMS;
7. a syntax for embedding the constructs mentioned under (1)–(5) as annotations into existing OMS;
8. a syntax for expressing (1)–(4) as standoff markup that points into existing OMS;
9. a formal semantics of (1)–(4);
10. structuring constructs for modeling non-monotonic behaviour;
11. criteria for existing or future OMS languages to conform with DOL.

The following are outside the scope of this OMG Specification:

1. the (re)definition of elementary OMS languages, i.e. languages that allow the declaration of OMS symbols (non-logical symbols) and stating sentences about them;
2. algorithms for obtaining mappings between OMS;
3. concrete OMS and their conceptualization and application;
4. mappings between services and devices, and definitions of service and device interoperability;
5. non-monotonic logics[1].

This OMG Specification describes the syntax and the semantics of the Distributed Ontology, Model and Specification Language (DOL) by defining an abstract syntax and an associated model-theoretic semantics for DOL.

[1]Logics (formalized as institutions) are necessarily monotonic; conformance criteria for non-monotonic logics ares till under development. However, minimization provides non-monotonic reasoning in DOL. It is also possible to include non-monotonic logics by construing entailments between formulas as sentences of the institution.

2. Conformance

This clause defines conformance criteria for languages and logics that can be used with DOL, as well as conformance criteria for serializations, translations and applications. The conformance of a number of OMS languages (namely OWL 2, Common Logic, RDF and RDF Schema, UML Class Diagrams, CASL) as well as translations among these is discussed in informative annexes of this OMG Specification.

It is expected that DOL will be used for other languages than the set of DOL-conforming languages that are discussed in this OMG Specification. There is a **registry for DOL-conforming languages and translations** hosted at `http://logichub.org`. The registry also includes descriptions of DOL-conforming languages and translations (as well as other information needed by implementors and users) in both human-readable and machine-processable form.

There will be Maintenance Authority (MA)[1] established to maintain the registry as an informative resource governed by the standard. The registry contents itself will not be normative; however, it is expected to become the basis for normative activities.

2.1. Conformance of an OMS Language/a Logic with DOL

Rationale: for an OMS language to conform with DOL,

- its logical language aspect either needs to satisfy certain criteria about its abstract syntax and formal semantics itself, or there must be a translation (again satisfying certain criteria) to a language that already is DOL-conforming.
- its structuring language aspect (if present) must not conflict with DOL's own structuring mechanisms
- its annotation language aspect must not conflict with DOL's meta-language constructs.

We also define different conformance levels with respect to the usage of IRIs as identifiers for all kinds of entities that the OMS language supports.

An OMS language is conforming with DOL if it satisfies the following conditions:

1. its abstract syntax is specified as an SMOF compliant meta model or as an EBNF grammar;
2. it has at least one serialization in the sense of section 2.2;
3. either there exists a translation of it into a conforming language[2], or:
   a) the logical language aspect (for expressing basic OMS) is conforming, and in particular has a semantics (see below),
   b) the structuring language aspect (for expressing structured OMS and relations between those) is conforming (see below), and

[1] or, depending on advisability, a Registration Authority

[2] For example, consider the translation of OBO1.4 to OWL, giving a formal semantics to OBO1.4.

c) the annotation language aspect (for expressing comments and annotations) is conforming (see below).

The *logical language aspect* of an OMS language is conforming with DOL if each logic corresponding to a profile (including the logic corresponding to the whole logical language aspect) is presented as an institution [22], and there is a mapping from the abstract syntax of the OMS language to signatures and sentences of the institution. Note that one OMS language can have several sublanguages or profiles corresponding to several logics (for example, OWL 2 has profiles EL, RL and QL, apart from the whole OWL 2 itself).

The *structuring language aspect* of an OMS language is conforming with DOL if it can be mapped to DOL's structuring language in a semantics-preserving way. The structuring language aspect **may** be empty.

The *annotation language aspect* of an OMS language is conforming with DOL if its constructs have no impact on the semantics. The annotation language aspect **shall** be non-empty; it **shall** provide the facility to express comments.

We define the following levels of conformance of the abstract syntax of a OMS language with DOL, listed from highest to lowest:

Full IRI conformance The abstract syntax specifies that IRIs be used for identifying all symbols and entities.

No mandatory use of IRIs The abstract syntax does not require IRIs to be used to identify entities. Note that this includes the case of optionally supporting IRIs without enforcing their use (such as in Common Logic).

Any conforming language and logic shall have a machine-processable description as detailed in clause 2.3.

2.1.1. Conformance of language/logic translations with DOL

Rationale: a translation between logics must satisfy certain criteria in order to conform with DOL. Also, a translation between OMS languages based on such logics must be consistent with the translation between these logics. Translations should break neither structuring language aspects nor comments/annotations.

A logic translation is conforming with DOL if it is presented either as an institution morphism or as an institution comorphism.

A language translation **shall** provide a mapping between the abstract syntaxes (it **may** also provide mappings between concrete syntaxes). A language translation from language L_1 (based on institution I_1) to language L_2 (based on institution I_2) is conforming with DOL if it is based on a logic translation such the following diagram commutes (i.e. following both possible paths from L_1 to I_2 leads to the same result):

$$\begin{tikzcd}
L_1 \arrow[r, "\text{mapping between abstract syntaxes}"] \arrow[d, "\text{abstract syntax to institution}"'] & L_2 \arrow[d, "\text{abstract syntax to institution}"] \\
I_1 \arrow[r, "\text{institution (co)morphism}"] & I_2
\end{tikzcd}$$

Language translations **may** also translate the structuring language aspect, in this case, they **shall** preserve the semantics of the structuring language aspect. Furthermore, language translations **should** preserve comments and annotations. All comments attached to a sentence (or symbol) in the source **should** be attached to its translation in the target (if there are more than one sentences (resp. symbols) expressing the translation, to at least one of them).

2.2. Conformance of a Serialization of an OMS Language With DOL

Rationale: The main reason for the following specifications is identifier injection. DOL is capable of assigning identifiers to entities (symbols, axioms, modules, etc.) inside fragments of OMS languages that occur in a DOL document, even if that OMS language doesn't support such identifiers by its own means. Such identifiers will be visible to a DOL tool, but not to a tool that only supports the OMS language. To achieve this without breaking the formal semantics of that OMS language, we make use of annotation or commenting features that the OMS language supports, in order to place such identifiers inside annotations/ comments. Depending on the nature of the concrete given serialization of the OMS language, be it plain text, some serialization of RDF, XML, or some other structured text format, we can be more specific about what the annotation/commenting facilities of that serialization must look like in order to support this identifier injection. Well-behaved XML and RDF schemas support identifier injection in a 'nice' way (rather than using text-level comments). In the worst case we cannot inject anything into an OMS language fragment, because the OMS language serialization simply would not allow us to write suitable comments, but we would have to point into it from the enclosing context by using standoff markup.

Further conformance criteria in this section are introduced to facilitate the convenient reuse of verbatim fragments of OMS language inside a DOL document.

Independently from these criteria, we distinguish different levels of conformance of a serialization with respect to its means of conveniently abbreviating long IRI identifiers.

We define seven levels of conformance of a serialization of an OMS language with DOL.

XMI conformance An XMI serialization has been automatically derived from the SMOF specification of the abstract syntax, using MOF 2 XMI Mapping.

XML conformance The given serialization has to be specified as an XML schema[3], which satisfies all of the following conditions:

[3]Here, we refer to the general *concept* of a schema, not of the W3C XML Schema language as one

- The elements of the schema belong to one or more non-empty XML namespaces.
- The serialization shall use XML *elements* to represent all structural elements of an OMS.
- The schema shall not forbid both attributes and child elements from foreign namespaces (here: the DOL namespace `http://www.omg.org/spec/DOL/1.0/xml`) on any elements.[4]

RDF conformance The given serialization has to be specified as an RDF vocabulary, which satisfies all of the following conditions:

- The elements of the vocabulary belong to one or more RDF namespaces identified by absolute URIs.
- The serialization shall specify ways of giving IRIs or URIs to all structural elements of an OMS.[5] [6]
- There shall be no additional rules (stated in writing in the specification of the serialization, or formalized in its implementation in, e.g., OWL) that forbid properties from foreign vocabulary namespaces to be stated about arbitrary subjects for the purpose of annotation.

Text conformance The given serialization has to satisfy all of the following conditions:

- The serialization conforms with the requirements for the *text/plain* media type specified in IETF/RFC 2046, section 4.1.3.
- The serialization shall provide a designated comment construct that can be placed sufficiently flexible as to be uniquely associated with any non-comment construct of the language. That means, for example, one of the following:
  - The serialization provides a construct that indicates the start and end of a comment and may be placed before/after each token that represents a structural element of an OMS.
  - The serialization provides line-based comments (ranging from an indicated position to the end of a line) but at the same time allows the flexible placement of line breaks before/after each token that represents a structural element of an OMS.

---

way of implementing it. It is not even required that a machine-readable implementation of the schema serialization exists.

[4]This is because either an attribute or a child element is used to inject identifiers into elements of the XML serialization; cf. clause 10.5.

[5]The rationale is that RDF in principle allows for identifying any kinds of items, so an RDF-based serialization of an OMS language should not forbid making use of such RDF constructs that do allow for identifying arbitrary items.

[6]The OWL RDF serialization, for example, does not satisfy the RDF conformance level, for the following reason. There is an `owl:imports` property but no class representing imports. Therefore, it is not possible to represent a concrete import, of an ontology O_1 importing an ontology O_2, as a resource, which could have an identifier. RDF reification would allow for giving the statement O_1 `owl:imports` O_2 an identifier. However, the RDF triples resulting from this reification, including, e.g., the triple `:import_id rdf:predicate owl:imports`, would not match the head of any rule in the mapping from RDF graphs to the OWL structural specification `http://www.w3.org/TR/2012/REC-owl2-mapping-to-rdf-20121211/#Mapping_from_RDF_Graphs_to_the_Structural_Specification`). They would thus remain left over in the RDF graph that is attempted to be parsed into an OWL ontology, and thus violate the requirement that at the end of this parsing process, the RDF graph must be empty.

Standoff markup conformance An OMS language is standoff markup conforming with DOL if one of its serializations conforms with the requirements for the *text/plain* media type specified in IETF/RFC 2046, section 4.1.3. Note that conformance with *text/plain* is a prerequisite for using, for example, fragment URIs in the style of IETF/RFC 5147 for identifying text ranges.

Independently from the conformance levels given above, there is the following hierarchy of conformance w.r.t. CURIEs (compact URIs) as a means of abbreviating IRIs (grammar specified in clause 9.7.2), listed from highest to lowest:

Prefixed CURIE conformance The given serialization allows non-logical symbol identifiers to have the syntactic form of a CURIE, or any subset of the CURIE grammar that allows named prefixes (`prefix:reference`, where a declaration of DOL-conformance of a serialization **may** redefine the separator character to a character different from `:`). A serialization that conforms w.r.t. prefixed CURIE conformande is **not required** to support CURIEs with no prefix: its declaration of DOL-conformance **may** forbid the use of prefixed CURIEs.
Informative comments:

- In the case that CURIEs are used, a prefix map with multiple prefixes **may** be used to map the non-logical symbol identifiers of a native OMS to IRIs in multiple namespaces (cf. clause 9.7.3)
- The reason for allowing redefinitions of the prefix/reference separator character is that certain serializations of OMS languages may not allow the colon (`:`) in identifiers.

Non-prefixed names only The given serialization only supports CURIEs with no prefix, or any subset of the grammar of the `REFERENCE` nonterminal in the CURIE grammar.
Informative comment: In this case, a binding for the empty prefix **has to** be declared, as this is the only possibility of mapping the identifiers of the native OMS to IRIs, which are located in one flat namespace.

Any conforming serialization of an OMS language shall have a machine-processable description as detailed in clause 2.3.

2.3. Machine-Processable Description of Conforming Languages, Logics, and Serializations

Rationale: When a parser processes a DOL OMS found somewhere, which refers to modules in OMS languages, or includes them verbatim, the parser needs to know what language to expect; further DOL-supporting software needs to know, e.g., what other DOL-conforming languages the module in the given OMS language can be translated to. Therefore we require that all languages/logics/serializations that conform with DOL describe themselves in a machine-comprehensible way.

For any conforming OMS language, logic, and serialization of an OMS language, it is required that it be assigned an HTTP IRI, by which it can be identified. It is also required that a machine-processable description of this language/logic/serialization be retrievable by dereferencing this IRI, according to the linked data principles. At least there has to be an RDF description in terms of the vocabulary specified in annex A, which has to be made available in the RDF/XML serialization when a client requests content of the MIME type *application/rdf+xml*. Descriptions of the language/logic/serialization in further representations, having different content types, may be provided.

2.4. Conformance of a Document With DOL

Rationale: for exchanging DOL documents with other users/tools, nothing that has a formal semantics must be left implicit. One DOL tool may assume that by default any OMS fragments inside a DOL document are in some fixed OMS language unless specified otherwise, but another DOL tool can't be assumed to understand such DOL documents. Defaults are, however, practically convenient, which is the reason for having the following section about the conformance of an *application*.

A document conforms with DOL if it contains a DOL text that is well-formed according to the grammar. That means, in particular, that any information related to logics has to be made explicit (as foreseen by the DOL abstract syntax specified in clause 9), such as:

- the logic of each OMS that is part of the DOL document,
- the translation that is employed between two logics (unless it is one of the default translations specified in annex I)

However, details about aspects of an OMS that do not have a formal, logic-based semantics, may be left implicit. For example, a conforming document may omit explicit references to matching algorithms that have been employed in obtaining an alignment.

2.5. Conformance of an Application With DOL

In practice, DOL-aware *applications* may also deal with documents that are not conforming with DOL according to the criteria established in clause 2.4. However, an application only *conforms* with DOL if it is capable of producing DOL-conforming documents as its output when requested.

We expect most DOL-aware applications to support a fixed (possibly extensible) set of OMS languages conforming with DOL. It is, for example, possible that a DOL-aware application only supports OWL and Common Logic. In that case, the application may process documents that mix OWL and Common Logic ontologies *without* explicitly declaring the respective logics, as the respective syntaxes of OWL and Common Logic can be distinguished by examining the different keywords. However, for DOL conformance, that application has to be capable of exporting documents with explicit references to the logics used.

DOL-aware applications also should be able to strip DOL annotations from embedded fragments in other OMS languages. Moreover, they shoud be able to expand CURIEs into IRIs when requested.

3. Normative References

1. W3C/TR REC-owl2-syntax:2009 OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. W3C Recommendation, 27 October 2009.
   `http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/`
2. ISO/IEC 14977:1996 Information technology – Syntactic metalanguage – Extended BNF
3. W3C/TR REC-xml:2008 Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation, 26 November 2008.
   `http://www.w3.org/TR/2008/REC-xml-20081126/`
4. W3C/TR REC-owl2-profiles:2009 OWL 2 Web Ontology Language: Profiles. W3C Recommendation, 27 October 2009.
   `http://www.w3.org/TR/2009/REC-owl2-profiles-20091027/`
5. ISO/IEC 24707:2007 Information technology – Common Logic (CL): a framework for a family of logic-based languages
6. OMG Document ptc/2010-11-14: OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.4.
   `http://www.omg.org/spec/UML/2.4/Superstructure`. Section 7 (Classes)
7. IETF/RFC 3986 Uniform Resource Identifier (URI): Generic Syntax. January 2005.
   `http://tools.ietf.org/html/rfc3986`
8. IETF/RFC 3987 Internationalized Resource Identifiers (IRIs). January 2005.
   `http://tools.ietf.org/html/rfc3987`
9. IETF/RFC 5147 URI Fragment Identifiers for the text/plain Media Type. April 2008.
   `http://tools.ietf.org/html/rfc5147`
10. W3C/TR REC-xptr-framework:2003 XPointer Framework. W3C Recommendation, 25 March 2003.
    `http://www.w3.org/TR/2003/REC-xptr-framework-20030325/`
11. W3C/TR REC-rdf11-concepts:2014 RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, 25 February 2014.
    `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`
12. W3C/TR REC-xml-names:2009 Namespaces in XML 1.0 (Third Edition). W3C Recommendation, 8 December 2009.
    `http://www.w3.org/TR/2009/REC-xml-names-20091208/`
13. W3C/TR REC-rdfa-core:2013 RDFa Core 1.1 – Second Edition. Syntax and processing rules for embedding RDF through attributes. W3C Recommendation, 22 August 2013.
    `http://www.w3.org/TR/2013/REC-rdfa-core-20130822/`
14. ISO/IEC 10646 Information technology – Universal Multiple-Octet coded Character Set (UCS)

15. W3C/TR REC-rdf-schema:2014 RDF Schema 1.1. W3C Recommendation, 25 February 2014.
`http://www.w3.org/TR/2014/REC-rdf-schema-20140225/`

16. W3C/TR REC-rdf11-mt:2014 RDF 1.1 Semantics. W3C Recommendation, 25 February 2014.
`http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/`

17. ODM Ontology Definition Metamodel, 2 September 2014.
`http://www.omg.org/spec/ODM/1.1/`

18. MOF Meta Object Facility
`http://http://www.omg.org/mof/`

19. SMOF Support for Semantic Structure, April 2013
`http://www.omg.org/spec/SMOF/1.0/`

20. XML Metadata Interchange (XMI) – using MOF 2 XMI, April 2014
`http://www.omg.org/spec/XMI//`

21. SBVR Semantics Of Business Vocabulary And Rules, November 2013
`http://www.omg.org/spec/SBVR/`

22. DTV Date-Time Vocabulary, August 2013
`http://www.omg.org/spec/DTV/1.0/`

23. RIF Rule Interchange Format, February 2013
`http://www.w3.org/TR/rif-overview/`

24. SKOS Simple Knowledge Organization System
`http://www.w3.org/2004/02/skos/`

4. Terms and Definitions

For the purposes of this document, the following terms and definitions apply.

4.1. Distributed Ontology, Model and Specification Language

Distributed Ontology, Model and Specification Language; DOL language for formalizing libraries of OMS and OMS networks, whose syntax and semantics are specified in this OMG Specification.

library collection of named OMS and OMS networks, possibly written in different OMS languages, linked by named OMS mappings.

4.2. Native OMS, OMS, and OMS Languages

native OMS A collection of expressions (like non-logical symbols, sentences and structuring elements) from a given OMS language.
EXAMPLE An UML class diagram, an ontology written in OWL 2 EL, and a specification written in CASLE are three different native OMS.
NOTE An OMS can be written in different OMS language serializations.

OMS language language equipped with a formal, declarative, logic-based semantics, plus non-logical annotations.
EXAMPLE OMS languages include OWL 2 DL, Common Logic, F-logic, UML class diagrams, RDF Schema, and OBO.
NOTE An OMS language is used for the formal specification of native OMS.
NOTE We distinguish between the logical language aspect, the structuring language aspect, and the annotation language aspect of an OMS language.

DOL structured OMS A syntactically valid DOL library (as defined in this OMG Specification) that contains a library definition (see section 10.4.1).
NOTE DOL structured OMS, typically, use native OMS as building blocks for defining structured OMS, OMS mappings or OMS networks.
NOTE All DOL structured OMS are structured OMS.

OMS (ontology, specification or model) A native OMS or a DOL structured OMS.
NOTE An OMS is either a basic or a structured OMS.
NOTE An OMS has a single signature and model class over that signature as its model-theoretic semantics.

basic OMS; flat OMS A native OMS that does not utilize any elements from the structuring language aspects of its language.
NOTE Basic OMS are self-contained in the sense that it semantics does not depend on some other OMS. In particular, the OMS does not involve any importations.
NOTE Since a basic OMS has no structuring elements, it consists just of a signature equipped with a set of sentences and annotations.

non-logical symbol; OMS symbol atomic expression or syntactic constituent of an OMS that requires an interpretation through a model.
NOTE This differs from the notion of "atomic sentence": such sentences may involve several non-logical symbols.

EXAMPLE Non-logical symbols in OWL W3C/TR REC-owl2-syntax:2009 (there called "entities") comprise

- individuals (denoting objects from the domain of discourse),
- classes (denoting sets of objects; also called concepts), and
- properties (denoting binary relations over objects; also called roles).

This is opposed to logical symbols in OWL, e.g. those for intersection and union of classes.

EXAMPLE Non-logical symbols in Common Logic ISO/IEC 24707:2007 comprise

- names (denoting objects from the domain of discourse),
- sequence markers (denoting sequences of objects).

This is opposed to logical symbols in Common Logic, e.g. logical connectives and quantifiers.

signature; vocabulary set (or otherwise structured collection) of non-logical symbols of an OMS.
NOTE The signature of a term is the set of all non-logical symbols occurring in the term. The signature of an OMS language is the set of all non-logical symbols possible in that language.
NOTE The signature of an OMS is usually uniquely determined.

model semantic interpretation of all non-logical symbols of a signature.
NOTE A model of an OMS is a model of the signature of the OMS that moreover satisfies all the axioms of the OMS.
NOTE This term refers to *model* in the sense of model theory (a branch of logic). It is not to be confused with *model* in the sense of modeling (i.e., the "M" in OMS).

term syntactic expression either consisting of a single non-logical symbol or recursively composed of other terms (a.k.a. its subterms).
NOTE A term belongs to the logical language aspect of an OMS language.

sentence term that is either true or false in a given model, i.e. which is assigned a truth value in this model.
NOTE In a model, on the one hand, a sentence is always true or false. In an OMS, on the other hand, a sentence can have several logical statuses: it can be an axiom, if postulated to

be true; a theorem, if proven from other axioms and theorems; a conjecture, if expecting to be proven from other axioms and theorems; or have another of many possible statuses.
NOTE A sentence can conform to one or more signatures (namely those signatures containing all non-logical symbols used in the sentence).
NOTE It is quite common that sentences are required to be closed (i.e. have no free variables). However, this depends on the OMS language at hand.
NOTE A sentence belongs to the logical language aspect of an OMS language.

axiom sentence that is postulated to be valid (i.e. true in every model).

theorem sentence that has been proven from other axioms and theorems.

satisfaction relation relation between models and sentences indicating which sentences hold true in the model.

logical theory signature equipped with a set of sentences over the signature.

entailment; specialization relation between two OMS expressing that the second one is logically implied by the first one.
NOTE The converse is generalization.

query language OMS language specifically dedicated to queries.
EXAMPLE SPARQL, Prolog
NOTE There are also general purpose OMS languages, which can express both OMS and queries.

query sentence containing query variables that can be instantiated by a substitution.

query variable symbol that will be used in a query and a substitution.
NOTE From an abstract point of view, query variables are just symbols; they are used in a way that they will be substituted using a substitution. Many OMS languages have special notations for (query) variables.
NOTE Usually, query variables are the free variables of a sentence; there can be other (bound) variables.
NOTE If there are no variables in an OMS language, constants can be used as query variables.

substitution OMS mapping that maps query variables of one OMS to complex terms of another OMS.

answer substitution substitution that, when applied to a given query, turns the latter into a logical consequence of a given OMS.

4.3. Semantic Web

resource something that can be globally identified.
NOTE IETF/RFC 3986:2005, Section 1.1 deliberately defines a resource as "in a general sense [...] whatever might be identified by [an IRI]". The original source refers to URIs, but DOL uses the compatible IRI standard IETF/RFC 3987:2005 for identification.
EXAMPLE Familiar examples include an electronic document, an image, a source of information with a consistent purpose (e.g., "today's weather report for Los Angeles"), a service (e.g., an HTTP-to-SMS gateway), and a collection of other resources. A resource is not necessarily accessible via the Internet; e.g., human beings, corporations, and bound books in a library can also be resources. Likewise, abstract concepts can be resources, such as the operators and operands of a mathematical equation, the types of a relationship (e.g., "parent" or "employee"), or numeric values (e.g., zero, one, and infinity). IETF/RFC 3986:2005, Section 1.1

element (of an OMS) any resource in an OMS (e.g. a non-logical symbol, a sentence, a correspondence, the OMS itself, ...) or a named set of such resources.

linked data structured data that is published on the Web in a machine-processable way, according to principles specified in [41, 8].
NOTE The linked data principles (adapted from [41] and its paraphrase at [65]) are the following:

1. Use IRIs as names for things.
2. Use HTTP IRIs so that these things can be referred to and looked up ("dereferenced") by people and user agents.[1]
3. Provide useful machine-processable (plus optionally human-readable) information about the thing when its IRI is dereferenced, using standard formats.
4. Include links to other, related IRIs in the exposed data to improve discovery of other related information on the Web.

NOTE RDF, serialized as RDF/XML [28], is the most common format for publishing linked data. However, its usage is not mandatory.
NOTE Using HTTP content negotiation [20] it is possible to serve representations in different formats from the same URL.

4.4. OMS Annotation and Documentation

annotation additional information without a logical semantics that is attached to an element of an OMS.
NOTE Formally, an annotation is given as a (subject, predicate, object) triple as defined by SOURCE: W3C/TR REC-rdf11-concepts:2014, Section 3.1. The subject of an annotation is an element of an OMS. The predicate is an RDF property defined in an external OMS and describes in what way the annotation object is related to the annotation subject.
NOTE According to note 4.4 it is possible to interpret annotations under an RDF semantics.

[1] I.e., the IRI is treated as a URL (uniform resource locator).

"Without a logical semantics" in this definition means that annotations to an OMS are not considered sentences of that OMS.

OMS documentation set of all annotations to an OMS, plus any other documents and explanatory comments generated during or after development or deployment of the OMS.
NOTE Adapted from [61].

4.5. Structured OMS

structured OMS OMS that results from other (basic and structured) OMS by import, union, combination, renaming or other structuring operations.
NOTE Structured OMS are either DOL structured OMS or native OMS that utilize elements of their structuring language aspect.

flattenable OMS OMS that can be seen, by purely syntactical means, to be logically equivalent to a flat OMS.
NOTE More precisely, an OMS is flattenable if and only if it is either a basic OMS or it is an extension, union, translation, module extraction, approximation, filtering, or reference of named OMS involving only flattenable OMS.

elusive OMS OMS that is not flattenable.

subOMS OMS whose sets of non-logical symbols and sentences are subsets of those present in a given larger OMS.

import reference to an OMS behaving as if it were verbatimly included; also import of libraries.
NOTE Semantically, an import of O_2 into O_1 is equivalent to the verbatim inclusion of O_2 in place of the import declaration.
NOTE The purpose of O_2 importing O_1 is to make non-logical symbols and sentences of O_1 available in O_2.
NOTE Importing O_1 into O_2 turns O_2 into an extension of O_1.
NOTE An owl:import in OWL is an import.
NOTE The import of a whole library into another library is also called import.

union aggregation of several OMS to a new OMS, without any renaming.

renaming assignment of new names to some non-logical symbols of an OMS.
NOTE A renaming results in an OMS mapping between the original and the renamed OMS.

reduction restriction of an OMS to a smaller signature.

extension OMS whose sets of non-logical symbols and sentences are supersets of those present in a given smaller OMS.

extension mapping inclusion OMS mapping between two OMS where the sets of non-logical symbols and sentences of the second OMS are supersets of those present in the first OMS.
NOTE The second OMS is said to extend the first, and is an extension of the first OMS.

consequence-theoretic conservative extension extension that does not add new theorems (in terms of the unextended signature).
NOTE An extension O_2 of an OMS O_1 is a consequence-theoretic conservative extension, if all properties formulated in the signature of O_1 hold for O_1 whenever they hold for O_2.

model-theoretic conservative extension extension that does not lead to a restriction of class of model s of an OMS.
NOTE An extension O_2 of an OMS O_1 is a model-theoretic conservative extension, if all properties formulated in the signature of O_1 hold for O_1 whenever they hold for O_2.
NOTE Any model-theoretic conservative extension is also a consequence-theoretic one.

conservative extension consequence-theoretic or model-theoretic conservative extension.
NOTE If used without qualification, the consequence-theoretic version is meant.

monomorphic extension extension whose newly introduced non-logical symbols are interpreted in a way unique up to isomorphism.
NOTE An extension O_2 of an OMS O_1 is a monomorphic extension, if each model of O_1 can be expanded to a model of O_2 that is unique up to isomorphism.
NOTE Each monomorphic extension is also a model-theoretic conservative extension but not vice versa.

definitional extension extension whose newly introduced non-logical symbols are interpreted in a unique way.
NOTE An extension O_2 of an OMS O_1 is a definitional extension, if each model of O_1 can be uniquely expanded to a model of O_2.
NOTE O_2 being a definitional extension of O_1 implies a bijective correspondence between the classes of models of O_2 and O_1.
NOTE Each definitional extension is also a monomorphic extension but not vice versa.

weak definitional extension extension whose newly introduced non-logical symbols can be interpreted in at most one way.
NOTE An extension O_2 of an OMS O_1 is a weak definitional extension, if each model of O_1 can be expanded to at most one model of O_2.
NOTE An extension is definitional if and only if it is both weakly definitional and model-theoretically conservative.

implied extension model-theoretic conservative extension that does not introduce new non-logical symbols.
NOTE A conservative extension O_2 of an OMS O_1 is an implied extension, if and only if the signature of O_2 is the signature of O_1. O_2 is an implied extension of O_1 if and only if the model class of O_2 is the model class of O_1.
NOTE Each implied extension is also a definitional extension but not vice versa.

module subOMS that conservatively extends to conservative extension the whole OMS.
NOTE The conservative extension can be either model-theoretic or consequence-theoretic; without qualification, the consequence-theoretic version is used.

module extraction activity of obtaining from an OMS concrete modules to be used for a particular purpose (e.g. to contain a particular sub-signature of the original OMS).
NOTE Cited and slightly adapted from [61].
NOTE The goal of module extraction is "decomposing an OMS into smaller, more manageable modules with appropriate dependencies" [60].
EXAMPLE Consider an OWL DL ontology about wines, from which we would like to extract a module about white wines. That module would contain the declaration of the non-logical symbol "white wine", all declarations of non-logical symbols related to "white wine", and all sentences about all of these non-logical symbols.

approximant approximation (in the sense of a logically implied theory, possibly after suitable translation) of an OMS in a smaller signature or a sublanguage.

maximum approximant best possible (in the sense of a maximum set of logical consequences) approximant of an OMS in a smaller signature or a sublanguage.
NOTE Technically, a maximum approximant is a uniform interpolant, see [45].

closed world assumption presumption that what is not known to be true, is false.

minimization; circumscription way of implementing the closed world assumption by restricting the models to those that are minimal.
NOTE See [48], [42].

combination aggregation of all the OMS in an OMS network, where non-logical symbol s are shared according to the OMS mapping s in the OMS network.
EXAMPLE Consider an ontology involving a concept `Person`, and another one involving `Human being`, and an alignment that relates these to concepts. In the combination of the ontologies along the alignment, there is only one concept, representing both `Person` and `Human being`.

sharing property of OMS symbols being mapped to the same symbol when computing a combination of an OMS network.
NOTE Sharing is always relative to a given OMS network that relates different OMS. That is, two given OMS symbols can share with respect to one OMS network, and not share with respect to some other OMS network.

4.6. Mappings Between OMS

OMS mapping; link relationship between two OMS.

symbol map item pair of symbols of two OMS, indicating how a symbol from the first OMS is mapped by a signature morphism to a symbol of the second OMS
NOTE A symbol map item is given as $s_1 \mapsto s_2$, where s_1 is a symbol from the *source* OMS and s_2 is a symbol from the *target* of the OMS mapping.

interpretation; view; refinement OMS mapping that postulates a specialization relation between two OMS along a morphism between their signatures.
NOTE An interpretation typically leads to proof obligations, i.e. one has to prove that translations of axioms of the source OMS along the morphism accompanying the interpretation are theorems in the target OMS.

equivalence OMS mapping ensuring that two OMS share the same definable concepts.
NOTE Two OMS are equivalent if they have a common definitional extension. The OMS may be written in different OMS languages.

interface signature signature mediating between an OMS and a module of that OMS in the sense that it contains those non-logical symbols that the sentences of the module and the sentences of the OMS have in common.
NOTE Adapted from [24].

module relation OMS mapping stating that one OMS is a module of the other one.

alignment an OMS mapping expressing a collection of semantic relations between entities of the two OMS.
NOTE Alignments consist of correspondences, each of which may have a confidence value. If all confidence values are 1, the alignment can be given a formal, logic-based semantics.

correspondence relationship between an non-logical symbol e_1 from an OMS O_1 and an non-logical symbol e_2 from an OMS O_2, or between an non-logical symbol e_1 from O_1 and a term t_2 formed from non-logical symbols from O_2.
NOTE A correspondence is given as a quadruple $(e_1, R, \left\{ \begin{array}{c} e_2 \\ t_2 \end{array} \right\}, c)$, where R denotes the type of relationship that is asserted to hold between the two non-logical symbols/terms, and $0 \leq c \leq 1$ is a confidence value. R and c may be omitted: When R is omitted, it defaults to the equivalence relation, unless another default relation has been explicitly specified; when c is omitted, it defaults to 1.
NOTE A confidence value of 1 does not imply logical equivalence (cf. [39] for a worked-out example).
NOTE Not all OMS languages implement logical equivalence. For example, OWL does not implement logical equivalence in general, but separately implements equivalence relations restricted to individuals (*owl:sameAs*), classes (*owl:equivalentClass*) and properties (*owl:equivalentProperty*).

matching algorithmic procedure that generates an alignment for two given OMS.
NOTE For both matching and alignment, see [19, 34].

OMS network; distributed OMS; hyperontology graph with OMS as nodes and OMS mappings as edges, showing how the OMS are interlinked.
NOTE In [54], a distinction between focused and distributed heterogeneous specifications is made. In the terminology of this standard, this is the distinction between OMS and OMS networks.
NOTE An OMS network is a diagram of OMS in the sense of category theory, but different from a diagram in the sense of model-driven architecture.
NOTE The links between the nodes of a distributed OMS can be given using interpretations or alignments. Imports between the nodes of a distributed OMS are automatically included in the distributed OMS. By including an interpretation or an alignment in a distributed OMS, the involved nodes are automatically included.
EXAMPLE Consider two ontologies and an interpretation between them. In the distributed OMS of the interpretation there are two nodes, one for each ontology, and one edge from the source ontology to the target ontology of the interpretation.

4.7. Features of OMS Languages

OMS language translation mapping from constructs in the source OMS language to their equivalents in the target OMS language.
NOTE An OMS language translation shall satisfy the property that the result of a translation is a well-formed text in the target language.

OMS language graph graph of OMS languages and OMS language translations, typically used in a heterogeneous environment.
NOTE In an OMS language graph, some of the OMS language translations can be marked to be default translations.

default translation specially marked OMS language translation or logic translation that will be used whenever a translation is needed and no explicit translation is given.

heterogeneous environment environment for the expression of homogeneous and heterogeneous OMS, comprising a logic graph, an OMS language graph and supports relations.
NOTE Although in principle, there can be many heterogeneous environments, for ensuring interoperability, there will be a global heterogeneous environment (maintained in some registry), with subenvironments for specific purposes.

sublanguage syntactically specified subset of a given language, consisting of a subset of its terminal and nonterminal symbols and grammar rules.

language aspect set of language constructs of a given language, not necessarily forming a sublanguage.

logical language aspect the (unique) language aspect of an OMS language that enables the expression of non-logical symbols and sentences in a logic.

structuring language aspect the (unique) language aspect of an OMS language that covers structured OMS as well as the relations of basic OMS and structured OMS to each other, including, but not limited to imports, OMS mappings, conservative extensions, and the handling of prefixes for CURIEs.

annotation language aspect the (unique) language aspect of an OMS language that enables the expression of comments and annotations.

profile (syntactic) sublanguage of an OMS language interpreting according to a particular logic that targets specific applications or reasoning methods.
EXAMPLE Profiles of OWL 2 include OWL 2 EL, OWL 2 QL, OWL 2 RL, OWL 2 DL, and OWL 2 Full.
NOTE Profiles typically correspond to sublogics.
NOTE Profiles can have different logics, even with completely different semantics, e.g. OWL 2 DL versus OWL 2 Full.
NOTE The logic needs to support the language.

4.8. OMS Language Serializations

serialization specific syntactic encoding of a given OMS language.
NOTE Serializations serve as standard formats for exchanging OMS between tools.
EXAMPLE OWL uses the term "serialization"; the following are standard OWL serializations: OWL functional-style syntax, OWL/XML, OWL Manchester syntax, plus any standard serialization of RDF (e.g. RDF/XML, Turtle, ...). However, RDF/XML is the only one tools are required to implement.
EXAMPLE Common Logic uses the term "dialect"; the following are standard Common Logic dialects: Common Logic Interchange Format (CLIF), Conceptual Graph Interchange Format (CGIF), eXtended Common Logic Markup Language (XCL).

document result of serializing an OMS using a given serialization.

standoff markup way of providing annotations to subjects in external resources, without embedding them into the original resource (here: OMS).

4.9. Logic

logic specification of valid reasoning that comprises signatures, sentences, models , and a satisfaction relation between models and sentences.
NOTE Most OMS languages have an underlying logic.
EXAMPLE $\mathcal{SROIQ}(D)$ is the logic underlying OWL 2 DL.
NOTE See annex A for the organization of the relation between OMS languages and their logics and serializations.

supports relation relation between OMS languages and logics expressing the logical language aspect of the former, namely that the constructs of the former lead to a logical theory in the latter.
NOTE There is also a supports relation between OMS languages and serializations.

institution metaframework mathematically formalizing the notion of a logic.
NOTE See clause 11 for a formal definition.

logic translation mapping of a source logic into a target logic (mapping signatures, sentences and models) that keeps or encodes the logical content of OMS.

logic reduction mapping of a source logic onto a (usually less expressive) target logic (mapping signatures, sentences and models) that simply forgets those parts of the logical structure not fitting the target logic.

theoroidal logic translation translation that maps signatures of the source logic to theories (i.e. signatures and sets of sentences) of the target logic.
EXAMPLE The translation from OWL to multi-sorted first-order logic translates each OWL built-in type to its first-order axiomatization as a datatype.

sublogic a logic that is a syntactic restriction of another logic, inheriting its semantics.

logic graph graph of logics, logic translations and logic reductions, typically used in a heterogeneous environment.
NOTE In a logic graph, some of the logic translations and reductions can be marked to be default translations.

homogeneous OMS OMS whose parts are all formulated in one and the same logic.
NOTE Opposite of heterogeneous OMS.

heterogeneous OMS OMS whose parts are formulated in different logics.
NOTE Opposite of homogeneous OMS.
EXAMPLE See section K.3.

logic approximation mapping of a source logic onto a (usually less expressive) target logic that tries to approximate the OMS expressed in the source logic with means of the expressivity of the target logic.
NOTE A unique maximal approximation need not exist.
NOTE The target logic typically is a sublogic of the source logic.

4.10. Interoperability

OMS integration the union of OMS which preserves the satisfaction and entailment relations among constituent OMS.

logically interoperable property of structured OMS, which may be written in different OMS languages supporting different logics, of being usable jointly in a coherent way (via suitable OMS language translations), such that the notions of their overall consistency and logical entailment have a precise logical semantics.
NOTE Within ISO 19763 and ISO 20943, metamodel interoperability is equivalent to the existence of mapping, which are statements that the domains represented by two models intersect and there is a need to register details of the correspondence between the structures in the models that semantically represent this overlap. Within these standards, a model is a representation of some aspect of a domain of interest using a normative modelling facility and modelling constructs.

The notion of logical interoperability is distinct from the notion of interoperability used in ISO/IEC 2381-1 Information Technology Vocabulary – Part 1: Fundamental Terms, which is restricted to the capability to communicate, execute programs, or transfer data among various hardware or software entities in a manner that requires the user to have little or no knowledge of the unique characteristics of those entities.

OMS interoperability relation among OMS (via OMS alignments) which are logically interoperable.

5. Symbols

As listed below, these symbols and abbreviations are generally for the main clauses of the OMG Specification. Some annexes may introduce their own symbols and abbreviations which will be grouped together within that annex.

CASL	Common Algebraic Specification Language, specified by the Common Framework Initiative
CGIF	Conceptual Graph Interchange Format
CL	Common Logic
CLIF	Common Logic Interchange Format
CORBA	Common Object Request Broker Architecture
CURIE	Compact URI expression
CWM	Common Warehouse Metamodel
DDL	Distributed description logic
DOL	Distributed Ontology, Model and Specification Language
DTV	Date-Time Vocabulary
EBNF	Extended Backus-Naur Form
E-connections	a modular ontology language (closely related to DDL)
F-logic	frame logic, an object-oriented ontology language
IDL	Interface Definition Language
IIOP	Internet Inter-ORB Protocol
IRI	Internationalized Resource Identifier
MDA	Model Driven Architecture
MOF	Meta-Object Facility
OCL	Object Constraint Language
OWL 2	Web Ontology Language (W3C), version 2: family of knowledge representation languages for authoring ontologies
OWL 2 DL	description logic profile of OWL 2
OWL 2 EL	a sub-Boolean profile of OWL 2 (used often e.g. in medical ontologies)
OWL 2 Full	the language that is determined by RDF graphs being interpreted using the OWL 2 RDF-Based Semantics [27]
OWL 2 QL	profile of OWL 2 designed to support fast query answering over large amounts of data
OWL 2 RL	fragment of OWL 2 designed to support rule-based reasoning
OWL 2 XML	XML-based serialization of the OWL 2 language
P-DL	Package-based description logic
PIM	Platform-independent Model
PSM	Platform-specific Model
RDF	Resource Description Framework, a graph data model
RDFS	RDF Schema
RDFa	a set of XML attributes for embedding RDF graphs into XML documents
RDF/XML	an XML serialization of the RDF data model

RIF	Rule Interchange Format
SBVR	Semantics of Business Vocabulary and Business Rules
SMOF	MOF Support for Semantic Structures
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XMI	XML Metadata Interchange
XML	eXtensible Markup Language

6. Additional Information

6.1. Changes to Adopted OMG Specifications

This specification does not require or request any change to any other OMG specification.

6.2. How to Read This Specification

The initial eight chapters of this specification are *informative* providing a high-level summary of usage scenarios and goals (Chapter 7) and an overview over the design of DOL (Chapter 8).

Chapter 9 defines the abstract syntax of DOL (*normative*) in Extended Backus–Naur Form (EBNF). Chapter 10 provides a human friendly text serialization of the abstract syntax of DOL (*normative*). Annex M contains the abstract syntax specified as an SMOF compliant meta model. (*informative*).

Chapter 11 defines the model-theoretic semantics of DOL on the abstract syntax, and also makes the notion of heterogeneous logical environment (providing languages, logics and translations) precise (*normative*).

Annex A specifies an RDF vocabulary for the terms in chapter 4, and for OMS languages and translation that conform with DOL (*normative*).

Various languages are shown to be DOL conformant in informative annexes: OWL2 (annex B), Common Logic (annex C), RDF and RDF Schema (annex D), UML class diagrams (annex E, EMOF (annex F), and CASL (annex H).

Annex I provides a core graph of logics and translations, covering those OMS languages whose conformance with DOL is established in the preceding annexes (*informative*). Annex J extends the graph presented in Annex I by a list of OMS language whose conformance with DOL will be established by a registry (*informative*).

Annex K provides of DOL texts, which provide examples for all DOL constructs, which are specified in the abstract syntax (*informative*). Annex L sketches scenarios that outline how DOL is intended to be applied. For each scenario, we list its status of implementation, the DOL features it makes use of, and provide a brief description (*informative*).

Annex N gives an overview of avaiable software tools for DOL. Annex O discusses the implementation of a linked-data compliant IRI scheme used in one of these tools (*informative*).

The bibliography contains O.5 references to the literature that is cited in this document (*informative*).

6.3. Acknowledgments

6.3.1. Submitting and supporting organizations

The following OMG members are submitting this specification:

- Fraunhofer FOKUS
- MITRE
- Thematix Partners LLC

The following organizations are supporting this specification:

- Otto-von-Guericke University Magdeburg
- Athan Services

6.3.2. Participants

The following people contributed directly to the development of this specification.

- Tara Athan, Athan Services, USA
- Conrad Bock, National Institute of Standards and Technology, USA
- Mihai Codescu, Otto-von-Guericke University Magdeburg, Germany
- Daniel Couto Vale, University of Bremen, Germany
- Martin Glauer, Otto-von-Guericke University Magdeburg, Germany
- Michael Gruninger, University of Toronto, Canada
- Stephan Günther, Otto-von-Guericke University Magdeburg, Germany
- Maria Hedblom, Otto-von-Guericke University Magdeburg, Germany
- Andreas Hoffmann, Fraunhofer FOKUS, Germany
- Yazmin Angelica Ibañez, University of Bremen
- Maria Keet, University of Cape Town, South Africa
- Elisa Kendall, Thematix Partners LLC, USA
- Alexander Knapp, University of Augsburg, Germany
- Oliver Kutz, Free University of Bolzano, Italy
- Christoph Lange, University of Bonn and Fraunhofer IAIS, Germany
- Terry Longstreth, Independent Consultant, USA
- Christian Maeder, Jacobs-University Bremen, Germany
- Till Mossakowski, Otto-von-Guericke University Magdeburg, Germany
- Fabian Neuhaus, Otto-von-Guericke University Magdeburg, Germany
- Leo Obrst, MITRE, USA
- Tim Reddehase, University of Bremen, Germany
- Madhura Thosar, Otto-von-Guericke University Magdeburg, Germany

7. Goals and Usage Scenarios

Often, engineering tasks require the use of several different OMS, which represent knowledge about a given domain or specify a given system from different perspectives or for different purposes. (E.g., a software engineer will typically use different OMS to model different aspects of a software system, including its behavior, its components, and its interactions with other systems.) Further, the OMS are often represented in different OMS languages (e.g., UML class diagrams, OWL, or Common Logic), which may differ in style, expressivity, and different computational properties.

The use of different OMS within the same context leads to several challenges in the design and deployment of OMS, which have been addressed by current research in ontological engineering, formal software specification and formal modeling:

- How can we support sharability and reusability of OMS within the same domain?
- How can we merge OMS in different domains, particularly in the cases in which the OMS are axiomatized in different logical languages?
- What notions of modularity play a role when only part of an OMS is being shared or reused?
- What are the relationships between versions of an OMS axiomatized in different logical languages?

To illustrate these challenges, in this clause we present a set of usage scenarios that involve the use of more than one OMS. These scenarios address the areas of ontology design, formal specification, and model-driven development. In spite of their many differences, they all highlight one common theme: the use of multiple OMS leads to interoperability challenges.

The purpose of DOL is to provide a standardized representation language, which can be used to represent structured OMS and the relations between OMS as part of OMS networks in a semantically well-defined way. Thus, tools that implement DOL are able to integrate different OMS into a coherent whole, thereby enabling users of DOL to overcome the different kind of interoperability issues that illustrated by the usage scenarios in this clause.

7.1. Use Case Onto-1: Interoperability Between OWL and FOL Ontologies

In order to achieve interoperability during ontology development it is often necessary to describe concepts in a language more expressive than OWL. Therefore, it is common practice to informally annotate OWL ontologies with FOL axioms (e.g., Keet's mereotopological ontology [Part-Whole], Dolce Lite [Dolce-lite], BFO-OWL). OWL is used because of better tool support, FOL because of greater expressiveness. However, relegating FOL axioms to informal annotations means that these are not available for machine processing. Another example of this problem is the following: For formally representing concept schemes (including taxonomies, thesauri and classification schemes) and provenance information there are the two W3C standards SKOS (Simple Knowledge Organization System) and PROV, as well as ISO and other domain-specific standards for metadata representation. The semantics for the SKOS and PROV languages are largely specified as OWL ontologies; however, as OWL cannot capture the full semantics, the rest is specified using some informal first-order rules. In other words, valid instance models that use SKOS or PROV may be required to satisfy both OWL and FOL axioms. When solving reasoning tasks over either SKOS or PROV ontologies, OWL reasoners are not able to consider the FOL axioms. Hence, the information contained in these axioms is lost.

DOL allows the user to replace such informal annotations by formal axioms in a suitable ontology language. The relation between the OWL ontology and the FOL axioms is that of a heterogeneous import. In the result, both the OWL and the FOL axioms are amenable to, e.g., automated consistency checks and theorem proving. Hence, all available information can be used in the reasoning process. For example, the ontology below extends the OWL definition of `isProperPartOf` as an asymmetric relation with a first-order axiom (in Common Logic) asserting that the relation is also transivitive.

```
logic CommonLogic
ontology Parthood =
ObjectProperty: isProperPartOf
  Characteristics: Asymmetric
  SubPropertyOf: isPartOf
with translation trans:SROIQtoCL
then
  (if (and (isProperPartOf x y) (isProperPartOf y z))
        (isProperPartOf x z))
```

OWL can express transitivity, but not together with asymmetry.

7.2. Use Case Onto-2: Ontology Integration by Means of a Foundational Ontology

One major use case for ontologies in industry is to achieve interoperability and data integration. However if ontologies are developed independently and used within the same domain, the differences between the ontologies may actually impede interoperability. One strategy to avoid this problem is the use of a shared foundational ontology (e.g., DOLCE or BFO), which can be used to harmonize different domain ontologies. One challenge for this approach is that foundational ontologies typically rely on expressive ontology languages (e.g., Common Logic), while domain ontologies may be represented in languages that are optimized for performance (e.g., OWL EL). For this reason, currently the role of the foundational ontology is mainly to provide a conceptual framework that may be reused by the domain ontologies; further, watered-down versions of the foundational ontologies in OWL (like DOLCE-lite or the OWL version of BFO) are used as basis for the development of domain ontologies, be this as is, in an even less expressive version (e.g., a DOLCE-lite in OWL 2 EL), or only a relevant subset thereof (e.g., only the branch of endurants). A sample orchestration of interactions between the foundational and domain ontologies in various languages is depicted in Figure 8.1 below.

DOL provides the framework for integrating different domain ontologies, aligning these to foundational ontologies [Alignment1-2] and combining the aligned ontologies into a coherent integrated ontology – even across different ontology languages. Thus, DOL enables ontology developers to utilize the complete, and most expressive, foundational ontologies for ontology integration and validation purposes.

The foundational ontology (FO) repository Repository of Ontologies for MULtiple USes (ROMULUS)[1] contains alignments between a number of foundational ontologies, expressing semantic relations between the aligned entities. We select three such ontologies, containing spatial and temporal concepts: DOLCE[2], GFO[3] and BFO[4], and present alignments between them using DOL syntax:

```
%prefix(
           gfo: <http://www.onto-med.de/ontologies/>
           dolce: <http://www.loa-cnr.it/ontologies/>
           bfo: <http://www.ifomis.org/bfo/>

        )%
logic OWL

alignment DolceLite2BFO :
  dolce:DOLCE-Lite.owl
  to
  bfo:1.1 =
 endurant = IndependentContinuant,
 physical-endurant = MaterialEntity,
 physical-object = Object,   perdurant = Occurrent,
 process = Process,          quality = Quality,
 spatio-temporal-region = SpatiotemporalRegion,
 temporal-region = TemporalRegion,  space-region = SpatialRegion

alignment DolceLite2GFO :
  dolce:DOLCE-Lite.owl to gfo:gfo.owl =
        particular = Individual, endurant = Presential,
        physical-object = Material_object, amount-of-matter = Amount_of_substrate,
        perdurant = Occurrent,  quality = Property,
        time-interval = Chronoid, generic-dependent < necessary_for,
        part < abstract_has_part, part-of < abstract_part_of,
        proper-part  <   has_proper_part,       proper-part-of  < proper_part_of,
        generic-location < occupies,    generic-location-of < occupied_by

alignment BFO2GFO :
  bfo:1.1 to gfo:gfo.owl =
        Entity = Entity, Object = Material_object,
        ObjectBoundary  = Material_boundary, Role < Role ,
        Occurrent = Occurrent,  Process = Process, Quality = Property
        SpatialRegion   = Spatial_region, TemporalRegion = Temporal_region
```

We can then combine the ontologies while taking into account the semantic dependencies given by the alignments using DOL combinations:

```
ontology Space =
 combine BFO2GFO, DolceLite2GFO, DolceLite2BFO
```

[1] See http://www.thezfiles.co.za/ROMULUS/home.html
[2] See http://www.loa.istc.cnr.it/DOLCE.html
[3] See http://www.onto-med.de/ontologies/gfo/
[4] See http://www.ifomis.org/bfo/

7.3. Use Case Onto-3: Module Extraction From Large Ontologies

Especially in the biomedical domain, ontologies tend to become very large (e.g., SNOMED CT, FMA) with over 100000 concepts and relationships. Yet, none of these ontologies covers all aspects of a domain, and frequently provide coverage at various levels of specificity, with excessive detail in some areas that may not be required for all usage scenarios. Often, for a given knowledge representation problem in industry, only relevant knowledge from two such large reference ontologies needs to be integrated, so a comprehensive integration would be both unfeasible and unwieldy. Hence, parts (modules) of these ontologies are obtained by selecting the concepts and relationships (roles) relevant for the intended application. An integrated version will then be based on these excerpts from the original ontologies (i.e., modules). For example, the Juvenile Rheumatoid Arthritis ontology JRAO has been created using modules from the NCI thesaurus and GALEN medical ontology. (See Figure 7.1) DOL supports the description of such subsets (modules) of ontologies, as well as their alignment and integration.

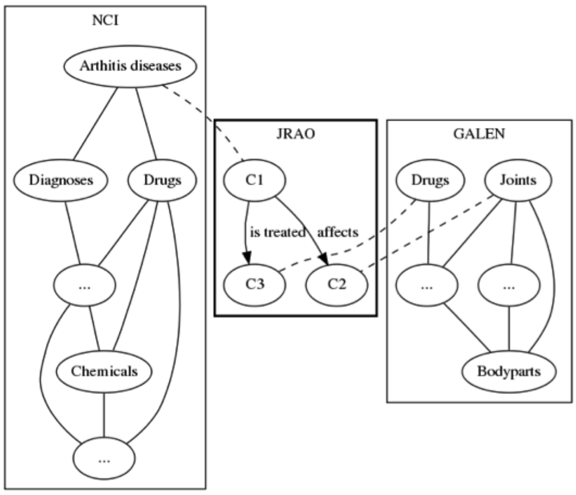


Figure 7.1.: JRAO – Example for Module Extraction

```
library GalenModule
logic OWL
ontology myGalen =
  http://purl.bioontology.org/ontology/GALEN extract Drugs, Joints, Bodyparts
end

module myGalenIsAModule : myGalen of http://purl.bioontology.org/ontology/GALEN
  for Drugs, Joints, Bodyparts
end
```

7.4. Use Case Onto-4: Interoperability Between Closed-World Data and Open-World Metadata

Data collection has become easier and much more widespread over the years. This data has to be assigned a meaning somehow, which occurs traditionally in the form of metadata annotations. For instance, consider geographical datasets derived from satellite data and raw sensor readings. Current implementations in, e.g., ecological economics[6] require manual annotation of datasets with the information relevant for their processes. While there have been attempts to standardize such information[13], metadata for datasets of simulation results are more difficult to standardize. Moreover, it is resource-consuming to link the data to the metadata, to ensure the metadata itself is of good quality and consistent, and to actually exploit the metadata when querying the data for data analysis.

The data is usually represented in a database or RDF triple store, which work with a closed world assumption on the dataset, and are not expressive enough to incorporate the metadata 'background knowledge', such as the conditions for validity of the physical laws in the model of the object of observation. These metadata require a more expressive language, such as OWL or Common Logic, which operate under an open-world semantics. However, it is unfeasible to translate the whole large dataset into OWL or first-order logic. To 'meet in the middle', it is possible to declare bridge rules (i.e., a mapping layer) that can link the metadata to the data. This approach can be used for intelligent data analysis that combines the data and metadata through querying the system. It enables the analysis of the data on the conceptual layer, instead of users having to learn the SQL/SPARQL query languages and how the data is stored. There are various tools and theories to realize this, which is collectively called Ontology-Based Data Access/Management, see also [OBDA].

The languages for representing the metadata or ontology, for representing the bridge rules or mapping assertions, and for representing the data are different yet they need to be orchestrated and handled smoothly in the system, be this for data analytics for large enterprises, for formulating policies, or in silico biology in the sciences.

DOL provides the framework for expressing such bridge rules in a systematic way, maintaining these, and building tools for them.

7.5. Use Case Onto-5: Verification of Rules Translating Dublin Core Into PROV

The Dublin Core Metadata terms, which have been formalized as an RDF Schema vocabulary, developed initially by the digital library community, are less comprehensive but more widely used than PROV (cf. Use Case Onto-1). The rules for translating Dublin Core to the OWL subset of PROV (and, with restrictions, vice versa) are not known to yield valid instances of the PROV data model, i.e. they are not known to yield OWL ontologies consistent with respect to the OWL axioms that capture part of the PROV data model. This may disrupt systems that would like to reason about the provenance of an entity, and thus the assessment of the entity's quality, reliability or trustworthiness. The Dublin Core to PROV ontology translation[5] is expressed partly by a symbol mapping and partly by FOL rules. These FOL rules are implemented by CONSTRUCT patterns in the SPARQL RDF query language.[6]

[5] `http://www.w3.org/TR/2013/NOTE-prov-dc-20130430/`

[6] E.g., `http://www.w3.org/TR/2013/NOTE-prov-dc-20130430/#dct-creator`

SPARQL has a formal specification of the evaluation semantics of its algebraic expressions, which is different from the model-theoretic semantics of the OWL and RDF Schema languages; nevertheless SPARQL CONSTRUCT is a popular and immediately executable syntax for expressing translation rules between ontologies in RDF-based languages in a subset of FOL. DOL not only supports the reuse of the existing Dublin Core RDF Schema and PROV OWL ontologies as modules of a distributed ontology (= OMS network), but it is also able to support the description of the FOL translation rules in a sufficiently expressive ontology language, e.g. Common Logic, and thus enable formal verification of the translation from Dublin Core to PROV.

7.6. Use Case Spec-1: Modularity of Specifications

Often specifications become so large that it is necessary to structure them in a modular way, for human readability and maintainability, and for more efficient tool support. The lack of a standard for such modular structuring hinders interoperability among different development efforts and the reuse of specifications. DOL provides a notion of structured modular specification that is equally applicable to all DOL-conforming logical languages.

Structuring pays off even for small specifications. For example, it makes structuring a simple specification of sorting lists in the following way enhances both readability and potential for re-use of specifications:

```
library Sorting

%% refinement from abstract sorting to insert sort

logic CASL
%right_assoc __::__
spec TotalOrder =
  sort Elem
  pred __<=__ : Elem * Elem
  forall x,y,z : Elem
  . x <= x                          %(reflexive)%
  . x <= z if x <= y /\ y <= z      %(transitive)%
  . x = y if x <= y /\ y <= x       %(antisymmetric)%
  . x <= y \/ y <= x                %(dichotomous)%
end

spec Nat =
  free type Nat ::= 0 | suc(Nat)
end

spec List =
  Nat
then
  sort Elem
  free type List ::= [] | __::__(Elem; List)
  op count : Elem * List -> Nat
  forall x,y : Elem; L : List
  . count(x,[]) = 0
  . count(x,x :: L) = suc(count(x,L))
  . count(x,y :: L) = count(x,L) if not x=y
end
```

```
spec Sorting =
  TotalOrder and List
then
  preds is_ordered : List;
        permutation : List * List
  vars x,y:Elem; L,L1,L2:List
  . is_ordered([])
  . is_ordered(x::[])
  . is_ordered(x::y::L) <=> x<=y /\ is_ordered(y::L)
  . permutation(L1,L2) <=> (forall x:Elem . count(x,L1) = count(x,L2))
then
  op sorter : List->List
  var L:List
  . is_ordered(sorter(L))
  . permutation(L,sorter(L))
hide is_ordered, permutation
end
```

In the last step, the structuring operation of hiding is used to restrict the specification to an export interface: we hide the predicates `is_ordered` and `permutation`, because they are only auxiliary and need not be implemented.

7.7. Use Case Spec-2: Specification Refinements

Formal software and hardware development methods are often used to ensure the correct function of systems which have safety-critical requirements or which may not be easily accessible for repair or replacement. Examples of such requirements can be found in safety-critical areas such as medical systems, or in the automotive, avionics and aerospace industries, as well as in components used by those industries such as in microprocessor design.

Typically, a requirement specification is refined into a design specification and then an implementation, often involving several intermediate steps (see, e.g. the V-model [V-model], although this does not require formal specification). There are numerous specification formalisms in use, including the OMG's SysML language; moreover, often during development, the formalism needs to be changed (e.g. from a specification to a programming language, or from a temporal logic to a state machine). For each of these formalisms, notions of refinement have been defined and implemented. However, the lack of a standardized, logically sound language and methodology for such refinement hinders interoperability among different development efforts and the reuse of refinements. DOL provides the capability to represent refinement that is equally applicable to all DOL-conforming logical languages, and that covers at least the most relevant of the industrial use cases of specification refinement.

A simple example is the refinement of the (purely declarative) sorting specification from use case in section 7.6 into a specification of a particular sorting algorithm (for simplicity, we choose insert sort):

```
spec InsertSort =
  TotalOrder and List
then
  ops insert : Elem*List -> List;
      insert_sort : List->List
  vars x,y:Elem; L:List
  . insert(x,[]) = x::[]
  . insert(x,y::L) = x::insert(y,L) when x<=y else y::insert(x,L)
```

```
  . insert_sort([]) = []
  . insert_sort(x::L) = insert(x,insert_sort(L))
 hide insert
end

refinement InsertSortCorrectness =
   Sorting refined via sorter |-> insert_sort to InsertSort
end
```

Note that hiding is essential here to make the signatures of both specifications compatible. If we had not hidden the predicates `is_ordered` and `permutation` in the `Sorting` specification, a refinement would not have been possible, since `InsertSort` does not implement these predicates (and it would be rather artificial to add an implementation for them).

Refinements can be composed. A simple example below illustrates this by expressing that natural numbers with addition form a monoid, and that natural numbers can be efficiently represented for implementation as lists of binary digits, together with several equivalent ways of composing these refinements.

```
spec Monoid =
 sort Elem
 ops 0 : Elem;
         __+__ : Elem * Elem -> Elem, assoc, unit 0
end

spec NatWithSuc = %mono
 free type Nat ::= 0 | suc(Nat)
 op __+__ : Nat * Nat -> Nat, unit 0
 forall x , y : Nat . x + suc(y) = suc(x + y)
 op 1:Nat = suc(0)
end

spec Nat =
  NatWithSuc hide suc
end

spec NatBin =
generated type Bin ::= 0 | 1 | __0(Bin) | __1(Bin)

ops __+__ , __++__ : Bin * Bin -> Bin
forall x, y : Bin
 .  0 0 = 0  .  0 1 = 1
 .  not  (0 = 1)  .  x 0 = y 0 => x = y .  not  (x 0 = y 1)  .  x 1 = y 1 => x = y
 .  0 + 0 = 0  .  0 ++ 0 = 1
 .  x 0 + y 0 = (x + y) 0  .  x 0 ++ y 0 = (x + y) 1
 .  x 0 + y 1 = (x + y) 1  .  x 0 ++ y 1 = (x ++ y) 0
 .  x 1 + y 0 = (x + y) 1  .  x 1 ++ y 0 = (x ++ y) 0
 .  x 1 + y 1 = (x ++ y) 0  .  x 1 ++ y 1 = (x ++ y) 1
end

refinement R2 =
 Nat refined via Nat |-> Bin to NatBin
end

refinement R3 =
 Monoid refined via Elem |-> Nat to
```

```
 Nat refined via Nat |-> Bin to NatBin
end

refinement R3' =
 Monoid refined via Elem |-> Nat to R2
end

refinement R3'' =
 Monoid refined via Elem |-> Nat to Nat then R2
end

refinement R3''' = R1 then R2
```

7.8. Use Case Model-1: Consistency Among UML Diagrams of Different Types

A typical UML model involves diagrams of different types. Such UML models may have intrinsic errors because diagrams of different types may specify conflicting requirements. Typical questions that arise in this context are, e.g.,

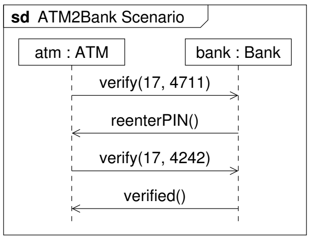
- whether the multiplicities in a class diagram are consistent with each other;
- wether the attributes and operations in a state machine are available in a class diagram;
- whether the sequential composition of actions in an interaction diagram is justified by an accompanying OCL specification;
- whether cooperating state machines comply with pre-/post-conditions and invariants;
- whether the behavior prescribed in an interaction diagram is realizable by several state machines cooperating according to a composite structure diagram.

Such questions are currently hard to answer in a systematic manner. One method to answer these questions and find such errors is a check for semantic consistency. Under some restrictions, the proof of semantic consistency can be (at least partially) performed using model-checking tools like Hugo/RT [37]. Once a formal semantics for the different diagram types has been chosen (see, e.g. [36]), it is possible to use DOL to specify in which sense the diagrams need to be consistent, and check this by suitable tools.

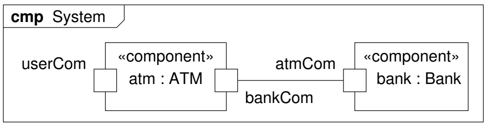
7.8.1. The ATM Example

We present as a small example in model-driven development using UML, taken from [36]. It involves the design of a traditional automatic teller machine (ATM) connected to a bank. For simplicity, we only describe the handling of entering a card and a PIN with the ATM. After entering the card, one has three trials for entering the correct PIN (which is checked by the bank). After three unsuccessful trials the card is kept.

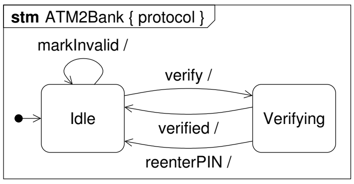
Figure 7.2(a) shows a possible *interaction* between an atm and a bank object, which consists of four messages: the atm requests the bank to verify if a card and PIN number combination is valid, in the first case the bank requests to reenter the PIN, in the second case the verification is successful. This interaction presumes that the system has an atm and a bank as objects. This can, e.g., be ensured by a *composite structure diagram*, see Fig. 7.2(b), which – among other things – specifies the objects in the initial system state. Furthermore, it specifies that the



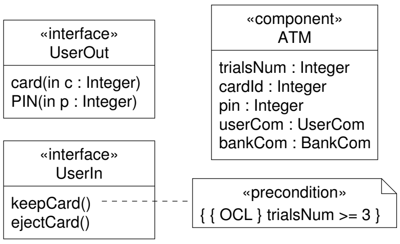
(a) Interaction



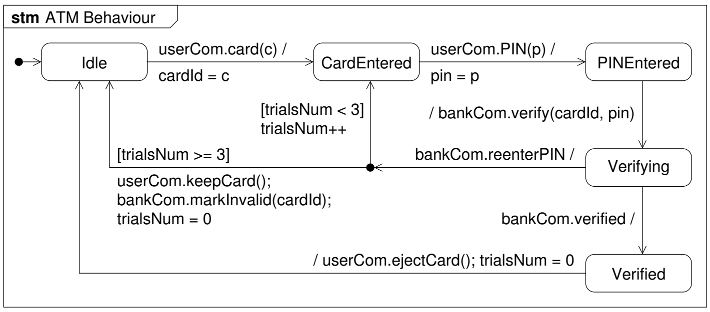
(b) Composite structure



(c) Protocol state machine



(d) Interfaces and components



(e) State machine

Figure 7.2.: ATM example

communication between atm and bank goes through the two ports bankCom and atmCom linked by a connector. The communication protocol on this connector is captured with a *protocol state machine*, see Fig. 7.2(c). The protocol state machine fixes in which order the messages verify, verified, reenterPIN, and markInvalid between atm and bank may occur. Figure 7.2(d) provides structural information in form of an interface specifying what is provided at the userCom port of the atm instance. An interface is a set of operations that other model elements have to implement. In our case, the interface is described in a *class diagram*. Here, the operation keepCard is enriched with the OCL constraint trialsNum >= 3, which refines its semantics: keepCard can only be invoked if the OCL constraints holds.

Finally, the dynamic behaviour of the atm object is specified by the *behavioural state machine* shown in Fig. 7.2(e). The machine consists of five states including Idle, CardEntered, etc. Beginning in the initial Idle state, the user can *trigger* a state change by entering the card. This has the *effect* that the parameter c from the card event is assigned to the cardId in the atm object (parameter names are not shown on triggers). Entering a PIN triggers another transition to PINEntered. Then the ATM requests verification from the bank using its bankCom port. The transition to Verifying uses a *completion event*: No explicit trigger is declared and the machine autonomously creates such an event whenever a state is completed, i.e., all internal activities of the state are finished (in our example there are no such activities). If the interaction with the bank results in reenterPIN, and the *guard* trialsNum < 3 is true, the user can again enter a PIN.

The ATM example in Fig. 7.2 consists of five different models, which naturally form a network. Coherence of this network is expressed as its consistency. We assume that XMI representations of the relevant UML models have been stored at `http://www.example.org/uml/`, that is under URL `http://www.example.org/uml/xxx.xmi`, where `xxx` is determined as follows:

Figure	xxx	diagram type
Fig. 7.2(a)	sd	sequence diagram
Fig. 7.2(b)	cmp	composite structure diagram
Fig. 7.2(c)	psm	protocol state machine
Fig. 7.2(d)	cd	class diagram
Fig. 7.2(e)	stm	state machine

```
%prefix( :      <http://www.example.org/uml/>
         uml:   <http://www.uml.org/spec/UML/>
%% descriptions of logics ...
         log:   <http://www.omg.org/spec/DOL/logics/>
library ATM

view cd2stm = cd to { atm hide along stm2cd} end
view cd2psm = cd to { psm hide along psm2cd} end
network ATM_network = %consistent
                      cd, stm, psm, cmp,
                      cd2stm, cd2psm, abstract_to_concrete_atm
entailment atm in ATM_network entails sd
network Some_refined_ATM_network = ...
refinement r = ATM_network refined to Some_refined_ATM_network
entailment e = Some_refined_ATM_network entails ATM_network
```

Here, `abstract_to_concrete_atm` is defined in the next section, and `stm2cd` and `psm2cd` are suitable logic projections extracting the classes, attributes and operations from a (protocol) state machine, delivering a class diagram.

7.9. Use Case Model-2: Refinements Between UML Diagrams of Different Types, and Their Reuse

A problem is a lack of reusability of refinements: Consider a controller for an elevator, which is specified with a UML protocol state machine, enriched with UML sequence diagrams and OCL constraints. Assume further that this model is not directly implemented, but first refined to a UML behavior state machine (which then can be automatically or semi-automatically transformed into some implementation using standard UML tools). However, there is no standardized language to express, document and maintain the refinement relation itself (UML only allows very simple refinements, namely between state machines). This hinders both the reuse of such refinements in different contexts, as well as the interoperability of tools proving such refinements to be correct. DOL addresses these problems by providing a standardized notation with formal semantics for such refinements. Refinements expressed in this language could, e.g., be parameterized and reused in different contexts.

Coming to a DOL example, we want to state the fact that the state machine of the atm, shown in Fig. 7.2(e), is a refinement of the protocol state machine in Fig. 7.2(c) as follows in DOL. We continue the library ATM above, again assuming that XMI representations of the relevant UML models have been stored at `http://www.example.org/uml/`, e.g. `http://www.example.org/uml/atm.xmi`:

```
refinement abstract_to_concrete_atm =
  psm refined via translation psm2atm to
       {  atm with Idle |-> Idle, CardEntered |-> Idle,
                   PINEntered |-> Idle, Verified |-> Idle,
                   Verifying |-> Verifying
          hide card, PIN   }
end
```

The refinement uses an abstraction of the atm, expressed by the translation via symbol map `Idle |-> Idle, CardEntered |-> Idle, PINEntered |-> Idle, Verified |-> Idle, Verifying |-> Verifying`, resulting in a two-state machine. Moreover, some detail of the atm is hidden using `hide`. Then, the protocol state machine can be refined to the thus abstracted atm.

7.10. Use Case Model-3: Coherent Semantics for Multi-Language Models

Often a single problem area within a given domain must be represented using several formalisms, e.g., because of user community requirements, expressiveness or tool support and usage. Typically the different representations are written by different people using formalisms that are based on different logics. Thus, it is a challenge to maintain consistency across the different representations. The need for the use of multiple OMS languages, even within the OMG community, is also reflected by the OMG Ontology Definition Metamodel (ODM), which provides a number of syntactic transformations between such languages. One example is the OMG Date-Time Vocabulary (DTV). DTV has been formulated in different languages, each of which addresses different audiences:

- SBVR: business users
- UML (class diagrams and OCL): software implementers
- OWL: ontology developers and users
- Common Logic: (foundational) ontology developers and users

With DOL, one can, e.g.,

- formally relate the different formalizations used for DTV, relate the different formalizations using translations,
- check consistency across the different formalizations (using suitable tools),

- extract sub-modules covering specific aspects, and
- specify the OWL version to be an approximation of the Common Logic version (using a heterogeneous interpretation of OMS).

Note that the last point does not specify what information is lost in the approximation. Indeed, DOL provides the means to specify requirements on the approximation, e.g., that it maximally preserves the information.

Coming to a DOL example, a model like the ATM model developed in section 7.8.1 typically is part of an application context that also contains some common terminology. This terminology often is specified by an ontology, and then it is desirable to relate the model to the ontology. Consider the following financial ontology fragment:

```
ontology myTaxonomy =
  ObjectProperty: owns
    Characteristics: Irreflexive, Asymmetric

  Class: FinancialIntermediary
    SubClassOf: CorporatePerson
  Class: CorporatePerson
    SubClassOf: ImmaterialEntity
  Class: ImmaterialEntity
    DisjointWith: MaterialEntity
    SubClassOf: has_part only ImmaterialEntity
  Class: Livestock
    SubClassOf: MaterialEntity
...
end
```

When we want to relate this ontology with the ATM model, we need to take care of various aspects:

- Translating into shared language (here, we choose Common Logic)
- Unifying terminology (Bank vs. FinancialIntermediary)
- Connecting related concepts (bank.owns.ATM vs. owns)
- Removing irrelevant parts (livestock)

```
model xmiStateModel = <https://ontohub.org/ATM/state.xmi>

model clStateModel = xmiStateModel with
                     translation UMLState2CL

model xmiClassModel = <https://ontohub.org/ATM/class.xmi>

model clClassModel = xmiClassModel with
            translation UMLClass2CL
            Bank |-> FinancialIntermediary

ontology BigTaxonomy = <https://ontohub.org/ATM/mytaxonmy.owl>

ontology NoLivestockTaxonomy = BigTaxonomy reject
```

```
                           Class: Livestock
                                                   end

ontology ExtendedTaxonomy = NoLivestockTaxonomy then
         ObjectProperty FinancialIntermediary.owns.ATM
           SubPropertyOf: owns
           Domain: FinancialIntermediary
           Range: ATM
end

ontology clTaxonomy = ExtendedTaxonomy with
                      translation OWL22CommonLogic

oms JointModel = clStateModel and
                 clClassModel and
                 clTaxonomy
end
```

7.11. Conclusion

In the next sections, we discuss the metalanguage DOL, its features that enable the support of a variety of formalisms, with syntax, well-defined semantics and model theory. DOL distills best practices of modularity and metarelations (such as refinement and alignment) across the three areas of ontology design, formal specification, and model-driven development. It provides the ability to specify the basis for formal interoperability even among heterogeneous OMS and OMS networks. DOL enables the solutions of the problems described in the use cases above. It also enables the development of OMS libraries, tools and workflows that allow a better exchange and reuse of OMS. Eventually, this will also lead to better, easier developed and maintained systems based on these OMS.

8. Design Overview

This clause is informative. Its purpose is to briefly describe the overall guiding principles and constraints of DOL's syntax and semantics.

We give an overview of the most important and innovative language constructs of DOL. Details can be found in clause 9.

8.1. DOL in a Nutshell

As the usage scenarios in clause 7 illustrate, the use of multiple OMS may lead to lack of interoperability. The goal of DOL is to enable users to overcome these interoperability issues by providing a language for representing structured OMS and the relations between OMS as part of an OMS network in a semantically well-defined way. One particular challenge that needs to be addressed is that OMS are written in a wide variety of OMS languages, which differ in style, expressivity and logical properties. We face this diversity not by proposing a "universal" language that is intended to subsume all the others, but by accepting this pluralism in OMS languages and by formulating means (on a sound and formal semantic basis) to compare and integrate OMS written in different formalisms. Thus, DOL is not 'yet-another-modeling language', but a meta-language that is used on top of existing OMS languages.

The major functions of DOL are the following:

- DOL allows the use of OMS in other OMS languages (e.g., UML class diagrams, CASL, OWL, Common Logic) without requiring any changes. These are called *native OMS*.
- DOL provides for defining new, *structured OMS* based on existing OMS.[1] DOL provides a number of operations for this purpose; e.g., it is possible to define a structured OMS C as the union of an OWL ontology A and a Common Logic ontology B.
- DOL provides for defining connections between two OMS by using *OMS mappings*. DOL provides a variety of mappings; e.g., one can align terminology between different OMS or specify that some OMS is an extension of another. A set of OMS and OMS mappings may form together an *OMS network*.
- Native OMS inherit their semantics from the underlying OMS languages. The DOL operations for defining structured OMS, OMS mappings, and OMS networks have a declarative model-theoretic semantics, which is defined in clause 11.

The syntax of DOL roughly follows these functions; native OMS, the various kind of structured OMS, OMS mappings, and OMS networks are the most important syntactic categories of DOL. They (together with queries and importation) form the items in a *DOL library*.

[1] Also native OMS can use the structuring constructs from their OMS language. However, these structuring constructs are often quite limited, and moreover, they differ from OMS language to OMS language.

8.2. Features of DOL

DOL is a language enabling OMS interoperability. DOL is

free DOL is freely available for unrestricted use.

generally applicable DOL is neither restricted to OMS in a specific domain, nor to foundational OMS, nor to OMS represented in a specific OMS language, nor to OMS stored in any specific repositories.

open DOL supports mapping, integrating, and annotating OMS across arbitrary internet locations. It makes use of existing open standards wherever suitable. The criteria for extending DOL (see next item) are transparent and explicit.

extensible DOL provides a framework into which any existing, and, desirably, any future OMS language can be plugged.

DOL is applicable to any OMS language that has a formal, logic-based semantics or a semantics defined by translation to another OMS language with such a formal semantics. The annotation framework of DOL is additionally applicable to the non-logical constructs of such languages. This OMG Specification specifies formal criteria for establishing the conformance of an OMS language with DOL. The annex establishes the conformance of a number of relevant OMS languages with DOL; a registry shall offer the possibility to add further (also non-standardized) languages.

DOL provides syntactic constructs for structuring OMS regardless of the logic their sentences are formalized in. Since DOL is a meta-language, it *inherits* the logical language aspects of conforming OMS languages. It is possible to literally include sentences expressed in such OMS languages in a DOL OMS.

DOL provides an initial vocabulary for expressing relations in correspondences (as part of alignments between OMS). Additionally, it provides a means of reusing relation types defined externally of this OMG Specification. DOL does not provide an annotation vocabulary, i.e. it neither provides annotation properties nor datatypes to be used with literal annotation objects.

8.3. OMS Languages

OMS languages are declarative languages for making ontological distinctions formally precise, for modeling a domain in an unambiguous way, or for expressing algebraic specifications of software. OMS languages are distinguished by the following features:

Logic Most commonly, OMS languages are based on a description logic or some other subset of first-order logic, but in some cases, higher-order, modal, paraconsistent and other logics are used.

Modularity A means of structuring an OMS into reusable parts, reusing parts of other OMS, mapping imported symbols to those in the importing OMS, and asserting additional properties about imported symbols.

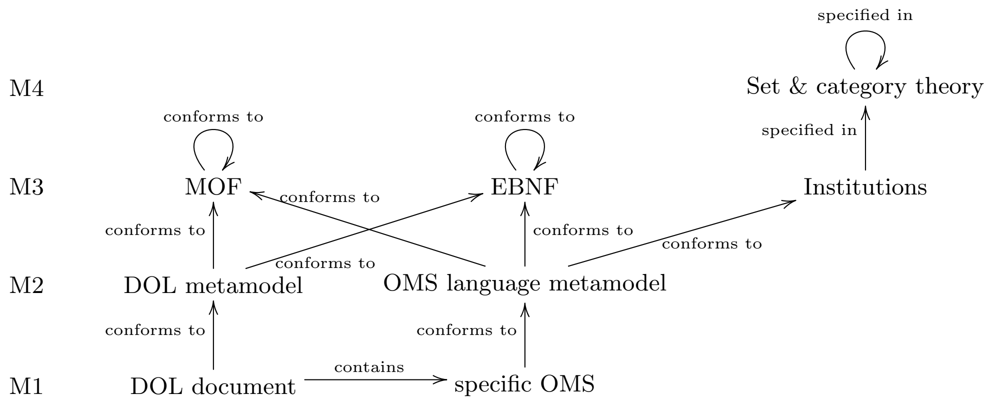
Annotation A means of enabling the attachment of human-readable descriptions to OMS symbols, addressing knowledge engineers and service developers, but also end users of OMS-based services.

Whereas the first feature determines the expressivity of the language and the possibilities for automated reasoning (decidability, tractability, etc.), the latter two facilitate OMS engineering as well as the engineering of OMS-based software.

Acknowledging the wide tool support that conforming established languages such as OWL, RDF, Common Logic, UML, MOF, or CASL enjoy, existing OMS in these (and any other) conforming languages remain as they are within the DOL framework. DOL enhances their modularity and annotation facilities to a superset of the modularity and annotation facilities they provide themselves. Using DOL's modularity constructs to make statements about modules of existing OMS works by making relevant parts of these OMS, e.g., sets of axioms, identifiable, and then referring to these identifiers from DOL statements. DOL's modularity constructs are semantically well-founded within a library of formal relationships between the logics underlying the different supported OMS languages. General annotation of OMS and their parts works in a similar way. Here, DOL does not provide its own annotation constructs, but once more DOL's general mechanism of making things of interest identifiable can be employed. Once these things have been identified, the actual annotations can be added using external mechanisms such as RDF.

8.4. DOL in the Metamodelling Hierarchy

DOL uses the metamodelling hierarchy known from model-driven engineering:



The syntax of a DOL conformant language can be written in MOF or EBNF, which are self-describing. The semantics of a DOL conformant language is its presentation as an institution. Institutions themselves are specified in the language of set theory and category theory.[2]

8.5. Semantic Foundations of DOL

A large variety of OMS languages in use can be captured at an abstract level using the concept of *institutions* [22]. This allows the development of results independently of the particularities of a logical system and to use the notions of institution and logical language interchangeably.

[2]In the future, it may be possible to specify the semantics of a DOL conformant language using a semantics-based logical framework such as MMT. This would close the loop already at M3 also for the semantics.

The main idea is to collect the non-logical symbols of the language in signatures and to assign to each signature the set of sentences that can be formed with its symbols. For each signature, we provide means for extracting the symbols it consists of, together with their kind. Institutions also provide a model theory, which introduces semantics for the language and gives a satisfaction relation between the models and the sentences of a signature.

It is also possible to complement an institution with a proof theory, introducing a derivability relation between sentences, formalized as an *entailment system* [49]. In particular, this can be done for all logics that have so far been in use in DOL.

Since institutions allow the differences between OMS languages to be elided to common abstractions, the semantics of basic OMS is presented in a uniform way. The semantics of structured OMS, OMS mappings, OMS networks, and other DOL expressions is defined using model-theoretic constructions on top of institutions.

8.6. DOL Enables Expression of Logically Heterogeneous OMS and Literal Reuse of Existing OMS

DOL is a mechanism for expressing logically heterogeneous OMS. It can be used to combine sentences and structured OMS expressed in different conforming OMS languages and logics into single documents or modules. With DOL, sentences or structured OMS of previously existing OMS in conforming languages can be reused by literally including them into a DOL OMS. A minimum of wrapping constructs and other annotations (e.g., for identifying the language of a sentence) are provided. See the abstract syntax category `OMS` in clause 9.

A heterogeneous OMS can import several OMS expressed in different conforming logics, for which suitable translations have been defined in the logic graph provided in annex I or in an extension to it that has been provided when establishing the conformance of some other logic with DOL. Determining the semantics of the heterogeneous OMS requires a translation into a common target language to be applied (cf. clause 11). This translation is determined via a lookup in the transitive closure of the logic graph. Depending on the reasoners available in the given application setting, it can, however, be necessary to employ a different translation. Authors can express which one to employ. In a multi-step translation, it is possible to implicitly apply as many default translations as possible, and to concentrate on making explicit only those translations that deviate from the default.

8.7. DOL Includes Provisions for Expressing Mappings Between OMS

DOL provides a syntax for expressing mappings between OMS. One use case illustrating both is sketched in Figure 8.1. OMS mappings supported by DOL include:

- imports (particularly including imports that lead to conservative extensions), see the abstract syntax categories `OMSRef` and `ExtensionOMS` in clause 9.
- interpretations (both between OMS and OMS networks), see the abstract syntax category `IntprDefn` in clause 9.
- alignments between OMS, see the abstract syntax category `AlignDefn` in clause 9.
- mappings between OMS and their modules, see the abstract syntax category `ModuleRelDefn` in clause 9.

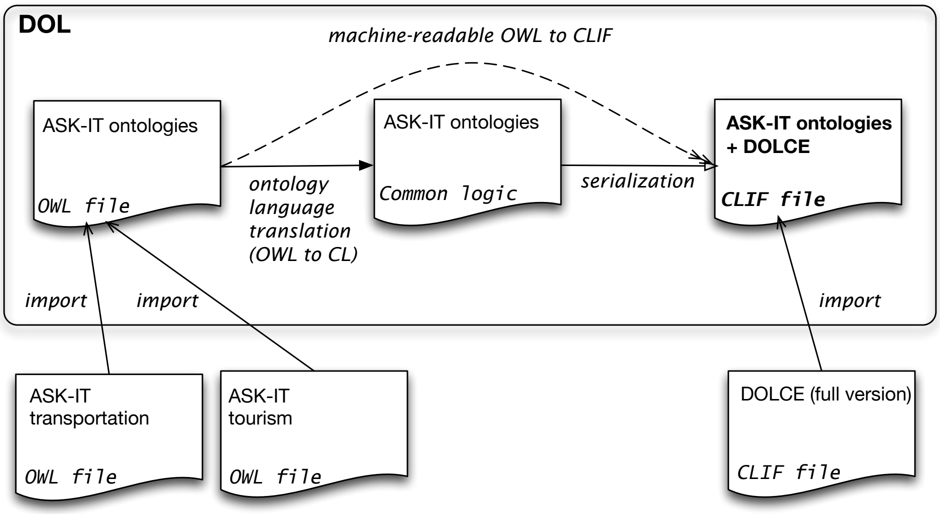


Figure 8.1.: Mapping between two OMS formulated in different OMS languages

DOL uses symbol maps to express signature translations in such OMS mappings; see the abstract syntax category `SymbolMapItems` in clause 9.

DOL need not be able to fully represent logical translations but is capable of referring to them.

DOL can also be used to combine or merge OMS along such OMS mappings, see the rule for `combination` for the abstract syntax category `OMS` in clause 9.

8.8. DOL Provides a Mechanism for Rich Annotation and Documentation of OMS

DOL provides a mechanism for identifying anything of relevance in OMS by assigning an IRI to it. With RDF there is a standard mechanism for annotating things identified by IRIs. Thus, DOL supports annotations in the full generality specified in clause 4.4.

9. DOL Abstract Syntax

9.1. Abstract Syntax Categories

DOL provides abstract syntax categories for

- OMS (which can be native OMS in some OMS language, or unions, translations, minimizations, combinations, approximations of OMS, among others)
- OMS mappings
- OMS networks
- queries
- libraries (items in libraries are: definitions of OMS, OMS mappings, OMS networks and queries, as well as qualifications choosing the logic, OMS language and/or serialization)
- identifiers
- annotations

Additionally, the categories of the abstract syntaxes of any conforming OMS languages (cf. clause 2.1) are also DOL abstract syntax categories.

The following subclauses, one per abstract syntax category, specify the abstract syntax of DOL in EBNF. Note that we deviate from the EBNF specification in ISO/IEC 14977:1996 in favor of a more modern and concise EBNF syntax.[1]

9.2. Libraries

A library (`Library`) consists of a collection of (named) OMS and mappings between these. More specifically, a library consists of a name, followed by a list of `LibraryItems`. A `LibraryItem` is either a definition of an OMS (`OMSDefn`), a mapping between OMS (`MappingDefn`), a definition of an OMS network (`NetworkDefn`), a definition related to queries (`QueryRelatedDefn`) or a `Qualification` selecting a specific OMS language, logic and/or syntax that is used to interpret the subsequent `LibraryItems`. Alternatively, a library can also be the verbatim inclusion of an OMS written in an OMS language that conforms with DOL (`NativeOMS`; cf. 2.1).

```
Library          ::= [PrefixMap] LibraryDefn | NativeOMS
LibraryDefn      ::= library LibraryName Qualification LibraryItem*
NativeOMS        ::= <language specific>
LibraryItem      ::= LibImport
                   | OMSDefn
```

[1] More precisely, ISO/IEC 14977:1996 requires commas between the (non-)terminals of a right-hand side, which we omit for the sake of better readability. Also, we replace the separator = between left and right hand-side of a rule with `::=`, and use the notation `N+` for one or more repetitions of `N`.

```
                  | NetworkDefn
                  | MappingDefn
                  | QueryRelatedDefn
                  | Qualification
LibImport        ::= lib-import LibraryName
Qualification    ::= LanguageQual | LogicQual | SyntaxQual
LanguageQual     ::= lang-select LanguageRef
LogicQual        ::= logic-select LogicRef
SyntaxQual       ::= syntax-select SyntaxRef
LibraryName      ::= IRI
PrefixMap        ::= prefix-map PrefixBinding*
PrefixBinding    ::= prefix-binding BoundPrefix
                                    IRIBoundToPrefix [Separators]
BoundPrefix      ::= bound-prefix [Prefix]
IRIBoundToPrefix ::= full-iri FullIRI
Separators       ::= separators LibraryOMSSeparator OMSSymbolSeparator
LibraryOMSSeparator ::= SeparatorString
OMSSymbolSeparator ::= SeparatorString
SeparatorString  ::= SeparatorChar SeparatorChar*
SeparatorChar    ::= ipchar | gen-delims - #< as defined in IETF/RFC 3987
>
```

At the beginning of a library, one can declare a `PrefixMap` for abbreviating long IRIs; see clause 9.7 for further details.

9.3. OMS Networks

Inside a library, one can define OMS networks (`NetworkDefn`). A `NetworkDefn` names an OMS network consisting of OMS and OMS mappings. OMS networks may build on previously-defined OMS networks, and they can be used in `combinations`.

```
NetworkDefn        ::= network-defn NetworkName [ConsStrength] Network
NetworkName        ::= IRI
Network            ::= network NetworkElements ExcludeExtensions
NetworkElements    ::= network-elements NetworkElement*
NetworkElement     ::= network-element [Id] OMSOrMappingorNetworkRef
ExcludeExtensions  ::= exclude-imports ElementRef*
ElementRef         ::= path OMSOrMappingorNetworkRef
                            OMSOrMappingorNetworkRef
                     | OMSOrMappingorNetworkRef
OMSOrMappingorNetworkRef ::= IRI
```

An OMS network by default also includes all inclusions (generated by `ExtensionOMS`) between the involved OMS—unless these are explicitly excluded.

9.4. OMS

An OMS (`OMS`) can be one of the following:

- a basic OMS `BasicOMS` written inline, in a conforming serialization of a conforming OMS language (which is defined outside this standard)[2],
- a translation of an OMS into a different signature or OMS language,
- a reduction of an OMS to a smaller signature and/or less expressive logic (that is, some non-logical symbols are hidden, but the semantic effect of sentences involving these is kept),
- a module extracted from an OMS, using a restriction signature,
- an approximation of an OMS, in a subsignature or sublogic, with the effect that sentences not expressible in the subsignature resp. sublogic are replaced with a suitable approximation,
- a filtering of an OMS, with the effect that some signature symbols and axioms are removed from the OMS,
- an extension of an OMS with a basic or a minimizable OMS, optionally named and/or marked as conservative, monomorphic, definitional, weakly definitional or implied,
- a union of several OMS (the major difference between a union and extension is that the members of the unions need to be self-contained OMS, while the extensions may reuse the signature of the extended OMS),
- a reference to an OMS existing on the Web,
- an OMS qualified with the OMS language that is used to express it,
- a combination of an OMS network (technically, this is a colimit, see [66]),
- a minimization of an OMS, forcing the subsequently declared non-logical symbols to be interpreted in a minimal way, while the non-logical symbols declared so far are fixed (alternatively, the non-logical symbols to be minimized and to be varied can be explicitly declared). Variants are maximization, freeness (minimizing also data sets and equalities on these), and cofreeness (maximizing also data sets and equalities on these),
- the application of a substitution to an OMS.

```
BasicOMS            ::= <language specific>
MinimizableOMS      ::= BasicOMS | oms-ref OMSRef [ImportName]
ExtendingOMS        ::= MinimizableOMS | MinType MinimizableOMS
OMS                 ::= ExtendingOMS
                      | minimize-symbols OMS Minimization
                      | translation OMS Translation
                      | reduction OMS Reduction
                      | module-extract OMS Extraction
                      | approximation OMS Approximation
                      | filtering OMS Filtering
                      | union OMS [ConsStrength] OMS
                      | extension OMS ExtensionOMS
                      | qual-oms Qualification* OMS
```

[2]In this place, any OMS in a conforming serialization of a conforming OMS language is permitted. However, DOL's module sublanguage should be given preference over the module sublanguage of the respective conforming OMS language; e.g. DOL's extension construct should be preferred over OWL's import construct.

```
                        | combination Network
                        | application OMS SubstName
Minimization        ::= MinType CircMin CircVars
MinType             ::= minimize | maximize | free | cofree
CircMin             ::= Symbol Symbol*
CircVars            ::= Symbol*
Translation         ::= renaming LogicTranslation* [SymbolMapItems]
LogicTranslation    ::= logic-translation OMSLangTrans
Reduction           ::= hidden LogicReduction* [SymbolItems]
                        | revealed SymbolItems
LogicReduction      ::= logic-reduction OMSLangTrans
SymbolItems         ::= symbol-items Symbol Symbol*
SymbolMapItems      ::= symbol-map-items SymbolOrMap SymbolOrMap*
Extraction          ::= extraction QualInterfaceSignature
Approximation       ::= approx [QualInterfaceSignature] [LogicRef]
Filtering           ::= select BasicOMS | reject BasicOMS
ExtensionOMS        ::= extension-oms [ExtConsStrength]
                                      [ExtensionName] ExtendingOMS
ConsStrength        ::= Conservative
                        | monomorphic
                        | weak-definitional
                        | definitional
ExtConsStrength     ::= ConsStrength | implied
Conservative        ::= consequence-conservative
                        | model-conservative
                        | not-consequence-conservative
                        | not-model-conservative
QualInterfaceSignature ::= keep-signature InterfaceSignature
                        | remove-signature InterfaceSignature
InterfaceSignature  ::= SymbolItems
ImportName          ::= IRI
ExtensionName       ::= IRI
```

An OMS definition `OMSDefn` names an OMS.

It can be optionally marked as inconsistent, consistent, monomorphic or having a unique model using `ConsStrength`.[3]. An `SymbolItems`, used in an OMS `Reduction`, is a list of non-logical symbols that are to be hidden. A `LogicReduction` denotes a logic reduction to a less expressive OMS language. A `SymbolMapItems`, used in OMS `Translations`, maps symbols to symbols, An OMS language translation `OMSLangTrans` can be either specified by its name, or be inferred as the default translation to a given target (the source will be inferred as the OMS language of the current OMS).

```
OMSDefn       ::= oms-defn OMSName [ConsStrength] OMS
Symbol        ::= IRI
```

[3]More precisely, `'consequence-conservative'` here requires the OMS to have only tautologies as signature-free logical consequences, while `'not-consequence-conservative'` expresses that this is not the case. `'model-conservative'` requires satisfiability of the OMS, `'not-model-conservative'` its unsatisfiability. `'definitional'` expresses the unique model property; this may be interesting for characterizing OMS (e.g. returned by model finders) that are used to describe single models.

```
SymbolMap      ::= symbol-map Symbol Symbol
SymbolOrMap    ::= Symbol | SymbolMap
Term           ::= <an expression specific to an OMS language>
Sentence       ::= <an expression specific to an OMS language>
OMSName        ::= IRI
OMSRef         ::= IRI
ExtensionRef   ::= IRI
LoLaRef        ::= LanguageRef | LogicRef
LanguageRef    ::= IRI
LogicRef       ::= IRI
SyntaxRef      ::= IRI
OMSLangTrans   ::= named-trans OMSLangTransRef | default-trans LoLaRef
OMSLangTransRef ::= IRI
```

9.5. OMS Mappings

An OMS mapping provides a connection between two OMS. An OMS mapping definition is the definition of either a named interpretation (`IntprDefn`, `Entailment` or `EquivDefn`), a named declaration of the relation between a module of an OMS and the whole OMS (`ModuleRelDefn`), or a named alignment (`AlignDefn`).

The `SymbolMapItems` in an interpretation always must lead to a signature morphism; a proof obligation expressing that the (translated) source OMS logically follows from the target OMS is generated. An entailment is a variant where all symbols are mapped identically, while an equivalence states that the model classes of two OMS are in bijective correspondence.

Interpretations, entailments and equivalences between OMS networks are also possible. An interpretation between OMS networks has to specify both a mapping between the nodes of the OMS network, as well as, for each node, a symbol map from the OMS of that node to the target OMS to which it is mapped.

In contrast to this functional style of mapping symbols, an alignment provides a relational connection between two OMS, using a set of `Correspondences`. Each correspondence may relate some OMS non-logical symbol to another one (possibly given by a term) with an optional confidence value. Moreover, the relation between the two non-logical symbols can be explicitly specified (like being equal, or only being subsumed) in a similar way to the Alignment API [16]. The relations that can be used in a correspondence are equivalence, disjointness, subsumption, membership (the last two with a variant for each direction) or a user-defined relation that is stored in a registry and must be prefixed with `http://www.omg.org/spec/DOL/correspondences/`. A default correspondence can be used; it is applied to all pairs of non-logical symbols with the same local names. The default relation in a correspondence is equivalence, unless a different relation is specified in a surrounding 'CorrespondenceBlock'. Using an `AlignCard`, left and right injectivity and totality of the alignment can be specified (the default is left-injective, right-injective, left-total and right-total). With `AlignSem`, different styles of networks of aligned ontologies (to be interpreted in a logic-specific way) of alignments can be specified: whether a single domain is assumed, all domains are embedded into a global domain, or whether several local domains are linked ("contextualized") by relations.

A `ModuleRelDefn` declares that a certain OMS actually is a module of some other OMS with respect to the `InterfaceSignature`.

```
MappingDefn        ::= IntprDefn
                     | Entailment
                     | EquivDefn
                     | ModuleRelDefn
                     | AlignDefn
IntprDefn          ::= intpr-defn IntprName [Conservative] IntprType
                                  LogicTranslation* [SymbolMapItems]
                     | refinement IntprName Refinement
IntprName          ::= IRI
IntprType          ::= intpr-type OMS OMS
Refinement         ::= ref-oms OMS
                     | ref-network Network
                     | ref-composition Refinement Refinement
                     | simple-oms-ref OMS RefMap Refinement
                     | simple-network-ref Network RefMap Refinement
RefMap             ::= refmap-oms [LogicTranslation] [SymbolMapItems]
                     | refmap-network NodeMap*
NodeMap            ::= node-map OMSName OMSName LogicTranslation*
                                [SymbolMapItems]
Entailment         ::= entailment EntailmentName EntailmentType
EntailmentType     ::= oms-oms-entailment OMS OMS
                     | network-oms-entailment Network OMSName OMS
                     | network-network-entailment Network Network
EntailmentName     ::= IRI
EquivDefn          ::= equiv-defn EquivName EquivType
EquivName          ::= IRI
EquivType          ::= oms-equiv OMS OMS OMS
                     | network-equiv Network Network Network
ModuleRelDefn      ::= module-defn ModuleName [Conservative]
                                   ModuleType InterfaceSignature
ModuleName         ::= IRI
ModuleType         ::= module-type OMS OMS
AlignDefn          ::= align-defn AlignName [AlignCard] AlignType
                                  AlignSem Correspondence*[4]
AlignName          ::= IRI
AlignCards         ::= AlignCardForward AlignCardBackward
AlignCardForward   ::= align-card-forward AlignCard
AlignCardBackward  ::= align-card-backward AlignCard
AlignCard          ::= injective-and-total
                     | injective
                     | total
                     | neither-injective-nor-total
AlignType          ::= align-type OMS OMS
AlignSem           ::= single-domain
                     | global-domain
```

[4]Note that this grammar uses “type” as in “the type of a function”, whereas the Alignment API uses “type” forthe totality/injectivity of the relation/function. For the latter, this grammar uses “cardinality”.

```
                     | contextualized-domain
Correspondence     ::= CorrespondenceBlock
                     | SingleCorrespondence
                     | default-correspondence
CorrespondenceBlock ::= correspondence-block [RelationRef]
                                             [Confidence] Correspondence
                                             Correspondence*
SingleCorrespondence ::= correspondence SymbolRef [RelationRef]
                                        [Confidence] TermOrSymbolRef
                                        [CorrespondenceID]
CorrespondenceID   ::= IRI
SymbolRef          ::= IRI
TermOrSymbolRef    ::= Term | SymbolRef
RelationRef        ::= subsumes
                     | is-subsumed
                     | equivalent
                     | incompatible
                     | has-instance
                     | instance-of
                     | default-relation
                     | IRI
Confidence         ::= Double

Double ::= < a number ∈ [0,1] >
```

A symbol map in an interpretation is **required** to cover all non-logical symbols of the source OMS; the semantics specification in clause 11 makes this assumption[5].

9.6. Queries

Queries are a means to extract information from an OMS. DOL's `QueryDefn`s cover "select"-type queries that deliver an answer substitution for the query variables. (Answer) substitutions can be stored separately, using a `SubstDefn`. A `ResultDefn` expresses that certain answer substitutions are the result of a query. Optionally, a result can be expressed to be complete, meaning that it comprises all answer substitutions to the query.

```
QueryRelatedDefn ::= QueryDefn | SubstDefn | ResultDefn
QueryDefn       ::= select-query-defn QueryName Vars Sentence
                                      OMS [OMSLangTrans]
SubstDefn       ::= subst-defn SubstName OMS OMS SymbolMap
ResultDefn      ::= result-def ResultName SubstName SubstName*
                               QueryName [Complete]
QueryName       ::= IRI
SubstName       ::= IRI
ResultName      ::= IRI
Vars            ::= Symbol*
Complete        ::= complete
```

[5]Mapping a non-logical symbol twice is an error. Mapping two source non-logical symbols to the same target non-logical symbol is legal, this then is a non-injective OMS mapping.

9.7. Identifiers

This section specifies the abstract syntax of identifiers of DOL OMS and their elements.

9.7.1. IRIs

In accordance with best practices for publishing OMS on the Web, identifiers of OMS and their elements **should** not just serve as *names*, but also as *locators*, which, when dereferenced, give access to a concrete representation of an OMS or one of its elements. (For the specific case of RDF Schema and OWL OMS, these best practices are documented in [29]. The latter is a specialization of the linked data principles, which apply to any machine-processable data published on the Web [41].) It is recommended that publicly accessible DOL OMS be published as linked data.

Therefore, in order to impose fewer conformance requirements on applications, DOL requires the use of IRIs for identification per IETF/RFC 3987:2005. It is **recommended** that libraries use IRIs that translate to URLs when applying the algorithm for mapping IRIs to URIs specified in IETF/RFC 3987:2005, Section 3.1. DOL descriptions of any element of a library that is identified by a certain IRI **should** be *located* at the corresponding URL, so that agents can locate them. As IRIs are specified with a concrete syntax only in IETF/RFC 3987:2005, DOL adopts the latter into its abstract syntax as well as all of its concrete syntaxes (serializations).

In accordance with semantic web best practices such as the OWL Manchester Syntax [26], this OMG Specification does not allow relative IRIs, and does not offer a mechanism for defining a base IRI, against which relative IRIs could be resolved.

Concerning these languages, note that they allow arbitrary IRIs in principle, but in practice they strongly recommend using IRIs consisting of two components [29]:

namespace an IRI that identifies an OMS,[6] usually ending with # or /

local name a name that identifies a non-logical symbol within an OMS

```
IRI     ::= full-iri FullIRI | curie CURIE⁷
FullIRI ::= < as defined by the IRI production in IETF/RFC 3987:2005 >
```

9.7.2. Abbreviating IRIs using CURIEs

As IRIs tend to be long, and as syntactic mechanisms for abbreviating them have been standardized, it is **recommended** that applications employ such mechanisms and support expanding abbreviatory notations into full IRIs. For specifying the *semantics* of DOL, this OMG Specification assumes full IRIs everywhere, but the DOL abstract *syntax* adopts CURIEs (compact URI expressions) as an abbreviation mechanism, as it is the most flexible one that has been standardized to date.

The CURIE abbreviation mechanism works by binding prefixes to IRIs. A CURIE consists of a *prefix*, which may be empty, and a *reference*. If there is an in-scope binding for the prefix, the CURIE is valid and expands into a full IRI, which is created by concatenating the IRI bound to the prefix and the reference.

DOL adopts the CURIE specification of RDFa Core 1.1 W3C/TR REC-rdfa-core:2013, Section 6 with the following changes:

[6] See annex O for a specific linked-data compliant URL scheme for DOL.
[7] specified below in clause 9.7.2

- DOL does not support the declaration of a “default prefix” mapping (covering CURIEs such as `:name`).
- DOL does support the declaration of a “no prefix” mapping (covering CURIEs such as `name`). If there is no explicit declaration for the “no prefix”, it defaults to a context-sensitive expansion mechanism, which always prepends the library IRI (in the context of a structured OMS where named OMS are referenced) resp. the current OMS IRI (in the context of a basic OMS) to a symbol name. Both the separator between the library and the OMS name and that between the OMS name and the symbol name can be declared (using the keyword `separators`), and both default to “//”.
- DOL does not make use of the `safe_curie` production.
- DOL does not allow binding a relative IRI to a prefix.
- Concrete syntaxes of DOL are encouraged but **not required** to support CURIEs.[8]

CURIEs can occur in any place where IRIs are allowed, as stated in clause 9.7.1. Informatively, we can restate the CURIE grammar supported by DOL as follows:

```
CURIE           ::= MaybeEmptyCURIE -
MaybeEmptyCURIE ::= [Prefix] RefWithoutComma
RefWithoutComma ::= Reference - StringWithComma
StringWithComma ::= UChar* ',' UChar*
UChar           ::= < any Unicode ISO/IEC 10646 character >
Prefix          ::= NCName ':'< see "NCName" in W3C/TR REC-xml-names:2009,
Section 3 >
Reference       ::= Path [Query] [Fragment]
Path            ::= ipath-absolute | ipath-rootless | ipath-empty< as de-
fined in IETF/RFC 3987 >
Query           ::= '?' iquery< as defined in IETF/RFC 3987 >
Fragment        ::= '#' ifragment< as defined in IETF/RFC 3987 >
```

Note that outside the context of a basic OMS the prefix/reference separator of a CURIE is always the colon (`:`); only for serializations of OMS languages other than DOL it may be redefined as stated in clause 2.2.

Prefix mappings can be defined at the beginning of a library (specified in clause 9.2; these apply to all parts of the library, including basic OMS as clarified in clause 9.7.3).

Bindings in a prefix map are evaluated from left to right. Authors **should not** bind the same prefix twice, but if they do, the later binding wins.

9.7.3. Mapping identifiers in basic OMS to IRIs

While DOL uses IRIs as identifiers throughout, OMS languages do not necessarily do; for example:

- OWL W3C/TR REC-owl2-syntax:2009, Section 5.5 does use IRIs.
- Common Logic ISO/IEC 24707:2007 supports them but does not enforce their use.

[8]This is a concession to having an RDF-based concrete syntax among the normative concrete syntaxes. RDFa is the only standardized RDF serialization to support CURIEs so far. Other serializations, such as RDF/XML or Turtle, support a subset of the CURIE syntax, whereas some machine-oriented serializations, including N-Triples, only support full IRIs.

- F-logic [35] does not use them at all.

However, DOL OMS mappings as well as certain operations on OMS require making unambiguous references to non-logical symbols of basic OMS (`SymbolRef`). Therefore, DOL provides a function that maps global identifiers used within basic OMS to IRIs. This mapping affects all non-logical symbol identifiers (such as class names in an OWL ontology), but not locally-scoped identifiers such as bound variables in Common Logic ontologies. DOL reuses the CURIE mechanism for abbreviating IRIs for this purpose (cf. clause 9.7.2).

The IRI of a non-logical symbol identifier in a basic OMS O is determined by the following function:

Require: D is a library
Require: O is a basic OMS in serialization S
Require: id is the identifier in question, identifying a symbol in O according to the specification of S
Ensure: i is an IRI
  if id represents a full IRI according to the specification of S **then**
    $i \leftarrow id$
  else
    {first construct a pattern cp for CURIEs in S, then match id against that pattern}
    if the declaration of DOL-conformance of S redefines the prefix/reference separator character cs (cf. clause 2.2) **then**
      $sep \leftarrow cs$
    else if S forbids prefixed CURIEs **then**
      $sep \leftarrow$ undefined
    else
      $sep \leftarrow$: {the standard CURIE separator character}
    end if
    {The following statements construct a modified EBNF grammar of CURIEs; see ISO/IEC 14977:1996 for EBNF, and clause 9.7.2 for the original grammar of CURIEs.}
    if sep is defined **then**
      $cp \leftarrow [NCName, sep], Reference$
    else
      $cp \leftarrow Reference$
    end if
    if id matches the pattern cp, where ref matches $Reference$ **then**
      if the match succeeded with a non-empty $NCName$ pn **then**
        $p \leftarrow concat(pn, :)$
      else
        $p \leftarrow$ no prefix
      end if
      if O binds p to an IRI pi according to the specification of S **then**
        $nsi \leftarrow pi$
      else
        $P \leftarrow$ the innermost prefix map in D, starting from the place of O inside D, and going up the abstract syntax tree towards the root of D
        while P is defined **do**
          if P binds p to an IRI pi **then**
            $nsi \leftarrow pi$
            break out of the **while** loop

end if
$P \leftarrow$ the next prefix map in D, starting from the place of the current P inside D, and going up the abstract syntax tree towards the root of D
end while
return an error
end if
$i \leftarrow concat(nsi, ref)$
else
return an error
end if
end if
return i

This mechanism applies to basic OMS given inline in a library document (`BasicOMS`), not to OMS in external documents (`NativeOMS`); the latter **shall** be self-contained.

While CURIEs used for identifying parts of a library (cf. clause 9.7.2) are merely syntactic sugar, the prefix map for a basic OMS is essential to determining the semantics of the basic OMS within the library. Therefore, any DOL serialization **shall** provide constructs for expressing such prefix maps, even if the serialization does not support prefix maps otherwise.

10. DOL Text Serialization

10.1. Document Type

MIME type *application/dol+text*

Filename extension .dol

10.2. Concrete Syntax

At several places, the concrete syntax uses the non-terminal `'end'` to mark the end of a definition or declaration. Tools may make this `'end'` optional. However, in this standard, we insist on the `'end'`, because it may be needed to effectively disambiguate heterogeneous texts.

10.2.1. Libraries

```
Library     ::= [PrefixMap] LibraryDefn | NativeOMS
LibraryDefn ::= 'library' LibraryName Qualification LibraryItem*
NativeOMS   ::= < language and serialization specific >
LibraryItem ::= LibImport
              | OMSDefn
              | NetworkDefn
              | MappingDefn
              | QueryRelatedDefn
              | Qualification
LibImport   ::= 'import' LibraryName
Qualification ::= LanguageQual | LogicQual | SyntaxQual
LanguageQual ::= 'language' LanguageRef
LogicQual   ::= 'logic' LogicRef
SyntaxQual  ::= 'serialization' SyntaxRef
LibraryName ::= IRI
```

```
PrefixMap      ::= '%prefix(' PrefixBinding* ')%'
PrefixBinding  ::= BoundPrefix IRIBoundToPrefix [Separators]
BoundPrefix    ::= ':' | Prefix<see definition in clause 9.7.2>
IRIBoundToPrefix ::= '<' FullIRI '>'
Separators     ::= 'separators' String String<see definition in clause 9.7.2>
```

Note that we denote the empty prefix (called "no prefix" in W3C/TR REC-rdfa-core:2013, Section 6) by a colon inside the prefix map, but completely omit it in CURIEs. This is the style of the OWL Manchester syntax [26] but differs from the RDFa Core 1.1 syntax.

10.2.2. Networks

```
NetworkDefn        ::= 'network' NetworkName '=' [ConsStrength] Network
NetworkName        ::= IRI
Network            ::= NetworkElements [ExcludeExtensions]
NetworkElements    ::= NetworkElement  ',' NetworkElement *
NetworkElement     ::= [Id ':'] OMSOrMappingorNetworkRef
ExcludeExtensions  ::= 'excluding' ElementRef  ',' ElementRef *
ElementRef         ::= IRI '->' IRI | OMSOrMappingorNetworkRef
OMSOrMappingorNetworkRef ::= IRI
Id                 ::= Letter LetterOrDigit*
```

10.2.3. OMS

While in most cases the translation from concrete to abstract syntax is obvious (the structure is largely the same), we note that both `%consistent` and `%mcons` are translated to `model-conservative`, while both `%inconsistent` and `%notmcons` are translated to `not-model-conservative`. Moreover, both `closed-world` and `minimize` are translated to `minimize`.

```
BasicOMS          ::= <language and serialization specific>
MinimizableOMS    ::= BasicOMS | OMSRef [ImportName]
ExtendingOMS      ::= MinimizableOMS
                    | MinimizeKeyword '{' MinimizableOMS '}'
OMS               ::= ExtendingOMS
                    | OMS Minimization
                    | OMS Translation
                    | OMS Reduction
                    | OMS Approximation
                    | OMS Filtering
                    | OMS 'and' [ConsStrength] OMS
                    | OMS 'then' ExtensionOMS
                    | Qualification* ':' GroupOMS
                    | 'combine' NetworkElements [ExcludeExtensions]
                    | OMS 'with' SubstName
                    | GroupOMS
Minimization      ::= MinimizeKeyword CircMin [CircVars]
MinimizeKeyword   ::= 'minimize'
                    | 'closed-world'
                    | 'maximize'
                    | 'free'
                    | 'cofree'
CircMin           ::= Symbol Symbol*
CircVars          ::= 'vars' Symbol Symbol*
```

```
GroupOMS          ::= '{' OMS '}' | OMSRef
Translation       ::= 'with' LogicTranslation* SymbolMapItems
                    | 'with' LogicTranslation LogicTranslation*
LogicTranslation ::= 'translation' OMSLangTrans
Reduction         ::= 'hide' LogicReduction* SymbolItems
                    | 'hide' LogicReduction LogicReduction*
                    | 'reveal' SymbolItems
LogicReduction    ::= 'along' OMSLangTrans
SymbolItems       ::= Symbol  ',' Symbol *
SymbolMapItems    ::= SymbolOrMap  ',' SymbolOrMap *
Extraction        ::= 'extract' InterfaceSignature
                    | 'remove' InterfaceSignature
Approximation     ::= 'forget' InterfaceSignature ['keep' LogicRef]
                    | 'keep' InterfaceSignature ['keep' LogicRef]
                    | 'keep' LogicRef
Filtering         ::= 'select' BasicOMS | 'reject' BasicOMS
ExtensionOMS      ::= [ExtConsStrength] [ExtensionName] ExtendingOMS
ConsStrength      ::= Conservative | '%mono' | '%wdef' | '%def'
ExtConsStrength   ::= ConsStrength | '%implied'
Conservative      ::= '%ccons'
                    | '%mcons'
                    | '%notccons'
                    | '%notmcons'
                    | '%consistent'
                    | '%inconsistent'
InterfaceSignature ::= SymbolItems
ImportName        ::= '%' IRI '%'
ExtensionName     ::= '%' IRI '%'
OMSkeyword        ::= 'ontology'
                    | 'onto'
                    | 'specification'
                    | 'spec'
                    | 'model'
                    | 'oms'
OMSDefn           ::= OMSkeyword OMSName '=' [ConsStrength] OMS 'end'
Symbol            ::= IRI
SymbolMap         ::= Symbol '|->' Symbol
SymbolOrMap       ::= Symbol | SymbolMap
Term              ::= <an expression specific to an OMS language>
Sentence          ::= <an expression specific to an OMS language>
OMSName           ::= IRI
OMSRef            ::= IRI
LanguageRef       ::= IRI
LogicRef          ::= IRI
SyntaxRef         ::= IRI
LoLaRef           ::= LanguageRef | LogicRef


OMSLangTrans  ::= OMSLangTransRef | '->' LoLaRef
```

```
OMSLangTransRef ::= IRI
```

The above grammar allows for some grouping ambiguity when using operators in OMS definitions. These ambiguities are resolved according to the following list, listing operators in decreasing order of precedence:

- `minimize`, `maximize`, `free`, and `cofree`.
- `extract`, `forget`, `hide`, `keep`, `reject`, `remove`, `reveal`, `select`, and `with`.
- `and`.
- `then`.

Multiple occurrences of the same operator are grouped in a left associative manner. In all other cases operators on the same precedence level are not implicitly grouped and have to be grouped explicitly. Omitting such an explicit grouping results in a parse error.

10.2.4. OMS Mappings

```
MappingDefn         ::= IntprDefn
                      | Entailment
                      | EquivDefn
                      | ModuleRelDefn
                      | AlignDefn
IntprDefn           ::= IntprKeyword IntprName [Conservative] ':'
                        IntprType 'end'
                      | IntprKeyword IntprName [Conservative] ':'
                        IntprType '=' LogicTranslation*
                        [SymbolMapItems] 'end'
                      | IntprKeyword IntprName '=' Refinement 'end'
IntprKeyword        ::= 'interpretation' | 'view' | 'refinement'
IntprName           ::= IRI
IntprType           ::= GroupOMS 'to' GroupOMS
Refinement          ::= GroupOMS
                      | NetworkName
                      | Refinement 'then' Refinement
                      | GroupOMS 'refined' [RefMap] 'to' Refinement
                      | NetworkName 'refined' [RefMap] 'to' Refinement
RefMap              ::= 'via' LogicTranslation [SymbolMapItems]
                      | 'via' [LogicTranslation] SymbolMapItems
                      | 'via' NodeMap ( ',' NodeMap )*
NodeMap             ::= OMSName '|->' OMSName
                        ['using' LogicTranslation* [SymbolMapItems]]
Entailment          ::= 'entailment' EntailmentName '='
                        EntailmentType 'end'
EntailmentName      ::= IRI
EntailmentType      ::= GroupOMS 'entails' GroupOMS
                      | OMSName 'in' Network 'entails' GroupOMS
                      | Network 'entails' Network
EquivDefn           ::= 'equivalence' EquivName ':' EquivType 'end'
```

```
EquivName           ::= IRI
EquivType           ::= GroupOMS '<->' GroupOMS '=' OMS
                      | Network '<->' Network '=' Network
ModuleRelDefn       ::= 'module' ModuleName [Conservative] ':'
                        ModuleType 'for' InterfaceSignature
ModuleName          ::= IRI
ModuleType          ::= GroupOMS 'of' GroupOMS
AlignDefn           ::= 'alignment' AlignName [AlignCards] ':'
                        AlignType 'end'
                      | 'alignment' AlignName [AlignCards] ':'
                        AlignType '=' Correspondence
                        ( ',' Correspondence )* 'assuming' AlignSem
                        'end'
AlignName           ::= IRI
AlignCards          ::= AlignCardForward AlignCardBackward
AlignCardForward    ::= AlignCard
AlignCardBackward   ::= AlignCard
AlignCard           ::= '1' | '?' | '+' | '*'
AlignType           ::= GroupOMS 'to' GroupOMS
AlignSem            ::= 'SingleDomain'
                      | 'GlobalDomain'
                      | 'ContextualizedDomain'
Correspondence      ::= CorrespondenceBlock | SingleCorrespondence | '*'
CorrespondenceBlock ::= 'relation' [RelationRef] [Confidence] '{'
                         Correspondence ( ',' Correspondence )* '}'
SingleCorrespondence ::= SymbolRef [RelationRef] [Confidence]
                          TermOrSymbolRef [CorrespondenceId]
CorrespondenceId    ::= '%(' IRI ')%'
SymbolRef           ::= IRI
TermOrSymbolRef     ::= Term | SymbolRef
RelationRef         ::= '>' | '<' | '=' | '%' | 'ni' | 'in' | IRI
Confidence          ::= Double
```

Double ::= $<$ a number $\in [0,1]$ $>$

10.2.5. Queries

```
QueryRelatedDefn ::= QueryDefn | SubstDefn | ResultDefn
QueryDefn       ::= 'query' QueryName '=' 'select' Vars 'where' Sentence
                    'in' GroupOMS ['along' OMSLangTrans] 'end'
SubstDefn       ::= 'substitution' SubstName ':' GroupOMS 'to'
                    GroupOMS '=' SymbolMapItems 'end'
ResultDefn      ::= 'result' ResultName '=' SubstName
                    ( ',' SubstName )* 'for' QueryName ['%complete']
                    'end'
QueryName       ::= IRI
SubstName       ::= IRI
```

```
ResultName     ::= IRI
Vars           ::= Symbol ( ',' Symbol )*
```

10.3. Identifiers

```
IRI            ::= '<' FullIRI '>' | CURIE
FullIRI        ::= < an IRI as defined in IETF/RFC 3987:2005 >
CURIE          ::= MaybeEmptyCURIE -
MaybeEmptyCURIE ::= [Prefix] RefWithoutComma
RefWithoutComma ::= Reference - StringWithComma
StringWithComma ::= UChar* ',' UChar*
UChar          ::= < any Unicode ISO/IEC 10646 character >
Prefix         ::= NCName ':'< see "NCName" in W3C/TR REC-xml-names:2009,
Section 3 >
Reference      ::= Path [Query] [Fragment]
Path           ::= ipath-absolute | ipath-rootless | ipath-empty< as de-
fined in IETF/RFC 3987 >
Query          ::= '?' iquery< as defined in IETF/RFC 3987 >
Fragment       ::= '#' ifragment< as defined in IETF/RFC 3987 >
```

In a CURIE without a prefix, the `reference` part is **not allowed** to match any of the keywords of the DOL syntax (cf. clause).

10.4. Lexical Symbols

The character set for the DOL text serialization is the UTF-8 encoding of Unicode ISO/IEC 10646. However, OMS can always be input in the Basic Latin subset, also known as US-ASCII.[1] For enhanced readability of OMS, the DOL text serialization particularly supports the native Unicode glyphs that represent common mathematical operators.

10.4.1. Key words and signs

The lexical symbols of the DOL text serialization include various key words and signs that occur as terminal symbols in the context-free grammar in annex 10.2. Key words and signs that represent mathematical signs are displayed as such, when possible, and those signs that are available in the Unicode character set may also be used for input.

Key words

Key words are always written lowercase. The following key words are reserved, and are not available for use as variables or as CURIEs with no prefix[2], although they can be used as parts of tokens.

```
  and end hide interpretation library logic minimize network
model onto ontology spec specification reveal then to vars view with keep
forget select reject for result substitution along where query ni in relation
```

[1]In this case, IRIs will have to be mapped to URIs following section 3.1 of IETF/RFC 3987:2005.

[2]In such a case, one can still rename affected variables, or declare a prefix binding for affected CURIEs, or use absolute IRIs instead. None of these rewritings changes the semantics.

Table 10.1.: Key Signs

Sign	Unicode Code Point	Basic Latin substitute
{	`U+007B LEFT CURLY BRACKET`	
}	`U+007D RIGHT CURLY BRACKET`	
:	`U+003A COLON`	
=	`U+003D EQUALS SIGN`	
,	`U+002C COMMA`	
↦	`U+21A6 RIGHTWARDS ARROW FROM BAR`	`|->`
→	`U+2192 RIGHTWARDS ARROW`	`->`

`assuming alignment of module equivalence entails entailment using via refined refinement oms %complete %inconsistent %consistent %notmcons %notccons %mcons %ccons %implied %def %wdef %mono remove extract translation cofree free maximize closed-world combine excluding separators %prefix(serialization language import`

Key signs

Table 10.1 following key signs are reserved, and are not available for use as complete identifiers. Key signs that are outside of the Basic Latin subset of Unicode may alternatively be encoded as a sequence of Basic Latin characters.

10.5. Integration of Serializations of Conforming Languages

Any document providing an OMS in a serialization of a DOL conforming language can be used as-is in DOL, by reference to its IRI.

The following cases apply for injecting identifiers into fragments of OMS languages, depending on the conformance level of the respective serialization of the OMS language used in terms of section 2.2:

XML conformance Identifiers are added to XML elements by using the IRI-valued *dol:id* XML attribute from the `http://www.omg.org/spec/DOL/1.0/xml` namespace, or, if the serialization does not support this attribute, by adding a *dol:id* XML element as the first child, containing exactly one text node with the IRI.

RDF conformance The RDF data model itself enables the assignment of IRI identifiers to all resources.

Text conformance Identifiers are added by inserting a special comment immediately[3] after the structural OMS element to be annotated, or, if this is not allowed and no ambiguity arises from inserting the comment *before* the structural element, by doing the latter. The complete comment **shall** read `%(I)%` if the language uses the `%` character to introduce comments, where `I` is the identifier IRI. If the language uses a different comment syntax, the *content* of the comment **shall** start with `%(I)%`, possibly preceded by whitespace.

[3]The serialization **may** allow whitespace between the keyword and the comment.

Standoff markup conformance Standard mechanisms such as XPointer (W3C/TR REC-xptr-framework:2003) or IETF/RFC 5147 shall be used as means of non-destructively assigning a URI to pieces of XML or text in the given OMS serialization.

Where the given OMS language does not provide a way of assigning IRIs to a desired subject of an annotation (e.g. if one wants to annotate an import in OWL), a library may employ RDF annotations that use XPointer or IETF/RFC 5147 as means of non-destructively referencing pieces of XML or text by URI.[4]

[4] We intend to utilise the extensibility of the XPointer framework by developing additional XPointer schemes, e.g. for pointing to subterms of Common Logic sentences.

11. DOL Semantics

DOL is a logical language with a precise formal semantics. The semantics gives DOL a rock-solid foundation, and provides increased trustworthiness in applications based on OMS written in DOL. The semantics of DOL is moreover the basis for formal interoperability, as well as for the meaningful use of logic-based tools for DOL, such as theorem provers, model-checkers, SMT solvers etc. Last but not least, the semantics has provided valuable feedback on the language design, and has led to some corrections on the abstract syntax. These reasons, plus the requirement in the OntoIOp RFP to provide a semantics, have lead to inclusion of the semantics in the standard document proper, even though the semantics is quite technical and therefore has a more limited readership than the other chapters of this standard.

The semantics starts with the theoretical foundations. Since DOL is a language that can be applied to a variety of logics and logic translations, it is based on some heterogeneous logical environment. Hence, the most important need is to capture precisely what a heterogeneous logical environment is.

The DOL semantics itself gives a formal meaning to libraries, OMS networks, OMS, OMS mappings, and queries. For each synactic construct in the abstract syntax, a *semantic domain* is given. It specifies the range of possible values for the semantics. Additionally, *semantic rules* are presented, mapping abstract syntax trees to some suitable semantic domain.

11.1. Theoretical Foundations of the DOL Semantics

We now specify the theoretical foundations of the semantics of DOL. The notions of *institution* and institution *comorphism* and *morphism* are introduced, which provide formalizations of the terms logic, resp. logic translation, resp. logic reduction.

Since DOL covers OMS written in one or several logical systems, the DOL semantics needs to clarify the notion of logical system. Traditionally, logicians have studied abstract logical systems as sets of sentences equipped with an entailment relation $\vdash$. Such an entailment relation can be generated in two ways: either via a proof system, or as the logical consequence relation for some model theory. We here follow the model-theoretic approach, since this is needed for many of the DOL constructs, and moreover, ontology, modeling and specification languages like OWL, Common Logic, or CASL come with a model-theoretic semantics, or (like UML class diagrams) can be equipped with one.

Hence, we recall the notion of satisfaction system [9], called 'rooms' in the terminology of [21]. They capture the Tarskian notion of satisfaction of a sentence in a model in an abstract way.

Definition 1 *A triple $\mathcal{R} = (Sen, \mathcal{M}, \models)$ is called a* ***satisfaction system****, or* ***room****, if $\mathcal{R}$ consists of*

- *a set Sen of* ***sentences****,*
- *a class $\mathcal{M}$ of* ***models****, and*
- *a binary relation $\models \subseteq \mathcal{M} \times Sen$, called the* ***satisfaction relation****.*

□

While this signature-free treatment enjoys simplicity and is wide-spread in the literature, many concepts and definitions found in logics, e.g. the notion of a conservative extension, involve the *vocabulary* or *signature* Σ used in sentences. Signatures can be extended with new non-logical symbols, or some of these symbols can be renamed; abstractly, this is captured using signature morphisms. Moreover, also morphisms between models are needed in order to give a semantics to `minimize`, `maximize`, `free` and `cofree`—these constructs use model morphisms to select certain models, e.g. the minimial ones. This leads to the notion of *institution*. An institution is nothing more than a family of satisfaction systems, indexed by signatures, and linked coherently by signature morphisms.

Definition 2 *An **institution** [22] is a quadruple $I = (\mathsf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models)$ consisting of the following:*

- *a category*[1] Sig *of* signatures *and* signature morphisms*,*
- *a functor* $\mathbf{Sen}\colon \mathsf{Sig} \longrightarrow \mathbb{S}et$[2] *giving, for each signature* Σ*, the set of* sentences $\mathbf{Sen}(\Sigma)$*, and for each signature morphism* $\sigma : \Sigma \to \Sigma'$*, the* sentence translation map $\mathbf{Sen}(\sigma) : \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$*, where often* $\mathbf{Sen}(\sigma)(\varphi)$ *is written as* $\sigma(\varphi)$*,*
- *a functor* $\mathbf{Mod} : \mathsf{Sig}^{op} \to \mathbb{C}at$[3] *giving, for each signature* Σ*, the category of* models $\mathbf{Mod}(\Sigma)$*, and for each signature morphism* $\sigma\colon \Sigma \longrightarrow \Sigma'$*, the* reduct functor $\mathbf{Mod}(\sigma) : \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$*, where often* $\mathbf{Mod}(\sigma)(M')$ *is written as* $M'|_\sigma$*, and* $M'|_\sigma$ *is called the* σ-reduct *of* M'*, while* M' *is called a* σ-expansion *of* $M'|_\sigma$*,*
- *a satisfaction relation* $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ *for each* $\Sigma \in |\mathsf{Sig}|$*,*

such that for each $\sigma\colon \Sigma \longrightarrow \Sigma'$ *in* Sig *the following **satisfaction condition** holds:*

$$(\star) \qquad M' \models_{\Sigma'} \sigma(\varphi) \text{ iff } M'|_\sigma \models_\Sigma \varphi$$

for each $M' \in |\mathbf{Mod}(\Sigma')|$ *and* $\varphi \in \mathbf{Sen}(\Sigma)$*, expressing that truth is invariant under change of notation and context.* □

Definition 3 (Propositional Logic) *The signatures of propositional logic are sets* Σ *of propositional symbols, and signature morphisms are just functions* $\sigma : \Sigma_1 \to \Sigma_2$ *between these sets. A* Σ*-model is a function* $M : \Sigma \to \{True, False\}$*, and the reduct of a* Σ_2*-model* M_2 *along a signature morphism* $\sigma : \Sigma_1 \to \Sigma_2$ *is the* Σ_1*-model given by the composition of* σ *with* M_2*.* Σ*-sentences are built from the propositional symbols with the usual connectives, and sentence translation is replacing the propositional symbols in* Σ *along the morphism. Finally, the satisfaction relation is defined by the standard truth-tables semantics. It is straightforward to see that the satisfaction condition holds.* □

Definition 4 (Common Logic - CL) *A common logic signature* Σ *(called vocabulary in Common Logic terminology) consists of a set of names, with a subset called the set of discourse names, and a set of sequence markers. A* Σ*-model consists of a set UR, the universe of reference, with a non-empty subset UD* $\subseteq$ *UR, the universe of discourse, and four mappings:*

[1] See [1, 46] for an introduction into category theory.
[2] $\mathbb{S}et$ is the category having all small sets as objects and functions as arrows.
[3] $\mathbb{C}at$ is the category of categories and functors. Strictly speaking, $\mathbb{C}at$ is not a category but only a so-called quasicategory, which is a category that lives in a higher set-theoretic universe.

- *rel from UR to subsets of* $UD^* = \{\langle x_1, \dots, x_n\rangle \mid x_1, \dots, x_n \in UD\}$ *(i.e., the set of finite sequences of elements of UD);*
- *fun from UR to total functions from* UD^* *into UD;*
- *int from names in* Σ *to UR, such that int(v) is in UD if and only if v is a discourse name;*
- *seq from sequence markers in* Σ *to* UD^*.

A Σ*-sentence is a first-order sentence, where predications and function applications are written in a higher-order like syntax:* $t(s)$*. Here,* t *is an arbitrary term, and* s *is a sequence term, which can be a sequence of terms* $t_1 \dots t_n$*, or a sequence marker. A predication* $t(s)$ *is interpreted by evaluating the term* t*, mapping it to a relation using rel, and then asking whether the sequence given by the interpretation* s *is in this relation. Similarly, a function application* $t(s)$ *is interpreted using fun. Otherwise, interpretation of terms and formulae is as in first-order logic. A difference to first-order logic is the presence of sequence terms (namely sequence markers and juxtapositions of terms), which denote sequences in* UD^**, with term juxtaposition interpreted by sequence concatenation. Note that sequences are essentially a non-first-order feature that can be expressed in second-order logic. For details, see [14].*

A CL *signature morphism consists of two maps between the sets of names and of sequence markers, such that the property of being a discourse name is preserved and reflected.*[4] *Model reducts leave UR, UD, rel and fun untouched, while int and seq are composed with the appropriate signature morphism component.* □

Further examples of institutions are: $\mathcal{SROIQ}(D)$, unsorted first-order logic, many-sorted first-order logic, and many others. Note that reduct is generally given by forgetting parts of the model.

For the rest of the section, we work in an arbitrary institution. A **theory** is a pair (Σ, Δ) where Σ is a signature and Δ is a set of Σ-sentences. A theory (Σ, Δ) is **consistent** if there exists a Σ-model M such that $M \models \varphi$ for $\varphi \in \Delta$. **Semantic entailment** is defined as usual: for a theory $\Delta \subseteq \mathbf{Sen}(\Sigma)$ and $\varphi \in \mathbf{Sen}(\Sigma)$, we write $\Delta \models \varphi$, if all models satisfying all sentences in Δ also satisfy φ. A **theory morphism** $\phi : (\Sigma, \Delta) \to (\Sigma', \Delta')$ is a signature morphism $\phi : \Sigma \to \Sigma'$ such that $\Delta' \models \phi(\Delta)$.

Institution comorphisms capture the intuition of encoding or embedding a logic into a more expressive one.

Definition 5 (Institution Comorphism) *An* ***institution comorphism*** *from an institution* $I = (\mathsf{Sig}^I, \mathsf{Mod}^I, \mathsf{Sen}^I, \models^I)$ *to an institution* $J = (\mathsf{Sig}^J, \mathsf{Mod}^J, \mathsf{Sen}^J, \models^J)$ *consists of a functor* $\Phi : \mathsf{Sig}^I \longrightarrow \mathsf{Sig}^J$*, and two natural transformations* $\beta : \mathsf{Mod}^J \circ \Phi \Longrightarrow \mathsf{Mod}^I$ *and* $\alpha : \mathsf{Sen}^I \Longrightarrow \mathsf{Sen}^J \circ \Phi$*, such that for each* I*-signature* Σ*, each sentence* $\varphi \in \mathsf{Sen}^I(\Sigma)$ *and each model* $M' \in \mathsf{Mod}^J(\Phi(\Sigma))$

$$M' \models^J_{\Phi(\Sigma)} \alpha_\Sigma(\varphi) \Leftrightarrow \beta_\Sigma(M') \models^I_\Sigma \varphi.$$

holds, called the ***satisfaction condition****.* □

Here, $\Phi(\Sigma)$ is the translation of the signature Σ from institution I to institution J, $\alpha_\Sigma(\varphi)$ is the translation of the Σ-sentence φ to a $\Phi(\Sigma)$-sentence, and $\beta_\Sigma(M')$ is the translation (or perhaps better: reduction) of the $\Phi(\Sigma)$-model M' to a Σ-model. Naturality of α and β means that for each signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ in I the following squares commute:

[4] That is, a name is a discourse name if and only if its image under the signature morphism is.

$$\begin{tikzcd}
\mathsf{Sen}^I(\Sigma_1) \arrow[r, "\alpha_{\Sigma_1}"] \arrow[d, "\mathsf{Sen}^I(\sigma)"'] & \mathsf{Sen}^J(\Phi(\Sigma_1)) \arrow[d, "\mathsf{Sen}^J(\Phi(\sigma))"] \\
\mathsf{Sen}^I(\Sigma_2) \arrow[r, "\alpha_{\Sigma_2}"'] & \mathsf{Sen}^J(\Phi(\Sigma_2))
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{Mod}^J(\Phi(\Sigma_2)) \arrow[r, "\beta_{\Sigma_2}"] \arrow[d, "\mathsf{Mod}^J(\Phi(\sigma))"] & \mathsf{Mod}^I(\Sigma_2) \arrow[d, "\mathsf{Mod}^I(\sigma)"] \\
\mathsf{Mod}^J(\Phi(\Sigma_1)) \arrow[r, "\beta_{\Sigma_1}"'] & \mathsf{Mod}^I(\Sigma_1)
\end{tikzcd}$$

A comorphism is:

- *faithful* if logical consequence is preserved and reflected along the comorphism:

$$\Gamma \models^I \varphi \text{ iff } \alpha(\Gamma) \models^J \alpha(\varphi)$$

- *model-expansive* if each β_Σ is surjective;
- *(weakly) exact* if for each signature morphism $\sigma \colon \Sigma_1 \longrightarrow \Sigma_2$, the naturality diagram

$$\begin{tikzcd}
\mathsf{Mod}^J(\Phi(\Sigma_2)) \arrow[r, "\beta_{\Sigma_2}"] \arrow[d, "\mathsf{Mod}^J(\Phi(\sigma))"] & \mathsf{Mod}^I(\Sigma_2) \arrow[d, "\mathsf{Mod}^I(\sigma)"] \\
\mathsf{Mod}^J(\Phi(\Sigma_1)) \arrow[r, "\beta_{\Sigma_1}"'] & \mathsf{Mod}^I(\Sigma_1)
\end{tikzcd}$$

  admits (weak) amalgamation, i.e. any for any two models $M_2 \in \mathsf{Mod}^I(\Sigma_2)$ and $M_1' \in \mathsf{Mod}^J(\Phi(\Sigma_1))$ with $M_2|_\sigma = \beta_{\Sigma_1}(M_1')$, there is a unique (not necessarily unique) $M_2' \in \mathsf{Mod}^J(\Phi(\Sigma_2))$ with $\beta_{\Sigma_2}(M_2') = M_2$ and $M_2'|_{\Phi(\sigma)} = M_1'$;
- a *subinstitution comorphism* if Φ is an embedding, each α_Σ is injective and each β_Σ is bijective[5];
- an *inclusion comorphism* if Φ and each α_Σ are inclusions, and each β_Σ is the identity.

It is known that each subinstitution comorphism is model-expansive and each model-expansive comorphism is also faithful. Faithfulness means that a proof goal $\Gamma \models^I \varphi$ in I can be solved by a theorem prover for J by just feeding the theorem prover with $\alpha(\Gamma) \models^J \alpha(\varphi)$. Subinstitution comorphism preserve the semantics of more advanced DOL structuring constructs such as renaming and hiding.

Definition 6 *Given an institution $I = (\mathsf{Sig}^I, \mathsf{Mod}^I, \mathsf{Sen}^I, \models^I)$, we can define the institution of its theories, denoted I^{th}, as follows. The category of signatures of I^{th} is the category of I-theories and I-theory morphisms, that we denote Th^I. For each theory (Σ, Δ), its sentences are just Σ-sentences in I, and its models are just Σ-models in I that satisfy the sentences in Δ, while the (Σ, Δ)-satisfaction is the Σ-satisfaction of sentences in models of I.* □

Using this notion, we can now capture logic translations that include axiomatization of parts of the syntax of the source logic into the target logic.

Definition 7 *Let $I = (\mathsf{Sig}^I, \mathsf{Mod}^I, \mathsf{Sen}^I, \models^I)$ and $J = (\mathsf{Sig}^J, \mathsf{Mod}^J, \mathsf{Sen}^J, \models^J)$ be two institutions. An **theoroidal institution comorphism** from I to J is a institution comorphism from I to J^{th}.*

□

[5]An isomorphism if model morphisms are taken into account.

Institution morphisms capture the intuition of projecting from a more expressive logic to a less expressive one.

Definition 8 (Institution Morphism) *An* ***institution morphism*** *from an institution* $I = (\mathsf{Sig}^I, \mathsf{Mod}^I, \mathsf{Sen}^I, \models^I)$ *to an institution* $J = (\mathsf{Sig}^J, \mathsf{Mod}^J, \mathsf{Sen}^J, \models^J)$ *consists of a functor* $\Phi : \mathsf{Sig}^I \longrightarrow \mathsf{Sig}^J$*, and two natural transformations* $\beta : \mathsf{Mod}^I \Longrightarrow \mathsf{Mod}^J \circ \Phi$ *and* $\alpha : \mathsf{Sen}^J \circ \Phi \Longrightarrow \mathsf{Sen}^I$*, such that for each* I*-signature* Σ*, each sentence* $\varphi \in \mathsf{Sen}^J(\Phi(\Sigma))$ *and each model* $M \in \mathsf{Mod}^I(\Sigma)$

$$M \models^I_\Sigma \alpha_\Sigma(\varphi) \Leftrightarrow \beta_\Sigma(M) \models^J_{\Phi(\Sigma)} \varphi.$$

holds, called the ***satisfaction condition****.* □

Colimits are a categorical concept providing means of combining objects interconnected by morphisms, where the colimit glues together objects along the morphisms. They can be employed for constructing larger theories from already available smaller ones, see [22].

A *network*[6] in a category C is a functor $D : G \to C$, where G is a small category[7], and can be thought of as the shape of the graph of interconnections between the objects of C selected by the functor D. A *cocone* of a network $D : G \to C$ consists of an object c of C and a family of morphisms $\alpha_i \colon D(i) \longrightarrow c$, for each object i of G, such that for each edge of the network, $e \colon i \longrightarrow i'$ we have that $D(e); \alpha_{i'} = \alpha_i$. A *colimiting cocone* (or colimit) $(c, \{\alpha_i\}_{i\in|G|})$ can be intuitively understood as a minimal cocone, i.e. has the property that for any cocone $(d, \{\beta_i\}_{i\in|G|})$ there exists a unique morphism $\gamma \colon c \longrightarrow d$ such that $\alpha_i; \gamma = \beta_i$. By dropping the uniqueness condition and requiring only that a morphism γ should exist, we obtain a *weak* colimit.

When G is the category $\bullet \longleftarrow \bullet \longrightarrow \bullet$ with 3 objects and 2 non-identity arrows, G-colimits are called *pushouts*.

A major property of colimits of specifications is *amalgamation* (also related to 'exactness' [17]). It can be intuitively explained as stating that models of given specifications can be combined to yield a uniquely determined model of a colimit specification, provided that the original models coincide on common components. Amalgamation is a common technical assumption in the study of specification semantics [57].

In the sequel, fix an arbitrary institution $I = (\mathsf{Sig}, \mathsf{Sen}, \mathsf{Mod}, \models)$.

Definition 9 *Given a network* $D \colon J \longrightarrow \mathsf{Sig}^I$*, a family of models* $\mathcal{M} = \{M_p\}_{j\in|J|}$ *is* consistent with D *(or sometimes compatible with* D*) if for each node* p *of* D*,* $M_p \in Mod(D(p))$ *and for each edge* $e : p \to q$*,* $M_p = M_q|_{D(e)}$*. A cocone* $(\Sigma, (\mu_j)_{j\in|J|})$ *over the network* $D \colon J \longrightarrow \mathsf{Sig}^I$ *is called* weakly amalgamable *if it is mapped to a weak limit by* Mod*. For models, this means that for each* D*-compatible family of models* $(M_j)_{j\in|J|}$*, there is a* Σ*-model* M*, called an amalgamation of* $(M_j)_{j\in|J|}$*, with* $M|_{\mu_j} = M_j$ $(j \in |J|)$*, and similarly for model morphisms. If this model is unique, the cocone is called* amalgamable*.* I *(or* Mod*) admits* (finite) (weak) amalgamation *if (finite) colimit cocones are (weakly) amalgamable. Finally,* I *is called* (weakly) semi-amalgamable *if it has pushouts and admits (weak) amalgamation for these.* □

[11] studies conditions for existence of weakly amalgamable cocones in a heterogeneous setting, where the network consists of signatures (or theories) in different logics. Since a network may admit more than one weakly amalgamable cocone, we assume selection operations

[6] A network is called a diagram in category theory texts. We prefer this terminology to disambiguate from UML diagrams.

[7] That is, it has a set of objects and sets of morphisms between them instead of classes.

both for the weakly amalgamable cocone of a network and for the (potentially non-unique) amalgamation of a family of models compatible with the network. This allows us to define a function *colimit* taking as argument a network of heterogeneous signatures and returning the selected weakly amalgamable cocone for the network and a function $\oplus$ taking as argument a family of models compatible with a network and returning its selected amalgamation.

We also want to be able to talk about the symbols of a signature in a formal way. For this reason, we assume that the category of signatures of an institution is an inclusive category with symbols, as defined below:

Definition 10 *An* inclusive category with symbols *is an inclusive category* $\mathbb{C}$ *equipped with a faithful functor* $|\_| : \mathbb{C} \to \mathbb{S}et$[8] *that preserves inclusions.*

11.2. Semantics of DOL Language Constructs

The semantics of DOL is based on a fixed (but in principle arbitrary) heterogeneous logical environment. The semantic domains are based on this heterogeneous logical environment. A specific heterogeneous logical environment is given in the annexes.

A heterogeneous logical environment is given by a collection of OMS languages and OMS language translations[9], a collection of institutions, institution morphisms and institution comorphisms (serving as logics, logic reductions and logic translations), and a collection of serializations. Moreover, some of the institution comorphisms are marked as default translations (but only at most one between a given source and target institution), and there is a binary supports relation between OMS languages and institutions, and a binary supports relation between OMS languages and serializations.

We assume that for each institution in the heterogeneous logical environment there is a trivial signature $\emptyset$ with model class $\mathcal{M}_\emptyset$ and such that there exists a unique signature morphism from $\emptyset$ to any signature of the institution. Moreover we assume the existence of a partial union operation on logics, denoted $\bigcup$: $L_1 \bigcup L_2 = (L, \rho_1 : L1 \to L, \rho_2 : L_2 \to L)$, when defined. Finally, some of the comorphisms are marked as default translations and some of the morphisms as default projection, with the condition that between two institutions at most one comorphism and at most one morphism is marked as default.

For each logic in the heterogeneous logical environment, we further assume the following:

- a function that turns a symbol map into a signature morphism,
- a function giving semantics of sentences
- a relativization procedure taking as argument a theory and giving as result a theory, and three procedures for translating correspondences of alignments into sentences in the logic, as needed in Section 11.2.4.
- a default language and serialization.

For each language, we assume a default logic and serialization, and for each serialization we assume a default logic and language.

We assume that for each institution, there exist (possibly partial) union and difference operations on signatures. These concepts can be captured in a categorical setting using *inclusion systems* [17]. However, inclusion systems are too strong for our purposes and therefore we will work under weaker assumptions.

[8]That is, $(\mathbb{C}, |\_|)$ is a concrete category.

[9]The terms *OMS language* and *serialization* are not defined formally. For this semantics, it suffices to know that there is a language-specific semantics of basic OMS as defined below.

Definition 11 *An* inclusive category *[23] is a category having a broad subcategory*[10] *which is a partially ordered class with finite products and coproducts, called intersection (denoted $\cap$) and union (denoted $\cup$) such that for each pair of objects A, B, $A \cup B$ is a pushout of $A \cap B$ in the category.* $\square$

A category *has pushouts which preserve inclusions* iff there exists a pushout

$$\begin{tikzcd} A \arrow[r, hook] \arrow[d] & A' \arrow[d] \\ B \arrow[r, hook] & B' \end{tikzcd}$$

for each span where one arrow is an inclusion.

A functor between two inclusive categories is inclusive if it takes inclusions in the source category to inclusions in the target category.

Definition 12 *An institution is* weakly inclusive *if*

- Sig *is inclusive and has pushouts which preserve inclusions,*
- Sen *is inclusive, and*
- *each model category have a broad subcategory of inclusions.*

$\square$

Let I be a weakly inclusive institution. We say that I *has differences*, if there is a binary operation $\setminus$ on signatures, such that for each pair of signatures Σ_1, Σ_2, we have:

1. $\Sigma_1 \setminus \Sigma_2 \subseteq \Sigma_1$
2. $(\Sigma_1 \setminus \Sigma_2) \cap \Sigma_2 = \emptyset$
3. for any Σ with the properties 1. and 2. above, $\Sigma \subseteq \Sigma_1 \setminus \Sigma_2$.

This concludes the definition of heterogeneous logical environment and the assumptions made about it.

DOL follows a model-theoretic approach on semantics: the semantics of OMS will be defined as a class of models over some signature of an institution. This is called *model-level* semantics. In some cases, but not in all, we can also define a *theory-level* semantics of an OMS as a set of sentences over some signature of an institution. The two semantics are related by the fact that, when both the model-level and the theory-level semantics of an OMS are defined, they are compatible in the sense that the class of models given by the model-level semantics is exactly the model class of the theory given by the theory-level semantics.

We will use the following unifying notation for the two semantics of an OMS O:

- the institution of O is denoted $\mathbf{Inst}(O)$,
- the signature of O is denoted $\mathsf{Sig}(O)$ (which is a signature in $\mathbf{Inst}(O)$),
- the class of models of O is denoted $\mathsf{Mod}(O)$ (which is a class of models over $\mathsf{Sig}(O)$),
- the set of axioms of O is denoted $\mathsf{Th}(O)$ (which is a set of sentences over $\mathsf{Sig}(O)$).

[10]That is, with the same objects as the original category.

Moreover, we may denote the semantics of O as a tuple $sem(O) = (I, \Sigma, \mathcal{M}, \Delta)$ when $\mathbf{Inst}(O) = I$, $\mathsf{Sig}(O) = \Sigma$, $\mathsf{Mod}(O) = \mathcal{M}$ and $\mathsf{Th}(O) = \Delta$.

The theory-level semantics of O can be undefined, and then so is $\mathsf{Th}(O)$. When $\mathsf{Th}(O)$ is defined, $\mathsf{Mod}(O)$ can be obtained as $\mathsf{Mod}(O) = \{M \in \mathsf{Mod}(\mathsf{Sig}(O)) \mid M \models \mathsf{Th}(O)\}$.

Intuitively, OMS mappings denote various types of links between two or more OMS. The semantics of OMS mappings can be captured uniformly as a graph whose nodes N are labeled with

- $\mathbf{Name}(N)$, the name of the node
- $\mathbf{Inst}(N)$, the institution of the node
- $\mathsf{Sig}(N)$, the signature of the node
- $\mathsf{Mod}(N)$, the class of $\mathsf{Sig}(N)$-models of the node
- $\mathsf{Th}(N)$, the set of $\mathsf{Sig}(N)$-sentences of the node

and which has two kinds of edges:

- import links (written using single arrows, $S \rightarrow T$)
- theorem links (written using double arrows, $S \Rightarrow T$)

both labeled with heterogeneous signature morphisms between the signatures of the source and target nodes. The theory of a node may be undefined, as in the case of OMS, and when it is defined, the class of models of that node is the class of models of $\mathsf{Th}(N)$. For brevity, we may write the label of a node as a tuple. We make the simplifying assumption that any OMS can be assigned a unique name.

The semantics of a network of OMS is a graph whose nodes are labeled like in the semantics of OMS mappings and edges are labeled with heterogeneous signature morphisms (i.e. an edge from the node S to the node T is labeled with a pair (ρ, σ) where $\rho = (\Phi, \alpha, \beta) : \mathbf{Inst}(S) \rightarrow \mathbf{Inst}(T)$ is an institution comorphism and $\sigma : \Phi(\mathsf{Sig}(S)) \rightarrow \mathsf{Sig}(T)$ is a signature morphism in $\mathbf{Inst}(T)$), The intuition is that network provide means of putting together graphs of OMS and OMS mappings and of removing sub-graphs of existing networks [11]

The semantics of OMS generally depends on a global environment Γ containing:

- a graph of imports between OMS, as in the semantics of OMS mappings but only with import links between nodes, denoted $\Gamma.imports$
- a mapping from `IRI`s to semantics of OMS, OMS mappings, OMS networks and OMS queries, that we also denote by Γ, providing access to previous definitions,
- a prefix map, denoted $\Gamma.prefix$, that stores the declared prefixes,
- a triple $\Gamma.current$ that stores the current language, logic and serialization.

If Γ is such a global environment, $\Gamma[\texttt{IRI} \mapsto \mathcal{S}]$ extends the domain of Γ with `IRI` and the newly added value of Γ in `IRI` is the semantic entity $\mathcal{S}$. $\Gamma_\emptyset$ is the empty global environment, i.e. the domain of $\Gamma_\emptyset$ is the empty set, its import graph $\Gamma.imports$ is empty, the prefix map is empty and the current triple contains the error logic together with its language and serialization. The union of two global environments Γ_1 and Γ_2, denoted $\Gamma_1 \cup \Gamma_2$, is defined only if the domains of Γ_1 and Γ_2, and of $\Gamma_1.prefix$ and $\Gamma_2.prefix$ are disjoint, and then

[11] Alternatively we could label the edges of the graph of a network with OMS mappings between the OMS labeling the nodes connected by the edge. This gives a more compact representation of the graph of the network, but it has the drawback that the subgraph giving the semantics of the mapping is not explicitly available and thus one cannot operate on it, e.g. eliminate just a node of it. Also, creating a combination of the network becomes more complicated.

$\Gamma_1 \cup \Gamma_2(\texttt{IRI}) = \begin{cases} \Gamma_1(\texttt{IRI}) & \text{if } \texttt{IRI} \in dom(\Gamma_1) \\ \Gamma_2(\texttt{IRI}) & \text{if } \texttt{IRI} \in dom(\Gamma_2) \end{cases}$, $\Gamma_1 \cup \Gamma_2.imports = \Gamma_1.imports \cup \Gamma_2.imports$, $\Gamma_1 \cup \Gamma_2.current = \Gamma_1.current$ and $\Gamma_1 \cup \Gamma_2.prefix = \Gamma_1.prefix \cup \Gamma_2.prefix$. We will write $\Gamma.\{prefix = PMap\}$ for the global environment that set the prefix map of Γ to $PMap$ and $\Gamma.\{current = (lang, logic, ser)\}$ for updating the current triple of Γ to $(lang, logic, ser)$.

We assume a *language-specific semantics* of native structured OMS, inherited from the OMS language. For a native structured OMS O in a language L, logic L' and serialization S, we denote by $sem_{(L,L',S)}(O)$ the language-specific semantics of O. We moreover assume similar language-specific semantics of a basic OMS fragment O in the context of previous declarations, denoted $sem^{(\mathcal{I},\Sigma,\mathcal{M},\Delta)}_{(L,L',S)}(O)$.

11.2.1. Semantics of Libraries

We define the semantics of DOL constructs regarding libraries.

$$\boxed{sem(\texttt{Library}) = \Gamma}$$

A library is either a list of definitions of OMS, OMS mappings and OMS networks, possibly starting with a prefix map, or an OMS in one of the languages supported by the heterogeneous logical environment.

$$sem(\texttt{PrefixMap LibraryDefn}) = \Gamma'$$

where $sem(\texttt{PrefixMap}) = PMap$, $\Gamma'' = \Gamma_\emptyset.\{prefix = PMap\}$ and $sem(\Gamma'', \texttt{LibraryDefn}) = \Gamma'$.

$$sem(\texttt{NativeOMS}) = \Gamma''$$

where $\Gamma' = \Gamma_\emptyset.\{current = L\}$, with L determined from the extension of the file containing the library and $\Gamma'' = \Gamma[\texttt{IRI} \mapsto sem_{(\Gamma'.lang, \Gamma'.logic, \Gamma'.ser)}(\texttt{NativeOMS})]$.

Note that if the OMS in the library does not conform with the logic determined by the extension of the library, $sem(\texttt{NativeOMS})$ will be undefined.

$$\boxed{sem(\Gamma, \texttt{LibraryDefn}) = \Gamma'}$$

$sem(\Gamma, \texttt{library LibraryName Qualification LibraryItem}_1 \ldots \texttt{LibraryItem}_n) = \Gamma'$
where $sem(\Gamma, \texttt{Qualification}) = \Gamma'$, $sem(\Gamma', \texttt{LibraryItem}_1) = \Gamma_1$, $sem(\Gamma_1, \texttt{LibraryItem}_2) = \Gamma_2$, ..., $sem(\Gamma_{n-1}, \texttt{LibraryItem}_n) = \Gamma'$. [12]

$$\boxed{sem(\Gamma, \texttt{LibraryItem}) = \Gamma'}$$

$$sem(\Gamma, \texttt{lib-import LibraryName}) = \Gamma \cup \Gamma'$$

where $sem(\Gamma, \texttt{LibraryName}) = \texttt{IRI}$ and $sem(\texttt{IRI}) = \Gamma'$.

Equations for `OMSDefn`, `NetworkDefn`, `MappingDefn` and `QueryRelatedDefn` are given in the next sections.

[12] Note that `LibraryName` is just discarded here. However, `LibraryName` should be the IRI of the document containing the `Library`. This is known as "linked data compliance". Tools can issue a warning (not an error), if a `Library` does not follow this practice.

$$\boxed{sem(\Gamma, \texttt{Qualification}) = \Gamma'}$$

$$sem(\Gamma, \texttt{lang-select LanguageRef}) = \Gamma'$$

where $\Gamma' = \Gamma.\{current = (\texttt{LanguageRef}, logic', ser')\}$ and
$logic = logic(\Gamma.current)$,

$$logic' = \begin{cases} logic, & \text{if } \texttt{LanguageRef} \text{ supports } logic \\ \text{default logic for } \texttt{LanguageRef}, & \text{otherwise} \end{cases}$$

$$ser' = \begin{cases} ser(\Gamma.current), & \text{if } \texttt{LanguageRef} \text{ supports } ser(\Gamma.current) \\ \text{default serialization for } \texttt{LanguageRef}, & \text{otherwise} \end{cases}$$

$$sem(\Gamma, \texttt{logic-select LogicRef}) = \Gamma'$$

where $\Gamma' = \Gamma.\{current = (lang', \texttt{LogicRef}, ser')\}$
$lang = lang(\Gamma.current)$, $ser = ser(\Gamma.current)$

$$lang' = \begin{cases} lang, & \text{if } lang \text{ supports } \texttt{LogicRef} \\ \text{the unique language supporting } \texttt{LogicRef}, & \text{otherwise} \end{cases}$$

$$ser' = \begin{cases} ser, & \text{if } ser \text{ supports } \texttt{LogicRef} \\ \text{the default serialization of } \texttt{LogicRef}, & \text{otherwise} \end{cases}$$

Note that "the unique language supporting `LogicRef`" may be undefined; in this case, the semantics of the whole `logic-select LogicRef` construct is undefined.

$$sem(\Gamma, \texttt{syntax-select SyntaxRef}) = \Gamma'$$

where $lang = lang(\Gamma.current), logic = logic(\Gamma.current)$ and
$\Gamma' = \Gamma.\{current = (lang, logic, \texttt{SyntaxRef})\}$. The semantics is defined only if $lang$ supports `SyntaxRef`.

11.2.2. Semantics of Networks

The semantics of networks of OMS is given with the help of a directed graph. Its nodes and edges are specified by the `NetworkElements`, which can be OMS, OMS mappings, or OMS networks. Intuitively, the graph of a network consists of the union of all graphs of the network elements it contains, where an OMS yields a graph with one isolated node. We make the convention that all imports in the graph $\Gamma.imports$ of the current context between nodes that are specified in the list of `NetworkElements` are also included in the graph of the network. The nodes and edges given in the `ExcludeExtensions` list are then removed from the graph of the network.

An additional `Id` can be specified for each node, with the purpose of letting the user specify a prefix in the colimit of a network for the symbols with the origin in that node that must be disambiguated.

We are going to make use of the following notations. If G is a graph, let $insert(G, \Gamma, \texttt{IRI}, \texttt{Id})$ be defined as follows:

- if `IRI` denotes an OMS in Γ, then a new node named `IRI` and labeled with $\Gamma(\texttt{IRI})$ and with `Id` is added to G, unless a node named `IRI` already exists in G, and in this case G is left unchanged,
- if `IRI` denotes an OMS mapping or a network in Γ, it denotes a graph G'. Then the result is the union of G with G'.

Similarly, the operation $remove(\Gamma, G, \texttt{OMSOrMappingorPathorNetworkRef})$ is defined as follows:

- if `OMSOrMappingorPathorNetworkRef` is an `IRI`, then
  - if `IRI` denotes an OMS in Γ, then the node labeled with `IRI` and all its incoming and outgoing edges are removed from G,
  - if `IRI` denotes an OMS mapping in Γ, then $\Gamma(\texttt{IRI})$ gives a graph G' and two nodes N_1 and N_2. Then all nodes of G' other than N_1 and N_2 and all the edges of G' are removed from G.
  - if `IRI` is a network in Γ, then all the nodes of its graph and all their incoming and outgoing edges are removed from G.
- if `OMSOrMappingorPathorNetworkRef` is an `path`, $\texttt{IRI}_1$, $\texttt{IRI}_2$, then all paths of imports in G between the nodes labeled with $\texttt{IRI}_1$ and $\texttt{IRI}_2$ are removed from G.

Finally, the operation $addImports(\Gamma, G, \texttt{NetworkElements})$ adds to G all import edges in $\Gamma.imports$ between nodes which appear in the list `NetworkElements`.

$$\boxed{sem(\Gamma, \texttt{NetworkDefn}) = \Gamma'}$$

$$sem(\Gamma, \texttt{network-defn NetworkName ConsStrength Network}) = \Gamma'$$

where $\Gamma' = \Gamma[\texttt{NetworkName} \mapsto sem(\Gamma, \texttt{Network})]$.

If `ConsStrength` is `model-conservative`, the semantics is only defined if $sem(\Gamma, \texttt{Network}) \neq \emptyset$.

If `ConsStrength` is `consequence-conservative`, the semantics is defined only if all signature-free sentences that follow from the network, see entailment of OMS by networks, are tautologies.

If `ConsStrength` is `monomorphic`, the semantics is only defined if $sem(\Gamma, \texttt{Network})$ consist of exactly one isomorphism class of families of models.

If `ConsStrength` is `weak-definitional`, the semantics is only defined if $sem(\Gamma, \texttt{Network})$ is at most a singleton.

If `ConsStrength` is `definitional`, the semantics is only defined if $sem(\Gamma, \texttt{Network})$ is a singleton.

If `ConsStrength` is `not-model-conservative`, the semantics is only defined if $sem(\Gamma, \texttt{Network}) = \emptyset$.

If `ConsStrength` is `not-consequence-conservative`, the semantics is defined only if not all signature-free sentences that follow from the network, see entailment of OMS by networks, are tautologies.

$$\boxed{sem(\Gamma, \texttt{Network}) = G}$$

$$sem(\Gamma, \texttt{network NetworkElements ExcludeExtensions}) = G'$$

where $sem(\Gamma, \texttt{NetworkElements}) = G$ and $sem(\Gamma, G, \texttt{ExcludeExtensions}) = G'$.

$$\boxed{sem(\Gamma, \texttt{NetworkElements}) = G'}$$

$$sem(\Gamma, \texttt{network-elements NetworkElement}_1 \ldots \texttt{NetworkElement}_n) = G'$$

where
$G_1 = sem(\Gamma, G_\emptyset, \texttt{NetworkElement}_1)$
$G_2 = sem(\Gamma, G_1, \texttt{NetworkElement}_2)$
...
$G_n = sem(\Gamma, G_{n-1}, \texttt{NetworkElement}_n)$,
$G' = addImports(\Gamma, G_n, \texttt{NetworkElements})$.

$$\boxed{sem(\Gamma, G, \texttt{NetworkElement}) = G'}$$

$$sem(\Gamma, G, \texttt{network-element Id IRI}) = insert(G, \Gamma, \texttt{IRI}, \texttt{Id})$$

$$\boxed{sem(\Gamma, G, \texttt{ExcludeExtensions}) = G'}$$

$$sem(\Gamma, G, \texttt{exclude-imports } \mathrm{R}_1 \ldots\ \mathrm{R}_n) = G'$$

where
$R_i = \texttt{OMSOrMappingorPathorNetworkRef}_i$ for each $i = 1, \ldots, n$
$G_1 = remove(\Gamma, G, \mathrm{R}_1)$
$G_2 = remove(\Gamma, G_1, \mathrm{R}_2)$
...
$G' = remove(\Gamma, G_{n-1}, \mathrm{R}_n)$

11.2.3. Semantics of OMS

$$\boxed{sem(\Gamma, \texttt{BasicOMS}) = (\Gamma', (I, \Sigma, \mathcal{M}, \Delta))}$$

For an OMS BasicOMS in a global environment Γ, the semantics is defined as follows:

$$sem(\Gamma, \texttt{BasicOMS}) = (\Gamma', sem_{(\Gamma.lang, \Gamma.logic, \Gamma.ser)}(\texttt{BasicOMS}))$$

where Γ' is obtained from Γ by adding to $\Gamma.imports$ a new node labeled with the name of BasicOMS and the other components as given by $sem_{(\Gamma.lang, \Gamma.logic, \Gamma.ser)}(\texttt{BasicOMS})$.

$$\boxed{sem(\Gamma, \texttt{MinimizableOMS}) = (\Gamma', (\mathcal{I}', \Sigma', \mathcal{M}', \Delta'))}$$

In the rest of this section, we will use the notation $Env(\Gamma, \texttt{OMS})$ for the global environment Γ' such that $sem(\Gamma, \texttt{OMS}) = \Gamma'$.

The semantics of a BasicOMS O has been defined above.

The semantics of $O = $ oms-ref O′ ImportName is given by

- $\mathbf{Inst}(\texttt{oms-ref O' ImportName}) = \mathbf{Inst}(\Gamma(O'))$
- $\mathsf{Sig}(\texttt{oms-ref O' ImportName}) = \mathsf{Sig}(\Gamma(O'))$
- $\mathsf{Mod}(\texttt{oms-ref O' ImportName}) = \mathsf{Mod}(\Gamma(O'))$
- $\mathsf{Th}(\texttt{oms-ref O' ImportName}) = \mathsf{Th}(\Gamma(O'))$
- $Env(\Gamma, O)$ extends the graph of imports $\Gamma.imports$ with a new node for O labeled as defined in the items above and with a new edge from O' to O named ImportName and labeled with the identity on $\mathsf{Sig}(\Gamma(O'))$.

$$\boxed{sem(\Gamma, (\Sigma, \mathcal{M}, \Delta), \texttt{ExtendingOMS}) = (\Gamma', (\mathcal{I}, \Sigma', \mathcal{M}', \Delta))}$$

The semantics for `MinimizableOMS` has been defined above.

The semantics for minimization selects the models that are minimal in the class of all models with the same interpretation for the local environment (= fixed non-logical symbols, in the terminology of circumscription).

Formally, if $O' = \texttt{minimize } O$, we have

- $\mathbf{Inst}(\texttt{minimize } O) = \mathbf{Inst}(O)$
- $\mathsf{Sig}(\texttt{minimize } O) = \mathsf{Sig}(O)$
- $\mathsf{Mod}(\texttt{minimize } O) = \{M \in \mathsf{Mod}(O) \mid M \text{ is minimal in } \{M' \in \mathsf{Mod}(O) \mid M'|_\Sigma = M|_\Sigma\}\}$
- $\mathsf{Th}(\texttt{minimize } O) = \bot$

where the semantics of the `MinimizableOMS` O is given relative to the environment Γ and the context $(\Sigma, \mathcal{M}, \Delta)$, and "minimal" is interpreted in the pre-order defined by $\Sigma_1 \leq \Sigma_2$ if there is a signature morphism $\Sigma_1 \to \Sigma_2$.

The theory-level semantics for `minimize O` cannot be defined.

$Env(\Gamma, O')$ is obtained from Γ by adding to $\Gamma.imports$ a new node labeled with $(Name(O'), \mathbf{Inst}(O'), \mathsf{Sig}(O'), \mathsf{Mod}(O'), \mathsf{Th}(O')))$ and an edge from the node of O to the node of O' labeled with the identity morphism on $\mathsf{Sig}(O')$.

The semantics of `maximize`, `free`, and `cofree` is defined similarly, only the model class differs:

- $\mathsf{Mod}(\texttt{maximize } O) = \{M \in \mathsf{Mod}(O) \mid M \text{ is maximal in } \{M' \in \mathsf{Mod}(O) \mid M'|_\Sigma = M|_\Sigma\}\}$
- $\mathsf{Mod}(\texttt{free } O) = \{M \in \mathsf{Mod}(O) \mid M \text{ is initial in } \{M' \in \mathsf{Mod}(O) \mid M'|_\Sigma = M|_\Sigma\}\}$
- $\mathsf{Mod}(\texttt{cofree } O) = \{M \in \mathsf{Mod}(O) \mid M \text{ is terminal in } \{M' \in \mathsf{Mod}(O) \mid M'|_\Sigma = M|_\Sigma\}\}$

Here, initial and terminal models are defined as in category theory: M is initial (terminal) in $\mathcal{M}$ if for each $N \in \mathcal{M}$, there is exactly one morphism $h : M \to N$ ($h : N \to M$).

$$\boxed{sem(\Gamma, \texttt{OMS}) = (\Gamma', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta))}$$

`OMS` is interpreted in a context similar to that for `MinimizableOMS`, the difference being that there is no local environment.

The semantics for `ExtendingOMS` has been defined above.

$$sem(\texttt{minimize-symbols OMS (MinType CircMin CircVars)}) = (I, \Sigma, \mathcal{M}')$$

where

$$(I, \Sigma, \mathcal{M}) = sem^M(\texttt{OMS}), \qquad \Sigma_{min} = sem(\texttt{CircMin}, \Sigma),$$
$$\Sigma_{var} = sem(\texttt{CircVars}, \Sigma), \qquad \Sigma_{fixed} = \Sigma \setminus (\Sigma_{min} \cup \Sigma_{var})$$

and

$$\mathcal{M}' = \{M \in \mathcal{M} \mid M|_{\Sigma_{min} \cup \Sigma_{fixed}} \text{ is minimal in } \{M' \in \mathcal{M}|_{\Sigma_{min} \cup \Sigma_{fixed}} \mid M'|_{\Sigma_{fixed}} = M|_{\Sigma_{fixed}}\}\}$$

The semantics of a translation $O' = \texttt{translation OMS Translation}$ is given by

- $\mathbf{Inst}(O') = J$, when $\mathbf{Inst}_{\mathsf{Sig}(\texttt{OMS})}(\texttt{Translation}) = (\Phi, \alpha, \beta) : \mathbf{Inst}(\texttt{OMS}) \to J$
- $\mathsf{Sig}(O') = \Sigma'$, when $\mathsf{Mor}_{\mathsf{Sig}(\texttt{OMS})}(\texttt{Translation}) = \sigma : \Phi(\mathsf{Sig}(\texttt{OMS})) \to \Sigma'$
- $\mathsf{Mod}(O') = \{M \in \mathsf{Mod}(\Sigma') \mid \beta_\Sigma(M|_\sigma) \in \mathsf{Mod}(\texttt{OMS})\}$
- $\mathsf{Th}(O') = \{Sen^J(\sigma)(\alpha_\Sigma(\delta)) \mid \delta \in \mathsf{Th}(\texttt{OMS})\}$. It is defined only if `OMS` is flattenable.
- $Env(\Gamma, O')$ is obtained from $\Gamma'' = Env(\Gamma, \texttt{OMS})$ by extending $\Gamma''.imports$ with a new node for O' labeled as in the items above and with a new edge from the node of O to the node of O' labeled with $((\Phi, \alpha, \beta), \sigma)$.

The semantics of a reduction $O' =$ `reduction OMS Reduction` is

- $\mathbf{Inst}(O') = J$, when $\mathbf{Inst}_{\mathsf{Sig}(\texttt{OMS})}(\texttt{Reduction}) = (\Phi, \alpha, \beta) : \mathbf{Inst}(\texttt{OMS}) \to J$
- $\mathsf{Sig}(O') = \Sigma'$, when $\mathsf{Mor}_{\mathsf{Sig}(\texttt{OMS})}(\texttt{Reduction}) = \sigma : \Sigma' \to \Phi(\mathsf{Sig}(\texttt{OMS}))$
- $\mathsf{Mod}(O') = \{\beta_\Sigma(M)|_\sigma \mid M \in \mathsf{Mod}(\texttt{OMS})\}$
- $\mathsf{Th}(O') = \bot$
- $Env(\Gamma, O')$ is obtained from $\Gamma'' = Env(\Gamma, \texttt{OMS})$ by extending $\Gamma''.imports$ with a new node for O' labeled as in the items above and with a new edge from the node of O' to the node of O labeled with $((\Phi, \alpha, \beta), \sigma)$.

The semantics of $O' =$ `module-extract OMS Extraction` is

- $\mathbf{Inst}(O') = \mathbf{Inst}(\texttt{OMS})$
- $\mathsf{Sig}(O') = \Sigma'$,
- $\mathsf{Th}(O') = \Delta'$
- $\mathsf{Mod}(O')$ is the class of $Th(O)$-models
- $Env(\Gamma, O')$ is obtained from $\Gamma'' = Env(\Gamma, \texttt{OMS})$ by extending $\Gamma''.imports$ with a new node for O' labeled as in the items above and with a new edge from the node of O' to the node of O labeled with the inclusion of Σ' in $\mathsf{Sig}(\texttt{OMS})$

where $sem(\Gamma, (\mathsf{Sig}(\texttt{OMS}), \mathsf{Th}(\texttt{OMS})), \texttt{Extraction})) = (\Sigma', \Delta')$.

The semantics of $O' =$ `approximation OMS Approximation` is

- $\mathbf{Inst}(O) = I$ when $(\Phi, \alpha, \beta) : \mathbf{Inst}(\texttt{OMS}) \to \mathcal{I})$ is the default projection (in case `LogicRef` is missing, it is the identity on $\mathbf{Inst}(\texttt{OMS})$)
- $\mathsf{Sig}(O) = \Phi(\Sigma)$
- $\mathsf{Th}(O) = \alpha^{-1}_{\mathsf{Sig}(\texttt{OMS})}(\mathsf{Th}(\texttt{OMS})^\bullet) \cap \mathsf{Sen}^I(\mathsf{Sig}(\texttt{OMS}))$ [13], i.e. that part of $\mathsf{Th}(\texttt{OMS})$ that can be expressed in the smaller signature and logic
- $\mathsf{Mod}(O)$ is the class of $\mathsf{Th}(O)$-models
- $Env(\Gamma, O')$ is obtained from $\Gamma'' = Env(\Gamma, \texttt{OMS})$ by extending $\Gamma''.imports$ with a new node for O' labeled as in the items above and with a new edge from the node of `OMS` to the node of O' labeled with $((\Phi, \alpha, \beta), \iota : \Phi(\Sigma) \to \mathsf{Sig}(\texttt{OMS}))$

where $(\mathcal{I}, \Sigma) = sem(\Gamma, (\mathbf{Inst}(\texttt{OMS}, \mathsf{Sig}(\texttt{OMS})), \texttt{Approximation})$.

The semantics of $O =$ `filtering OMS Filtering` is defined only if $\mathsf{Sig}(\texttt{Filtering}) \subseteq \mathsf{Sig}(\texttt{OMS})$ and $\mathsf{Th}(\texttt{Filtering}) \subseteq \mathsf{Th}(\texttt{OMS})$. We distinguish two cases based on the value of c, where $sem(\Gamma, (\mathsf{Sig}(\texttt{OMS}), \mathsf{Th}(\texttt{OMS}), \texttt{Filtering}) = (c, \mathcal{I}, \Sigma, \Delta)$ If $c = select$, the semantics of O is given by

[13] In practice, one looks for a finite subset that still is logically equivalent to this set. Note that $\Delta^\bullet$ is the set of logical consequences of Δ, i.e. $\Delta^\bullet = \mathbf{Th}(\Delta)$.

- $\mathbf{Inst}(O) = \mathcal{I}$
- $\mathsf{Sig}(O) = \Sigma'$ where Σ' is the smallest signature with $\Sigma \subseteq \Sigma'$ and $\Delta \subseteq \mathsf{Sen}(\Sigma')$[14]
- $\mathsf{Th}(O) = (\mathsf{Th}(\texttt{OMS}) \cap \mathsf{Sen}(\mathsf{Sig}(O))) \cup \Delta$
- $\mathsf{Mod}(O)$ is the class of all $\mathsf{Th}(O)$-models.
- $Env(\Gamma, O')$ is obtained from $\Gamma'' = Env(\Gamma, \texttt{OMS})$ by extending $\Gamma''.imports$ with a new node for O labeled as in the items above and with a new edge from the node of O to the node of `OMS` labeled with the inclusion of Σ' in $\mathsf{Sig}(\texttt{OMS})$.

If $c = reject$, the semantics of $O =$ `filtering OMS reject BasicOMS` is

- $\mathbf{Inst}(O) = \mathcal{I}$
- $\mathsf{Sig}(O) = \mathsf{Sig}(\texttt{OMS}) \setminus \Sigma$
- $\mathsf{Th}(O) = \mathsf{Th}(\texttt{OMS}) \cap \mathsf{Sen}(\mathsf{Sig}(O)) \setminus \Delta$
- $\mathsf{Mod}(O)$ is the class of all $\mathsf{Th}(O)$-models.
- $Env(\Gamma, O')$ is obtained from $\Gamma'' = Env(\Gamma, \texttt{OMS})$ by extending $\Gamma''.imports$ with a new node for O labeled as in the items above and with a new edge from the node of O to the node of `OMS` labeled with the inclusion of Σ' in $\mathsf{Sig}(\texttt{OMS})$.

The semantics of $O =$ `union` $\texttt{OMS}_1$ `ConsStrength` $\texttt{OMS}_2$ is

- $\mathbf{Inst}(O) = I$ where $\mathbf{Inst}(\texttt{OMS}_1) \bigcup \mathbf{Inst}(\texttt{OMS}_2) = (I, (\Phi_1, \alpha_1, \beta_1) : \mathbf{Inst}(\texttt{OMS}_1) \to I, (\Phi_2, \alpha_2, \beta_2) : \mathbf{Inst}(\texttt{OMS}_2) \to I)$
- $\mathsf{Sig}(O) = \Phi_1(\mathsf{Sig}(\texttt{OMS}_1)) \cup \Phi_2(\mathsf{Sig}(\texttt{OMS}_2))$
- $\mathsf{Mod}(O) = \{M \in \mathsf{Mod}(\mathsf{Sig}(O)) \mid \beta_{\Sigma_i}(M|_{\Phi_i(\mathsf{Sig}(\texttt{OMS}_i))}) \in \mathsf{Mod}(\texttt{OMS}_i),\ \text{for } i = 1, 2\}$
- $\mathsf{Th}(O) = \alpha_1(\mathsf{Th}(\texttt{OMS}_1)) \cup \alpha_2(\mathsf{Th}(\texttt{OMS}_2))$.
- $Env(\Gamma, O')$ is obtained from $\Gamma'' = Env(Env(\Gamma, \texttt{OMS}_1), \texttt{OMS}_2)$ by extending $\Gamma''.imports$ with a new node for O labeled as in the items above and with edges from the nodes of $\texttt{OMS}_1$ and $\texttt{OMS}_2$, respectively, to the node of O, labeled for each $i = 1, 2$ with $((\Phi_i, \alpha_i, \beta_i, \iota_i : \Phi_i(\texttt{OMS}_i) \to \mathsf{Sig}(O))$.

If `ConsStrength` is present, then O must be a conservative extension of the appropriate strength of $\texttt{OMS}_1$.

The semantics of $O =$ `extension OMS ExtensionOMS` is

- $\mathbf{Inst}(O) = \mathbf{Inst}(\texttt{OMS}) = \mathbf{Inst}(\texttt{ExtensionOMS})$ (which means that the instutions of `OMS` and `ExtensionOMS` must be the same)
- $\mathsf{Sig}(O) = \mathsf{Sig}(\texttt{OMS}) \cup \mathsf{Sig}((\mathbf{Inst}(\texttt{OMS}), \mathsf{Sig}(\texttt{OMS}, \mathsf{Mod}(\texttt{OMS}), \mathsf{Th}(\texttt{OMS}))), \texttt{ExtensionOMS})$
- $\mathsf{Mod}(O) = \{M \in \mathsf{Mod}(\mathsf{Sig}(O)) \mid M|_{\mathsf{Sig}(\texttt{OMS})} \in \mathsf{Mod}(\texttt{OMS}) \text{ and } M|_{\mathsf{Sig}(\texttt{ExtensionOMS})} \in \mathsf{Mod}(\texttt{ExtensionOMS})\}$
- $\mathsf{Th}(O) = \mathsf{Th}(\texttt{OMS}) \cup \mathsf{Th}(\texttt{ExtensionOMS})$
- $Env(\Gamma, O')$ is obtained from $\Gamma'' = Env(\Gamma, \texttt{OMS})$ by extending $\Gamma''.imports$ with a new node for O labeled as in the items above and with a new edge from the node of `OMS` to the node of O labeled with the inclusion of $\mathsf{Sig}(\texttt{OMS})$ in $\mathsf{Sig}(O)$.

[14] If this smallest signature does not exist, the semantics is undefined.

The semantics of `qual-oms Qualification OMS` in the context Γ is the same as the semantics of `OMS` in the context Γ' given by the semantics of `Qualification` in the context Γ. The change of context is local to `OMS`, which means that if the qualification appears as a term in a larger expression, after its analysis the context will be Γ and not Γ'.

The semantics of $O =$ `combination Network` is

- $\mathbf{Inst}(O) = I$,
- $\mathsf{Sig}(O) = \Sigma$, where $(I, \Sigma, \{\mu_i\}_{i \in |G|})$ is the colimit of the graph G given by the semantics of `Network`,
- $\mathsf{Th}(O) = \cup_{i \in |G|} \mu_i(\mathsf{Th}(O_i))$, where O_i is the OMS label of the node i in G
- $\mathsf{Mod}(O) = \{M \in \mathsf{Mod}(\Sigma) \mid M|_{\mu_i} \in \mathsf{Mod}(O_i), i \in |G|\}$, where O_i is the OMS label of the node i in G.
- $Env(\Gamma, O)$ is obtained from Γ by adding to $\Gamma.imports$ a new node for O labeled as in the items above and with edges from each node in G to this new node labeled with the morphisms μ_i for each $i \in |G|$.

For the semantics of `application OMS SubstName`, see section 11.2.5.

$$\boxed{sem(\Gamma, \Sigma, \texttt{Translation}) = ((\Phi, \alpha, \beta), \sigma)}$$

The semantics of a translation $O =$ `renaming LogicTranslation SymbolMapItems` is given by

- $\mathbf{Inst}(O) = sem(\texttt{LogicTranslation}) : \Gamma.logic \to logic'$
- $\mathsf{Mor}(O) = sem(\Gamma.\{current = (lang', logic', ser')\}, \Phi(\Sigma), \texttt{SymbolMapItems})$

where $lang'$ and ser' are the default language and serialization for logic $logic'$. If `LogicTranslation` is missing, it defaults to the identity comorphism of the current logic.

$$\boxed{sem(\Gamma, \texttt{LogicTranslation}) = (\Phi, \alpha, \beta)}$$

$sem(\Gamma, \texttt{logic-translation IRI}) = (\Phi, \alpha, \beta)$, where (Φ, α, β) is the institution comorphism named by `IRI` in the heterogeneous logical environment.

$$\boxed{sem(\Gamma, \texttt{LogicTranslation}\star) = (\Phi, \alpha, \beta)}$$

$sem(\Gamma, \texttt{logic-translation IRI}_1, \ldots, \texttt{logic-translation IRI}_n) = (\Phi, \alpha, \beta)$, where $sem(\Gamma, \texttt{logic-translation IRI}_i) = (\Phi_i, \alpha_i, \beta_i)$ for $i = 1, \ldots, n$ and $(\Phi, \alpha, \beta) = (\Phi_1, \alpha_1, \beta_1); \ldots; (\Phi_n, \alpha_n, \beta_n)$.

$$\boxed{sem(\Gamma, \Sigma, \texttt{Reduction}) = ((\Phi, \alpha, \beta), \sigma)}$$

The semantics of a reduction $O =$ `hidden LogicReduction SymbolItems` is given by

- $\mathbf{Inst}(O) = sem(\texttt{LogicReduction}) : \Gamma.logic \to logic'$
- $\mathsf{Mor}(O) = \iota : \Sigma' \to \Phi(\Sigma)$, where $\Sigma' = sem(\Gamma.\{current = (lang', logic', ser')\}, \Phi(\Sigma), \texttt{SymbolItems})$, $lang'$ and ser' are the default language and serialization for logic $logic'$ and ι is the inclusion morphism.

If `LogicReduction` is missing, it defaults to the identity morphism of the current logic of Γ.

The semantics of a reduction $O =$ `revealed SymbolItems` is

- **Inst**(O) is the identity morphism on the current logic of Γ
- $\mathsf{Mor}(O)$ is the inclusion of $sem(\Gamma, \Sigma, \texttt{SymbolItems})$ in Σ.

$$\boxed{sem(\Gamma, L, \texttt{LogicReduction}) = (\Phi, \alpha, \beta)}$$

$sem(\Gamma, \texttt{logic-reduction IRI}) = (\Phi, \alpha, \beta)$, where (Φ, α, β) is the institution morphism named by `IRI` in the heterogeneous logical environment.

$$\boxed{sem(\Gamma, \Sigma, \texttt{SymbolItems}) = \Sigma'}$$

$$sem(\Gamma, \Sigma, \texttt{symbol-items Symbol}_1 \ \ldots \ \texttt{Symbol}_n) = \Sigma'$$

where Σ' is the smallest sub-signature of Σ containing $sem(\Gamma, \Sigma, \texttt{Symbol}_1), \ldots, sem(\Gamma, \Sigma, \texttt{Symbol}_n)$, if such a sub-signature exists and is otherwise undefined.

$$\boxed{sem(\Gamma, \Sigma, \Sigma', \texttt{SymbolMapItems}) = \sigma : \Sigma \to \Sigma'}$$

$$sem(\Gamma, \Sigma, \Sigma', \texttt{symbol-map-items SymbolOrMap}_1 \ldots \texttt{SymbolOrMap}_n) = \sigma$$

where $\sigma = makeMorphism_{logic(\Gamma.current)}((s_1, t_1), \ldots, (s_n, t_n)))$
and $(s_i, t_i) = sem(\Gamma, \Sigma_1, \Sigma_2, \texttt{SymbolOrMap}_i)$ for $i = 1, \ldots, n$.

Applications **shall** implicitly map those non-logical symbols of the source OMS, for which an explicit mapping is not given, to non-logical symbols of the same (local) name in the target OMS, wherever this is uniquely defined – in detail:

Require: O_s, O_t are OMS
Require: $M \subseteq |\mathsf{Sig}(O_s)| \times |\mathsf{Sig}(O_t)|$ maps non-logical symbols (i.e. elements of the signature) of O_s to non-logical symbols of O_t
  for all $e_s \in |\Sigma(O_s)|$ not covered by M **do**
    $n_s \leftarrow$ localname(e_s)
    $N_t \leftarrow \{$localname$(e) | e \in |\Sigma(O_t)|\}$
    if $N_t = \{e_t\}$ **then** {i.e. if there is a unique target}
      $M \leftarrow M \cup \{(e_s, e_t)\}$
    end if
  end for
Ensure: M completely covers $|\Sigma(O_s)|$

The local name of a non-logical symbol is determined as follows[15]:

Require: e is a non-logical symbol (identified by an IRI; cf. clause 9.7)
  if e has a fragment f **then** {production `ifragment` in IETF/RFC 3987:2005}
    return f

[15] In practice, this can often have the effect of undoing an IRI abbreviation mechanism that was used when writing the respective OMS (cf. clause 9.7). In general, however, functions that turn abbreviations into IRIs are not invertible. For this reason, the implicit mapping of non-logical symbols is specified independently from IRI abbreviation mechanisms possibly employed in the OMS.

else
$n \leftarrow$ the longest suffix of e that matches the `Nmtoken` production of XML W3C/TR REC-xml:2008
return n
end if

$$\boxed{sem(\Gamma, (\Sigma, \Delta), \texttt{Extraction}) = (\Sigma', \Delta')}$$

$sem(\Gamma, (\Sigma, \Delta), \texttt{extraction QualInterfaceSignature}) = (\Sigma', \Delta')$
where $sem(\Gamma, \Sigma, \texttt{QualInterfaceSignature}) = \Sigma''$, $\langle \Sigma', \Delta' \rangle$ is the smallest depleting Σ''-module (see [38] for the definition in a description logic context and [33] for a generalization to an arbitrary institution), i.e. the smallest[16] sub-theory $\langle \Sigma', \Delta' \rangle$ of $(\mathsf{Sig}(\texttt{OMS}), \mathsf{Th}(\texttt{OMS}))$ such that the following model-theoretic inseparability holds

$$\mathsf{Th}(\texttt{OMS}) \setminus \Delta' \equiv_{\Sigma' \cup \Sigma''} \emptyset.$$

This means intuitively that $\mathsf{Th}(\texttt{OMS}) \setminus \Delta'$ cannot be distinguished from $\emptyset$ (what $\Sigma' \cup \Sigma''$ concerns) and formally that

$$\begin{array}{cl} & \{M|_{\Sigma' \cup \Sigma''} \mid M \in \mathsf{Mod}(\mathsf{Sig}(\texttt{OMS})), M \models \mathsf{Th}(\texttt{OMS}) \setminus \Delta'\} \\ = & \{M|_{\Sigma' \cup \Sigma''} \mid M \in \mathsf{Mod}(\mathsf{Sig}(\texttt{OMS}))\}. \end{array}$$

$$\boxed{sem(\Gamma, (\mathcal{I}, \Sigma), \texttt{Approximation}) = (\mathcal{I}, \Sigma')}$$

$sem(\Gamma, \Sigma, \texttt{approx QualInterfaceSignature LogicRef}) = (\mathcal{I}, \Sigma')$
where $\Sigma' = sem(\Gamma, \Sigma, \texttt{QualInterfaceSignature})$ and $sem(\texttt{LogicRef}) = \mathcal{I}$.

$$\boxed{sem(\Gamma, (\Sigma, \Delta), \texttt{Filtering}) = (c, \mathcal{I}, \Sigma', \Delta')}$$

$$sem(\Gamma, (\Sigma, \Delta), \texttt{select BasicOMS}) = (\mathit{select}, \mathcal{I}, \Sigma', \Delta')$$

where $sem(\Gamma, (\Sigma, \Delta), \texttt{BasicOMS}) = (\mathcal{I}, \Sigma', \Delta')$.

$$sem(\Gamma, (\Sigma, \Delta), \texttt{reject BasicOMS}) = (\mathit{reject}, \mathcal{I}, \Sigma', \Delta')$$

where $sem(\Gamma, (\Sigma, \Delta), \texttt{BasicOMS}) = (\mathcal{I}, \Sigma', \Delta')$.

$$\boxed{sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), \texttt{ExtensionOMS}) = (\mathcal{I}, \Sigma', \mathcal{M}', \Delta')}$$

$sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), \texttt{extension-oms ConsStrength ExtensionName ExtendingOMS}) = (\mathcal{I}, \Sigma', \mathcal{M}', \Delta')$
where $(\mathcal{I}, \Sigma', \mathcal{M}', \Delta') = sem(\Gamma, (\Sigma, \mathcal{M}), \texttt{ExtendingOMS})$.

If `ConsStrength` is `model-conservative` or `implied`, the semantics is only defined if each model in $\mathcal{M}$ is the Σ-reduct of some model in $\mathcal{M}'$. In case that `ConsStrength` is `implied`, it is furthermore required that $\Sigma = \Sigma'$. If `ConsStrength` is `consequence-conservative`, the semantics is only defined if for each Σ-sentence φ, $\mathcal{M}' \models \varphi$ implies $\mathcal{M} \models \varphi$. If `ConsStrength` is `definitional`, the semantics is only defined if each model in $\mathcal{M}$ is the Σ-reduct of a unique model in $\mathcal{M}'$.

If `ExtensionName` is present, the inclusion link is labeled with this name.

[16] In [38], it is shown that the smallest depleting Σ''-module exists in description logics, and in [33] this is generalized to arbitrary institutions.

$$\boxed{sem(\Gamma, \Sigma, \texttt{QualInterfaceSignature}) = \Sigma'}$$

$sem(\Gamma, \Sigma,$ `Qual SymbolItems`$) = \Sigma'$, where

$$\Sigma' = \begin{cases} \Sigma \cap sem(\Gamma, \Sigma, \texttt{SymbolItems}) & \text{if} \quad \texttt{Qual = keep-signature} \\ \Sigma \setminus sem(\Gamma, \Sigma, \texttt{SymbolItems}) & \text{if} \quad \texttt{Qual = remove-signature} \end{cases}$$

$$\boxed{sem(\Gamma, \texttt{OMSDefn}) = \Gamma'}$$

An OMS definition extends the global environment:
$sem(\Gamma, \texttt{oms-defn OMSName ConsStrength OMS})$
$= (\Gamma[\texttt{OMSName} \mapsto sem(\Gamma, \texttt{OMS})], L)$

If `ConsStrength` is `model-conservative`, the semantics is only defined if $sem(\Gamma, \texttt{OMS}) \neq \emptyset$. If `ConsStrength` is `consequence-conservative`, the semantics is only defined if $sem(\Gamma, \texttt{OMS})$ has only tautologies[17] as signature-free[18] logical consequences. If `ConsStrength` is `monomorphic`, the semantics is only defined if $sem(\Gamma, \texttt{OMS})$ consist of exactly one isomorphism class of models. If `ConsStrength` is `weak-definitional`, the semantics is only defined if $sem(\Gamma, \texttt{OMS})$ is empty or a singleton. If `ConsStrength` is `definitional`, the semantics is only defined if $sem(\Gamma, \texttt{OMS})$ is a singleton.

$$sem(\Gamma, \texttt{OMSRef}) = \Gamma(\texttt{OMSRef})$$

$$\boxed{sem(\Gamma, \Sigma, \texttt{Symbol}) = s}$$

$$sem(\Gamma, \Sigma, \texttt{Symbol}) = s$$

where s is a logic-specific symbol with the name `Symbol` from $|\Sigma|$. If such symbol does not exist, the semantics is undefined.

$$\boxed{sem(\Gamma, \Sigma_1, \Sigma_2, \texttt{SymbolMap})}$$

$$sem(\Gamma, \Sigma_1, \Sigma_2, \texttt{symbol-map } \texttt{Symbol}_1, \texttt{Symbol}_2) = (s_1, s_2)$$

where $sem(\Gamma, \Sigma_1, \texttt{Symbol}_1) = s_1$ and $sem(\Gamma, \Sigma_2, \texttt{Symbol}_2) = s_2$.

$$\boxed{sem(\Gamma, \Sigma_1, \Sigma_2, \texttt{SymbolOrMap}) = (s, t)}$$

$sem(\Gamma, \Sigma_1, \Sigma_2, \texttt{symbol-map } \texttt{Symbol}_1, \texttt{Symbol}_2) = (s_1, s_2)$
and
$sem(\Gamma, \Sigma_1, \Sigma_2, \texttt{Symbol}) = (s, s)$ where $sem(\Gamma, \Sigma_1, \texttt{Symbol}) = s$.
For the semantics of `Term`, see section 11.2.5.

$$\boxed{sem(\Gamma, \Sigma, \texttt{Sentence}) = \varphi}$$

[17] A tautology is a sentence holding in every model.
[18] A signature-free sentence is one over the empty signature.

$$sem(\Gamma, \Sigma, \texttt{Sentence}) = \varphi$$

where $\varphi \in Sen(\Sigma)$ and the analysis is done in a logic-specific way.

$$\boxed{sem(\texttt{LolaRef}) = L}$$

L is the language or the institution from the heterogeneous logical environment named by `LogicRef`.

$$\boxed{sem(\texttt{LanguageRef}) = L}$$

L is the language from the heterogeneous logical environment named by `LogicRef`.

$$\boxed{sem(\texttt{SyntaxRef}) = S}$$

S is the serialization from the heterogeneous logical environment named by `LogicRef`.

$$\boxed{sem(\texttt{LogicRef}) = L}$$

L is the institution from the heterogeneous logical environment named by `LogicRef`.

$$\boxed{sem(\Gamma, \texttt{OMSLangTrans}) = \rho}$$

$sem(\Gamma, \texttt{named-trans OMSLangTransRef}) = \rho$ where ρ is the institution comorphism from the heterogeneous logical environment named by `OMSLangTransRef`. This is defined only if the domain of ρ is the current logic of Γ.

$sem(L, \texttt{default-trans LolaRef}) = \rho$ where ρ is the unique default institution comorphism from the heterogeneous logical environment running from L to $sem(\texttt{LolaRef})$ (if this is a logic) or to some logic supported by $sem(\texttt{LolaRef})$ (if this is a language). If there is no or no unique such comorphism, the semantics is undefined.

11.2.4. Semantics of OMS Mappings

$$\boxed{sem(\Gamma, \texttt{MappingDefn}) = \Gamma'}$$

See equations for `IntprDefn`, `Entailment`, `EquivDefn`, `ModuleRelDefn` and `AlignDefn`.

$$\boxed{sem(\Gamma, \texttt{IntprDefn}) = \Gamma'}$$

$$\begin{aligned} sem(\Gamma, \; & \texttt{intrp-defn IntprName Conservative IntrpType,} \\ & \texttt{LogicTranslation*, SymbolMapItems}) = \Gamma' \end{aligned}$$

where $\Gamma' = \Gamma[\texttt{IntprName} \to (G, (\rho, \sigma), L_1, L_2)]$
and G is the graph $L_1 \xrightarrow{(\rho,\sigma)} L_2$ where

- $(L_1, L_2) = sem(\Gamma, \texttt{IntrpType})$

- $\rho = (\Phi, \alpha, \beta) : \mathbf{Inst}(L_1) \to \mathbf{Inst}(L_2)$ is the comorphism given by $sem(\Gamma, \texttt{LogicTranslation*})$. If `LogicTranslation*` is missing, the default translations between the logics is selected.
- $sem(\Gamma.\{current = (lang, logic', ser)\}, \Phi(\mathsf{Sig}(L_1)), \mathsf{Sig}(L_2), \texttt{SymbolMapItems}) = \sigma$, where $\Gamma.current = (lang, logic, ser)$ and $logic'$ is the target logic of ρ.

The semantics is only defined if $\beta_{\mathsf{Sig}(L_1)}(M_2|_\sigma) \in \mathsf{Mod}(L_1)$ for each $M_2 \in \mathsf{Mod}(L_2)$. If the optional argument `Conservative` is `model-conservative`, for each model $M_1 \in \mathsf{Mod}(L_1)$ there must exist a model $M_2 \in \mathsf{Mod}(L_2)$ such that $\beta_{\mathsf{Sig}(L_1)}(M_2|_\sigma) = M_1$. If the optional argument `Conservative` is `consequence-conservative`, for each $\mathsf{Sig}(L_1)$-sentence φ, if $\mathcal{M}_2 \models \sigma(\alpha_{\mathsf{Sig}(L_1)}(\varphi))$ then $\mathcal{M}_1 \models \varphi$. If the optional argument `Conservative` is `not-model-conservative`, there must exist a model $M_1 \in \mathsf{Mod}(L_1)$ such that there is no model $M_2 \in \mathsf{Mod}(L_2)$ such that $\beta_{\mathsf{Sig}(L_1)}(M_2|_\sigma) = M_1$. If the optional argument `Conservative` is `not-consequence-conservative`, there is a $\mathsf{Sig}(L_1)$-sentence φ, such that $\mathcal{M}_2 \models \sigma(\alpha_{\mathsf{Sig}(L_1)}(\varphi))$ and $\mathcal{M}_1 \not\models \varphi$.

$$sem(\Gamma, \texttt{refinement IntprName Refinement}) = \Gamma'$$

where $\Gamma' = \Gamma[\texttt{IntprName} \mapsto (G, \sigma, N_1, N_2)]$ and $sem(\Gamma, \texttt{Refinement}) = (G, \sigma, N_1, N_2)$.

$$\boxed{sem(\Gamma, \texttt{IntprType}) = ((N_1, \mathcal{I}_1, \Sigma_1, \mathcal{M}_1, \Delta_1), (N_2, \mathcal{I}_2, \Sigma_2, \mathcal{M}_2, \Delta_2))}$$

$$sem(\Gamma, \texttt{intpr-type OMS}_1\ \texttt{OMS}_2) = (L_1, L_2)$$

where

- $\mathbf{Name}(L_1) = \mathbf{Name}(\texttt{OMS}_1)$ and $\mathbf{Name}(L_2) = \mathbf{Name}(\texttt{OMS}_2)$,
- $(\mathbf{Inst}(L_i), \mathsf{Sig}(L_i), \mathsf{Mod}(L_i), \mathsf{Th}(L_i)) = sem(\Gamma, \texttt{OMS}_i)$, for $i = 1, 2$.

$$\boxed{sem(\Gamma, \texttt{Refinement}) = ((G_1, G_2), \sigma, \mathcal{M})}$$

The signature of a refinement is a pair consisting of the graph of the OMS or network of OMS being refined and the graph of the OMS or network of OMS after refinement. Together with this pair we store the mapping along which the refinement is done. Given two networks G_1 and G_2, a *network morphism* $\sigma : G_1 \to G_2$ is a functor $\sigma^G : Shape(G_1) \to Shape(G_2)$ together with a natural transformation $\sigma^M : G_1 \to \sigma^G; G_2$ such that for each node N_1 in G_1 labeled with $(\mathcal{I}_1, \Sigma_1, \mathcal{M}_1)$ such that $\sigma^G(N_1)$ is a node N_2 labeled with $(\mathcal{I}_2, \Sigma_2, \mathcal{M}_2)$ in G_2, we have a signature morphism $(\rho^M_{N_1}, \sigma^M_{N_1}) : (\mathcal{I}_1, \Sigma_1) \to (\mathcal{I}_2, \Sigma_2)$, where $\rho^M_{N_1} = (\Phi, \alpha, \beta) : \mathcal{I}_1 \to \mathcal{I}_2$ is an institution comorphism between the logics of the two nodes and $\sigma^M_{N_1} : \Phi(\Sigma_1) \to \Sigma_2$ is a signature morphism, such that $\beta_{\Sigma_1}(M_2|_{\sigma^M_{N_1}}) \in \mathcal{M}_1$ for each $M_2 \in \mathcal{M}_2$.

A refinement model is a class $\mathcal{M}$ of pairs of families of models compatible with the two networks. Given a network morphism $\sigma : G_1 \to G_2$ and a G_2 model F, we define $F|_\sigma$ as the family of models $\{M_i\}_{i \in Nodes(G_1)}$ such that $M_i = F_{\sigma^G(i)}|_{\sigma^M_i}$ for each $i \in Nodes(G_1)$.

$$sem(\Gamma, \texttt{ref-oms OMS}) = ((G, G), \sigma, \mathcal{M})$$

where

- G is a graph with just one isolated node N such that $\mathbf{Name}(N) = \mathbf{Name}(\texttt{OMS})$ and the other elements of the tuple labeling L are given by $sem(\Gamma, \texttt{OMS})$,

- σ is the identity morphism on $\mathsf{Sig}(\texttt{OMS})$,
- $\mathcal{M} = \{((M),(M)) \mid M \in \mathsf{Mod}(\texttt{OMS})\}$, where (M) is the singleton family consisting of M.

$$sem(\Gamma, \texttt{ref-network Network}) = ((G,G),\sigma,\mathcal{M})$$

where $sem(\Gamma,\texttt{Network}) = G$, σ is the identity network morphism on G and $\mathcal{M} = \{(F,F) \mid F \in \mathsf{Mod}(G)\}$.

$$sem(\Gamma, \texttt{ref-composition Refinement}_1\ \texttt{Refinement}_2) = ((G_1,G_2'),\sigma,\mathcal{M})$$

where
$sem(\Gamma,\texttt{Refinement}_1) = ((G_1,G_1'),\sigma_1,\mathcal{M}_1)$, $sem(\Gamma,\texttt{Refinement}_2) = ((G_2,G_2'),\sigma_2,\mathcal{M}_2)$ such that $G_1' = G_2$, $\sigma = \sigma_1;\sigma_2$ is a network morphism from G_1 to G_2', and $\mathcal{M} = \{(F_1,F_3) \mid \exists F_2$ such that $(F_1,F_2) \in \mathcal{M}_1$ and $(F_2,F_3) \in \mathcal{M}_2\}$

$$sem(\Gamma, \texttt{simple-oms-ref OMS RefMap Refinement}) = ((G,G_2),\sigma,\mathcal{M})$$

where
$sem^M(\Gamma,\texttt{OMS}) = (\mathcal{I}_1,\Sigma_1,\mathcal{M}_1,\Delta_1)$,
$sem(\Gamma,\texttt{Refinement}) = ((G_1,G_2),(\rho_2,\sigma_2),\mathcal{M}')$ such that G_1 consists of an isolated node labeled with $(\mathcal{I}_2,\Sigma_2,\mathcal{M}_2,\Delta_2)$
$sem(\Gamma,(\mathcal{I}_1,\Sigma_1),(\mathcal{I}_2,\Sigma_2),\texttt{RefMap}) = (\rho_1 = (\Phi,\alpha,\beta) : \mathcal{I}_1 \to \mathcal{I}_2, \sigma_1 : \Phi(\Sigma_1) \to \Sigma_2)$,
for each $(M_1,M_2) \in \mathcal{M}'$ we have that $\beta_{\Sigma_1}(M_1|_{\sigma_1}) \in \mathcal{M}_1$,
G consists of an isolated node labeled with $sem^M(\Gamma,\texttt{OMS})$
$\sigma = (\rho_1,\sigma_1);(\rho_2,\sigma_2)$ and $\mathcal{M} = \{(\beta_{\Sigma_1}(M_1|_{\sigma_1}),M_2) \mid (M_1,M_2) \in \mathcal{M}'\}$.

$$sem(\Gamma, \texttt{simple-network-ref Network RefMap Refinement}) = ((G_1,G_2),\sigma,\mathcal{M})$$

where
$sem^M(\Gamma,\texttt{Network}) = G_1$,
$sem(\Gamma,\texttt{Refinement}) = ((G_1',G_2),\sigma_2,\mathcal{M}')$,
$sem(\Gamma,G_1,G_2,\texttt{RefMap}) = \sigma_1 : G_1 \to G_1'$,
$\sigma = \sigma_1;\sigma_2$ is a network morphism and $\mathcal{M} = \{(F_2|_\sigma,F_2) \mid (F_1,F_2) \in \mathcal{M}'\}$.

$$\boxed{sem(\Gamma,(I_1,\Sigma_1),(I_2,\Sigma_2),\texttt{RefMap}) = (\rho,\sigma)}$$

$$sem(\Gamma,(I_1,\Sigma_1),(I_2,\Sigma_2),\texttt{refmap-oms LogicTranslation SymbolMapItems}) = ((\Phi,\alpha,\beta),\sigma)$$

where
$sem(\Gamma,\texttt{LogicTranslation}) = (\Phi,\alpha,\beta) : \mathcal{I}_1' \to \mathcal{I}_2'$ such that $\mathcal{I}_1' = \mathcal{I}_1$ and $\mathcal{I}_2' = \mathcal{I}_2$
and $sem(\Gamma.current = (lang',logic',ser'),\Phi(\Sigma_1),\Sigma_2,\texttt{SymbolMapItems}) = \sigma : \Phi(\Sigma_1) \to \Sigma_2$ where $\Gamma.current = (lang,logic,ser)$, $logic'$ is the target logic of (Φ,α,β), and $lang'$ and ser' are the default language and serializations for $logic'$.

$$\boxed{sem(\Gamma,G_1,G_2,\texttt{RefMap}) = \sigma : G_1 \to G_2}$$

$$sem(\Gamma, G_1, G_2, \texttt{refmap-networkNodeMap}_1 \ldots \texttt{NodeMap}_n) = \sigma$$

where
$sem(\Gamma, G_1, G_2, \texttt{NodeMap}_1) = (\texttt{OMSName}_1^1, \texttt{OMSName}_2^1, \rho_1, \sigma_1)$, ...
$sem(\Gamma, G_1, G_2, \texttt{NodeMap}_n) = (\texttt{OMSName}_1^n, \texttt{OMSName}_2^n, \rho_n, \sigma_n)$ and
$\sigma^G(\texttt{OMSName}_1^i) = \texttt{OMSName}_2^i$ and $\sigma^M_{\texttt{OMSName}_1^i} = (\rho_i, \sigma_i)$ for each $i = 1, \ldots, n$. The map is required to be total on the nodes of G_1.

$$\boxed{sem(\Gamma, G_1, G_2, \texttt{NodeMap}) = (\texttt{OMSName}_1, \texttt{OMSName}_2, \rho, \sigma)}$$

$sem(\Gamma, G_1, G_2, \texttt{node-map OMSName}_1 \texttt{ OMSName}_2 \texttt{ LogicTranslation* SymbolMapItems}) =$ $(\texttt{OMSName}_1, \texttt{OMSName}_2, \rho, \sigma)$ where $(\mathcal{I}_1, \Sigma_1, \mathcal{M}_1)$ is the label of $\texttt{OMSName}_1$ in G_1, $(\mathcal{I}_2, \Sigma_2, \mathcal{M}_2)$ is the label of $\texttt{OMSName}_2$ in G_2, $sem(\Gamma, \texttt{LogicTranslation*}) = \rho : \mathcal{I}_1 \to \mathcal{I}_2$, $\rho = (\Phi, \alpha, \beta)$, $sem(\Gamma.current = (lang', logic', ser'), \Phi(\Sigma_1), \Sigma_2, \texttt{SymbolMapItems}) = \sigma : \Phi(\Sigma_1) \to \Sigma_2$. where $\Gamma.current = (lang, logic, ser)$, $logic'$ is the target logic of (Φ, α, β) and $lang'$ and ser' are the default language and serialization for $logic'$.

$$\boxed{sem(\Gamma, \texttt{Entailment}) = \Gamma'}$$

$$sem(\Gamma, \texttt{entailment EntailmentName EntailmentType}) = \Gamma'$$

where $\Gamma' = \Gamma[EntailmentName \mapsto sem(\Gamma, \texttt{EntailmentType})]$.

$$\boxed{sem(\Gamma, \texttt{EntailmentType}) = (G, id, L_2, L_1)}$$

$$sem(\Gamma, \texttt{oms-oms-entailment OMS}_1 \texttt{ OMS}_2) = L_2 \xrightarrow{\text{id}} L_1$$

where $\mathbf{Name}(L_1) = \mathbf{Name}(\texttt{OMS}_1)$, $\mathbf{Name}(L_2) = \mathbf{Name}(\texttt{OMS}_2)$, $(\mathbf{Inst}(L_i), \mathsf{Sig}(L_i), \mathsf{Mod}(L_i), \mathsf{Th}(L_i)) = sem(\Gamma, \texttt{OMS}_i)$ for $i = 1, 2$ such that $\mathsf{Sig}(L_1) = \mathsf{Sig}(L_2)$ and $\mathsf{Mod}(L_1) \subseteq \mathsf{Mod}(L_2)$ and id is the identity morphism on $\mathsf{Sig}(L_1)$.

$sem(\Gamma, \texttt{network-oms-entailment Network OMSName OMS}) = G$
where $sem(\Gamma, \texttt{Network}) = G'$ such that G' contains a node n labeled with ($\mathbf{Name}(\texttt{OMSName})$, $sem(\Gamma, \texttt{OMS}) = (\mathcal{I}, \Sigma, \mathcal{M}_2, \Delta_2)$ and $\{\mathcal{M}_n \mid \mathcal{M}$ is compatible with $G'\} \subseteq \mathcal{M}_2$. Then G extends G' with a new node whose label has the name $\mathbf{Name}(\texttt{OMS})$ and the other components given by $sem(\Gamma, \texttt{OMS})$ and with a new theorem link from this new node to the node $\mathbf{Name}(\texttt{OMSName})$, labeled with the identity morphism on Σ.

$$sem(\Gamma, \texttt{network-network-entailment Network}_1 \texttt{ Network}_2) = G$$

where $sem(\Gamma, \texttt{Network}_1) = G_1$, $sem(\Gamma, \texttt{Network}_2) = G_2$, such that $Shape(G_1) = Shape(G_2)$ and for each node $i \in |Shape(G_1)|$ we have that its names in the networks G_1 and G_2 are the same, its signatures are the same and the class of models obtained by projecting each family of models compatible with G_1 to the component i is included in the class of models obtained by projecting each family of models compatible with G_2 to the component i. Then G extends the union of G_1 and G_2 for each pair of nodes (i_1, i_2), where i_1 and i_2 identify the occurences of the same node i in G_1 and G_2 respectively, with a theorem link from i_1 to i_2 labeled with the identity on $\mathsf{Sig}(i_1)$.

$\boxed{sem(\Gamma, \texttt{EquivDefn}) = \Gamma'}$

$$sem(\Gamma, \texttt{equiv-defn EquivName (oms-equiv } O_1\ O_2\ O_3 \texttt{))} = \Gamma'$$

where $\Gamma' = \Gamma[\texttt{EquivName} \mapsto (G, id, N_1, N_2)]$ where G is the graph $N_1 \xrightarrow{\iota_1} N_3 \xleftarrow{\iota_2} N_3$ and N_1 is labelled with $(\mathbf{Name}(O_1), \mathbf{Inst}(O_1), \mathsf{Sig}(O_1), \mathsf{Mod}(O_1), \mathsf{Th}(O_1))$, N_2 with $(\mathbf{Name}(O_2), \mathbf{Inst}(O_2), \mathsf{Sig}(O_2), \mathsf{Mod}(O_2), \mathsf{Th}(O_2))$ and N_3 with $(\mathbf{Name}(O_3), \mathcal{I}, \Sigma, \mathcal{M}, \Delta)$ where $sem^{(\mathsf{Sig}(O_1)\cup\mathsf{Sig}(O_2),\emptyset)}_{\Gamma.lang,\Gamma.logic,\Gamma.ser}(O_3) = (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)$ such that $\iota_i : \mathsf{Sig}(O_i) \to \Sigma$ are signature inclusions and we have that $\mathbf{Inst}(O_1) = \mathbf{Inst}(O_2) = \mathbf{Inst}(O_3)$ and for each $i = 1, 2$ and each model $M_i \in \mathsf{Mod}(O_i)$ there exists a unique model $M \in \mathcal{M}$ such that $M|_{\mathsf{Sig}(O_i)} = M_i$.

$$sem(\Gamma, \texttt{equiv-defn EquivName (network-equiv } N_1\ N_2\ N_3 \texttt{))} = \Gamma'$$

$\boxed{sem(\Gamma, \texttt{ModuleRelDefn}) = \Gamma'}$

$$sem(\Gamma, \texttt{module-defn ModuleName Conservative ModuleType InterfaceSignature}) = \Gamma'$$

where $\Gamma' = \Gamma[\texttt{ModuleName} \mapsto (G, \iota, N_2, N_1)]$ and G is the graph $N_1 \xrightarrow{\iota} N_2$ where N_1 is labeled with $(O_1, \mathbf{Inst}(O_1), \mathsf{Sig}(O_1), \mathsf{Mod}(O_1), \mathsf{Th}(O_1)$ N_2 with $(O_2, \mathbf{Inst}(O_2), \mathsf{Sig}(O_2), \mathsf{Mod}(O_2), \mathsf{Th}(O_2))$, and ι is an inclusion, when $\Sigma \subseteq \mathsf{Sig}(O_2) \subseteq \mathsf{Sig}(O_1)$ and if $c =$%mcons and for each $M \in \mathsf{Mod}(O_2)$ there is a model $M' \in \mathsf{Mod}(O_1)$ such that $M'|_\Sigma = M|_\Sigma$, or if $c =$%ccons and for each $\varphi \in \mathsf{Sen}(\Sigma)$, $O_1 \models \varphi$ implies $O_2 \models \varphi$.

$\boxed{sem(\Gamma, \texttt{AlignDefn}) = \Gamma'}$

$sem(\Gamma, \texttt{align-defn AlignName AlignCard AlignType AlignSem Corresps}) = \Gamma'$
where $sem(\Gamma, \texttt{AlignType}) = (L_1, L_2)$ and
$\Gamma' = \Gamma[\texttt{AlignType} \mapsto (sem(\Gamma, (L_1, L_2), \texttt{AlignCard AlignSem Corresps}), id, L_1, L_2)]$

$\boxed{sem(\Gamma, \texttt{AlignType}) = (L_1, L_2)}$

$$sem(\Gamma, \texttt{align-type OMS}_1\ \texttt{OMS}_2) = (L_1, L_2)$$

where L_1 is a node label whose name is $\mathbf{Name}(\texttt{OMS}_1)$ and whose other components are given by $sem(\Gamma, \texttt{OMS}_1)$ and similarly, L_2 is a node label whose name is $\mathbf{Name}(\texttt{OMS}_2)$ and whose other components are given by $sem(\Gamma, \texttt{OMS}_2)$.

$\boxed{sem(\Gamma, L_1, L_2, \texttt{AlignCard AlignSem Corresps}) = G}$

$$sem(\Gamma, L_1, L_2, \texttt{AlignCard AlignSem}, C_1, \ldots, C_n) = G$$

where
$L_1' = sem(\Gamma, L_1, \texttt{AlignSem})$,
$L_2' = sem(\Gamma, L_2, \texttt{AlignSem})$,
$G = sem(\Gamma, L_1', L_2', \texttt{AlignCard AlignSem}, C_1, \ldots, C_n)$.

$$\boxed{sem(\Gamma, L_1, \texttt{AlignSem}) = L_1'}$$

$$sem(\Gamma, L_1, \texttt{AlignSem}) = \begin{cases} L_1 & \text{if } \texttt{AlignSem} = \texttt{global-domain} \\ relativize_{logic(\Gamma.current)}(L_1) & \text{otherwise} \end{cases}$$

where the relativisation procedure is logic-specific.

$$\boxed{sem(\Gamma, L_1, L_2, \texttt{AlignCard AlignSem}\ C_1 \ldots C_n) = G}$$

$sem(\Gamma, L_1, L_2, \texttt{AlignCard AlignSem}, C_1, \ldots, C_n) = G$
where
if at least one of the correspondences $C_1, \ldots, C_n$ has a confidence value different than 1, then the semantics of the alignment is not defined, and the alignment is ill-formed if the alignment mapping does not have the arities given by `AlignCard`, otherwise G is a W-shaped graph as below

$$\begin{tikzcd}
L_1 & & L_B & & L_2 \\
& L_s \arrow[ul, "\iota_1"] \arrow[ur, "\sigma_1"'] & & L_t \arrow[ul, "\sigma_2"'] \arrow[ur, "\iota_2"] &
\end{tikzcd}$$

where L_B, L_s and L_t are built in a logic-specific way from the correspondences $C_1, \ldots, C_n$ taking into account `AlignSem`. [?] illustrates how this construction works in the case of OWL, in a way that can be generalized to other logics.

11.2.5. Semantics of Queries

While queries are very important from a practical point of view, their semantics so far has been developed only for individual institutions. In [53], three options for an institution-independent semantics of queries and derived signature morphisms (which can map symbols to terms) are discussed. Currently, it is not clear which one would be the best choice. It is expected that after some experience with DOL, a choice will crystallize. This means that in the current version, the semantics of queries is elided, and left for a later version of DOL.

Annex

A. Annex (normative): LoLa, an RDF vocabulary that implements the DOL terminology

This annex specifies LoLa, an RDF vocabulary that implements the terms and definitions from clause 4. Applications of LoLa include modeling statements about OMS in RDF, e.g., when annotating OMS, or when describing new conforming logics, OMS languages, serializations, translations, etc., in the registry stipulated by chapter 2. LoLa is currently maintained as an OWL ontology and, prospectively, as an OMS library implemented in DOL, at `https://ontohub.org/meta/lola/ontology`.[1]. For a full treatment of the background and design considerations of LoLa please see [40].

The tables in this annex list the classes and object properties of LoLa and thus the essential parts of its implementation. All classes and object properties are assumed to be in the LoLa namespace unless stated otherwise.

Table A.1 lists the classes of LoLa. Each row of the table translates into the following OWL declarations (given in OWL Manchester syntax [26]). The definitions of the classes can be found in clause 4, sometimes under different names if stated so in the table. Classes rendered in italics are abstract superclasses that have no direct correspondence in the terminology.

```
Class: ...
  SubClassOf: ...
```

Table A.2 lists the object properties of LoLa. Each row of the table translates into the following OWL declarations (given in OWL Manchester syntax). The definitions of the properties can be found in clause 4, sometimes under different names if stated so in the table.

```
ObjectProperty: ...
  Domain: ...
  Range: ...
  SubPropertyOf: ...
```

[1]The preferred location for LoLa snapshots being part of this OMG standard is `http://www.omg.org/spec/DOL/Current/lola`.

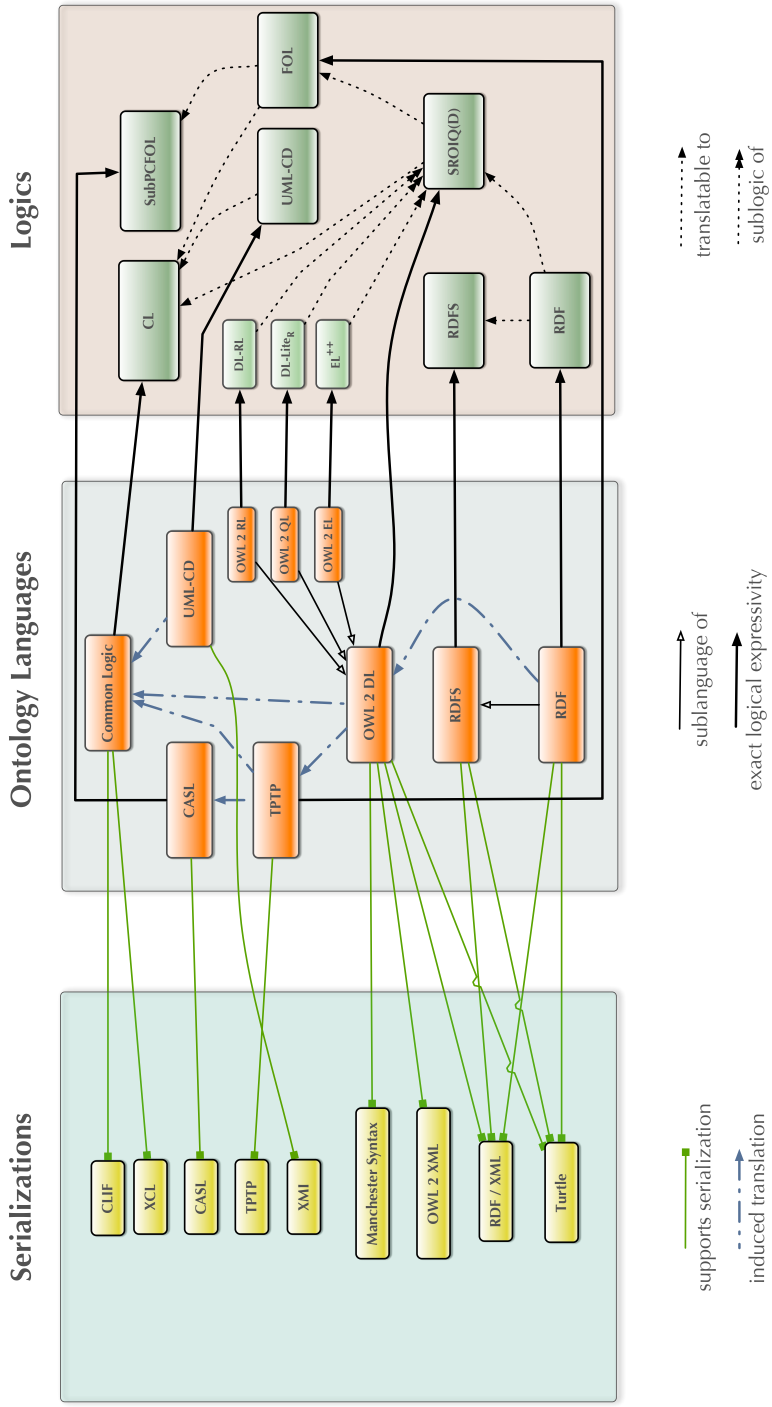


Figure A.1.: Subset of the OntoIOp registry, shown as an RDF graph

Table A.1.: LoLa Classes

Class	Superclass	Section
Language		
OMSLanguage	*Language*	4.2
Logic		4.9
Serialization		4.8
Mapping		4.6 (OMS mapping)
LanguageMapping	Mapping	
LogicMapping	Mapping	
Translation	Mapping	
Reduction	Mapping	
DefaultMapping	Mapping	
WeaklyExactMapping	Mapping	
ExactMapping	WeaklyExactMapping	
FaithfulMapping	Mapping	
ModelExpansiveMapping	FaithfulMapping	
ModelBijectiveMapping	ModelExpansiveMapping	
Embedding	ModelBijectiveMapping, LogicMapping, Translation	
PlainMapping	Mapping	
SimpleTheoroidalMapping	Mapping	

Table A.2.: LoLa Properties

Property *documentation*	domain	range
subLogicOf *The subject is a sublogic of the object*	Logic	Logic
supportsLogic *The subject OMS language has a semantics specified in terms of the object logic.*	Language	Logic
specifiesSemanticsOf *The subject logic is used to specify the semantics of the object OMS language; inverse of supportsLogic.*	Logic	Language
supportsSerialization *OMS in the subject OMS language can be serialized in the object serialization. Note that the serialization should be as specific as possible, i.e. one should not say that "OWL can be serialized in XML" and "Common Logic can be serialized in XML", but instead "OWL can be serialized in OWL/XML" and "Common Logic can be serialized in XCL", taking into account that OWL/XML and XCL are two different XML languages.*	Language	Serialization
serializes *The subject logic is used to specify the semantics of the object OMS language; inverse of supportsSerialization.*	Serialization	Language

B. Annex (informative): Conformance of OWL 2 DL With DOL

The semantic conformance of OWL 2 DL (as specified in W3C/TR REC-owl2-syntax:2009) with DOL is established in [50].

The OWL/XML serialization satisfies the criteria for XML conformance. The mapping of OWL 2 DL to RDF graphs satisfies the criteria for RDF conformance . The OWL 2 Manchester syntax satisfies the criteria for text conformance.

OWL can be formalized as an institution as follows:

Definition 13 ***OWL 2 DL.*** OWL *2 DL is the description logic (DL) based fragment of the web ontology language* OWL*. We start with the simple description logic $\mathcal{ALC}$, and then proceed to the more complex description logic $\mathcal{SROIQ}$ which is underlying* OWL *2 DL. Signatures of the description logic $\mathcal{ALC}$ consist of a set $\mathcal{A}$ of atomic concepts, a set $\mathcal{R}$ of roles and a set $\mathcal{I}$ of individual constants. Signature morphisms are tuples of functions, one for each signature component. Models are first-order structures $I = (\Delta^I, .^I)$ with universe Δ^I that interpret concepts as unary and roles as binary predicates (using $.^I$). $I_1 \leq I_2$ if $\Delta^{I_1} = \Delta^{I_2}$ and all concepts and roles of I_1 are subconcepts and subroles of those in I_2. Sentences are subsumption relations $C_1 \sqsubseteq C_2$ between concepts, where concepts follow the grammar*

$$C ::= \mathcal{A} \mid \top \mid \bot \mid C_1 \sqcup C_2 \mid C_1 \sqcap C_2 \mid \neg C \mid \forall R.C \mid \exists R.C$$

These kind of sentences are also called TBox sentences. Sentences can also be ABox sentences, which are membership assertions of individuals in concepts (written $a : C$ for $a \in \mathcal{I}$) or pairs of individuals in roles (written $R(a, b)$ for $a, b \in \mathcal{I}, R \in \mathcal{R}$). Satisfaction is the standard satisfaction of description logics.

The logic $\mathcal{SROIQ}$ [31], which is the logical core of the Web Ontology Language OWL *2 DL*[1]*, extends $\mathcal{ALC}$ with the following constructs: (i) complex role inclusions such as $R \circ S \sqsubseteq S$ as well as simple role hierarchies such as $R \sqsubseteq S$, assertions for symmetric, transitive, reflexive, asymmetric and disjoint roles (called RBox sentences, denoted by $\mathcal{SR}$), as well as the construct $\exists R.\mathsf{Self}$ (collecting the set of 'R-reflexive points'); (ii) nominals, i.e. concepts of the form $\{a\}$, where $a \in \mathcal{I}$ (denoted by $\mathcal{O}$); (iii) inverse roles (denoted by $\mathcal{I}$); qualified and unqualified number restrictions ($\mathcal{Q}$). For details on the rather complex grammatical restrictions for $\mathcal{SROIQ}$ (e.g. regular role inclusions, simple roles) compare [31].*

OWL profiles *are syntactic restrictions of* OWL *2 DL that support specific modeling and reasoning tasks, and which are accordingly based on DLs with appropriate computational properties. Specifically,* OWL *2* EL *is designed for ontologies containing large numbers of concepts or relations,* OWL *2* QL *to support query answering over large amounts of data, and* OWL *2* RL *to support scalable reasoning using rule languages (*EL, QL, *and* RL *for short) .*

We sketch the logic $\mathcal{EL}$ which is underlying the EL *profile.*[2] *$\mathcal{EL}$ is a syntactic restriction of*

[1] See also `http://www.w3.org/TR/owl2-overview/`

[2] To be exact, EL adds various 'harmless' expressive means and syntactic sugar to $\mathcal{EL}$ resulting in the DL $\mathcal{EL}$ ++.

$\mathcal{ALC}$ to existential restriction, concept intersection, and the top concept:

$$C ::= \mathcal{A} \mid \top \mid C_1 \sqcap C_2 \mid \exists R.C$$

Note that $\mathcal{EL}$ does not have disjunction or negation, and is therefore a sub-Boolean logic. □

Remark: strictly speaking, the institution defined above is *OWL 2 DL without restrictions* in the sense of [58]. The reason is that in an institution, the sentences can be used for arbitary formation of theories. This is related to the presence of DOL's union operator on OMS. OWL 2 DL's specific restrictions on theory formation can be modeled *inside* this institution, as a constraint on OMS. This constraint is generally not preserved under unions or extensions. DOL's multi-logic capability allows the clean distinction between ordinary OWL 2 DL and OWL 2 DL without restrictions.

C. Annex (informative): Conformance of Common Logic with DOL

The semantic conformance of Common Logic (as specified in ISO/IEC 24707:2007) with DOL is established in [50].

The XCF dialect of Common Logic has a serialization that satisfies the criteria for XML conformance. The CLIF dialect of Common Logic has a serialization that satisfies the criteria for text conformance.

Common Logic can be defined as an institution as follows:

Definition 14 ***Common Logic.*** *A common logic signature Σ (called vocabulary in Common Logic terminology) consists of a set of names, with a subset called the set of discourse names, and a set of sequence markers. An signature morphism maps names and sequence markers separately, subject to the requirement that a name is a discourse name in the smaller signature if and only if it is one in the larger signature. A Σ-model $I = (UR, UD, rel, fun, int, seq)$ consists of a set UR, the universe of reference, with a non-empty subset $UD \subseteq UR$, the universe of discourse, and four mappings:*

- *rel from UR to subsets of $UD^* = \{< x_1, \ldots, x_n > | x_1, \ldots, x_n \in UD\}$ (i.e., the set of finite sequences of elements of UD);*
- *fun from UR to total functions from UD^* into UD;*
- *int from names in Σ to UR, such that $int(v)$ is in UD if and only if v is a discourse name;*
- *seq from sequence markers in Σ to UD^*.*

A Σ-sentence is a first-order sentence, where predications and function applications are written in a higher-order like syntax: $t(s)$. Here, t is an arbitrary term, and s is a sequence term, which can be a sequence of terms $t_1 \ldots t_n$, or a sequence marker. A predication $t(s)$ is interpreted by evaluating the term t, mapping it to a relation using rel, and then asking whether the sequence given by the interpretation s is in this relation. Similarly, a function application $t(s)$ is interpreted using fun. Otherwise, interpretation of terms and formulae is as in first-order logic. A further difference to first-order logic is the presence of sequence terms (namely sequence markers and juxtapositions of terms), which denote sequences in UD^*, with term juxtaposition interpreted by sequence concatenation. Note that sequences are essentially a non-first-order feature that can be expressed in second-order logic.

Model reducts are defined in the following way: Given a signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ and a Σ_2-model $I_2 = (UR, UD, rel, fun, int, seq)$, $I|_\sigma = (UR, UD, rel, fun, int \circ \sigma, seq \circ \sigma)$.

Given two CL *models $I_1 = (UR_1, UD_1, rel_1, fun_1, int_1, seq_1)$ and $I_2 = (UR_2, UD_2, rel_2, fun_2, int_2, seq_2)$, a homomorphism $h : I_1 \to I_2$ is a function $h : UR_1 \to UR_2$ such that*

- *h restricts to $k : UD_1 \to UD_2$,*
- *for each $x \in UR_1$ and $s \in UD_1^*$, if $s \in rel_1(x)$, then $k^*(s) \in rel_2(h(x))$[1],*

[1] k^* is the extension of h to sequences.

- *for each* $x \in UR_1$, $k \circ fun_1(x) = fun_2(h(x)) \circ k^*$,
- *for each name* n *in* Σ, $int_2(n) = h(int_1(n))$,
- *for each sequence marker* n *in* Σ, $seq_2(n) = k^*(seq_1(n))$.

We call the restriction of CL *to sentence without sequence markers* CL $^-$. $\square$

Note that Common Logic also includes sentence formation constructs like `cl:imports` that in DOL terms belong to the structuring language. They have been omitted from the institution, because they must not occur in basic OMS. They can occur in structured native OMS, however, and need to be flattened out in order to obtain a theory in the CL institution.

D. Annex (informative): Conformance of RDF and RDF Schema with DOL

The semantic conformance of RDF Schema (as specified in W3C/TR REC-rdf-schema:2014) with DOL is established in [50].

The way of representing RDF Schema ontologies as RDF graphs satisfies the criteria for RDF conformance.

Definition 15 (RDF **and RDF Schema**) *Following [43], we define the institutions for the Resource Description Framework (*RDF*) and* RDF *Schema (also known as* RDFS*), respectively. These are based on a logic called* bare RDF *(*SimpleRDF*), which consists of triples only (without any predefined resources).*

A signature $\mathbf{R_s}$ *in* SimpleRDF *is a set of resource references. For* $sub, pred, obj \in \mathbf{R_s}$*, a triple of the form* $(sub, pred, obj)$ *is a sentence in* SimpleRDF*, where sub, pred, obj represent subject name, predicate name, object name, respectively. An* $\mathbf{R_s}$*-model* $M = \langle R_m, P_m, S_m, EXT_m \rangle$ *consists of a set* R_m *of resources, a set* $P_m \subseteq R_m$ *of predicates, a mapping function* $S_m : \mathbf{R_s} \rightarrow R_m$*, and an extension function* $EXT_m : P_m \rightarrow \mathcal{P}(R_m \times R_m)$ *mapping every predicate to a set of pairs of resources. Satisfaction is defined as follows:*

$$\mathfrak{M} \models_{\mathbf{R_s}} (sub, pred, obj) \Leftrightarrow (S_m(sub), (S_m(obj)) \in EXT_m(S_m(pred)).$$

Both RDF *and* RDFS *are built on top of* SimpleRDF *by fixing a certain standard vocabulary both as part of each signature and in the models. Actually, the standard vocabulary is given by a certain theory. In case of* RDF*, it contains e.g. resources* `rdf:type` *and* `rdf:Property` *and* `rdf:subject`*, and sentences like, e.g.* (`rdf:type`, `rdf:type`, `rdf:Property`)*, and* (`rdf:subject`, `rdf:type`, `rdf:Property`)*.*

In the models, the standard vocabulary is interpreted with a fixed model. Moreover, for each RDF*-model* $M = \langle R_m, P_m, S_m, EXT_m \rangle$*, if* $p \in P_m$*, then it must hold* $(p, S_m(\texttt{rdf:Property})) \in EXT_m(\texttt{rdf:type})$*. For* RDFS*, similar conditions are formulated (here, for example also the subclass relation is fixed).*

In the case of RDFS*, the standard vocabulary contains more elements, like* `rdfs:domain`, `rdfs:range`, `rdfs:Resource`, `rdfs:Literal`, `rdfs:Datatype`, `rdfs:Class`, `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:member`, `rdfs:Container`, `rdfs:ContainerMembershipProperty`*.*

There is also OWL *Full, an extension of* RDFS *with resources such as* `owl:Thing` *and* `owl:oneOf`*, tailored towards the representation of* OWL *[27].*

□

E. Annex (informative): Conformance of UML class and object diagrams with DOL

This informative annex demonstrates conformance of UML class and object diagrams with DOL by defining an institution for both. We concentrate on the static aspects of class diagrams; that is, change of state is ignored. This means that all operations are query operations.

The institution of UML class and object diagrams is defined using a translation of UML class diagrams to Common Logic, following the fUML specification and [59].

E.1. Preliminaries

From the fUML specification, section 10.3.1, we inherit the axioms for primitive types: Booleans, numbers, sequences and strings. These axiomatize (among others) predicates corresponding to primitive types, e.g. `buml:Boolean`, `form:Number`, `form:NaturalNumber`, `buml:Integer`, `form:Sequence`, `form:Character`, and `buml:String`.

We additionally need to axiomatize a number of predicates in Common Logic (note that enumerations are not axiomatized in fUML):

```
logic CLIF

oms pairs =
  (forall (x y) (= (form:first (form:pair x y)) x))
  (forall (x y) (= (form:second (form:pair x y)) y))
  (forall (x y) (form:Pair (form:pair x y)))
  (forall (p) (if (form:Pair p)
                  (= (form:pair (form:first p) (form:second p)) p)))
end

oms sequences =
fuml:sequences.clif and pairs
then
  // fuml:sequence - membership of an element in a sequence
  (forall (x s)
      (if (form:sequence-member x s)
          (form:Sequence s)))

  (forall (x s)
      (iff (form:sequence-member x s)
           (exists (pt)
```

```
               (and (form:in-sequence s pt)
                    (form:in-position pt x)) )))

  // selection of elements
  (forall (o) (= (form:select1 o form:empty-sequence) form:empty-sequence))
  (forall (o y s)
          (= (form:select1 o (form:sequence-insert (form:pair o y) s))
             (form:sequence-insert y (form:select1 o s))))
  (forall (o x y s)
          (if (not (= x o))
              (= (form:select1 o (form:sequence-insert (form:pair x y) s))
                 (form:select1 o s))))
  (forall (o) (= (form:select2 o form:empty-sequence) form:empty-sequence))
  (forall (o x s)
          (= (form:select2 o (form:sequence-insert (form:pair x o) s))
             (form:sequence-insert x (form:select2 o s))))
  (forall (o x y s)
          (if (not (= y o))
              (= (form:select2 o (form:sequence-insert (form:pair x y) s))
                 (form:select2 o s))))

  (forall (i s)
          (= (form:n-select form:empty-sequence i s)
             form:empty-sequence))
  (forall (a i s t x)
          (if (= (insert-i i x t) s)
              (= (form:n-select (form:sequence-insert s a) i t)
                 (form:sequence-insert s (form:n-select a i t)))))
  (forall (a i s t)
          (if (not (exists (x) (= (insert-i i x t) s)))
              (= (form:n-select (form:sequence-insert s a) i t)
                 (form:n-select a i t))))

  // insert element at i-th position
  (forall (x s)
          (= (insert-i form:0 x s) (form:sequence-insert x s)))
  (forall (i j x y s)
          (if (form:add-one i j)
              (= (insert-i j x (form:sequence-insert y s))
                 (form:sequence-insert y (insert-i i x s)))))
end

oms sequences-insert =
sequences then
  // insertion of elements
  (forall (x s1 s2)
    // inserting an element means...
    (if (= (form:sequence-insert x s1) s2)
        (and (form:Sequence s1)
```

```
             (form:Sequence s2)
             // the new element is at the first position
             (form:in-position-count s2 form:1 x)
             // and all other elements are shifted by one
             (forall (n1 n2 y)
               (if (form:add-one n1 n2)
                   (iff (form:in-position-count s1 n1 y)
                        (form:in-position-count s2 n2 y)))))))
  // synonym
 (forall (s) (= (form:sequence-length s) (form:sequence-size s)))
end

oms ordered-sets =
sequences with
  form:Sequence |-> form:Ordered-Set,
  form:empty-sequence |-> form:empty-ordered-set,
  form:sequence-length |-> form:ordered-set-size,
  form:same-sequence |-> form:same-ordered-set,
  form:sequence-member |-> form:ordered-set-member,
  form:in-sequence |-> form:in-ordered-set,
  form:before-in-sequence |-> form:before-in-ordered-set,
  form:position-count |-> form:ordered-set-position-count
  form:in-position-count |-> form:in-ordered-set-position-count
then
//Different positions contain different elements
  (forall (s x1 x2 n1 n2)
            (if (and (form:in-ordered-set-position-count s n1 x1)
                     (form:in-ordered-set-position-count s n2 x2)
                     (= x1 x2))
                (= n1 n2)))
  // insertion of elements
  (forall (x s1 s2)
    (if (= (form:ordered-set-insert x s1) s2)
        (and (form:Ordererd-Set s1)
             (form:Ordererd-Set s2)
  // no element can be inserted twice
  (forall (x s)
    (if (from:ordered-set-member x s)
        (= (form:ordered-set-insert x s) s)))
  // inserting a new element
  (forall (x s)
    (if (not (from:ordered-set-member x s1))
        (exists (s2)
          (and (= (form:ordered-set-insert x s1) s2)
               // the new element is at the first position
               (form:in-ordered-set-position-count s2 form:1 x)
               // and all other elements are shifted by one
               (forall (n1 n2 y)
                 (if (form:add-one n1 n2)
```

```
                    (iff (form:in-ordered-set-position-count s1 n1 y)
                         (form:in-ordered-set-position-count s2 n2 y)))))))
end

oms sets =
//An empty set has no members.
(forall (s)
        (if (form:empty-set s)
            (form:Set s)))
(forall (s)
        (if (form:Set s)
            (iff (form:empty-set s)
                 (not (exists (x)
                              (form:set-member x s))))))
//Size of sets
(forall (s n)
        (if (form:set-size s n)
            (and (form:Set s)
                 (buml:UnlimitedNatural n))))
(= (form:set-size form:empty-set) form:0)
(forall (x s)
        (if (not (form:set-member x s))
            (exists (n)
              (and (form:add-one (form:set-size s) n)
                   (= (form:set-size (form:set-insert x s))
                      n)))))

//The same-set relation is true for sets that have the same members.
// but: why not replace same-set with = ?
(forall (s1 s2)
        (if (form:same-set s1 s2)
            (and (form:Set s1)
                 (form:Set s2))))
(forall (s1 s2)
        (iff (form:same-set s1 s2)
             (forall (x)
                     (iff (form:set-member x s1)
                          (form:set-member x s2)))))
//Insertion of elements into sets and set membership
(forall (x s)
        (if (form:Set s)
            (form:Set (form:set-insert x s))))
(forall (x y s)
        (iff (form:set-member x (form:set-insert y s))
             (or (= x y)
                 (form:set-member x s))))
end

oms bags =
```

```
//An empty bag has no members.
(forall (s)
        (if (form:empty-bag s)
            (form:Bag s)))
(forall (s)
        (if (form:Bag s)
            (iff (form:empty-bag s)
                 (not (exists (x)
                              (form:bag-member x s))))))
//Size of bags
(forall (s n)
        (if (form:bag-size s n)
            (and (form:Bag s)
                 (buml:UnlimitedNatural n))))
(= (form:bag-size form:empty-bag) form:0)
(forall (x s)
        (exists (n)
            (and (form:add-one (form:bag-size s) n)
                 (= (form:bag-size (form:bag-insert x s))
                    n))))

//The same-bag relation is true for bags that have the same members.
(forall (s1 s2)
        (if (form:same-bag s1 s2)
            (and (form:Bag s1)
                 (form:Bag s2))))
(forall (s1 s2)
        (iff (form:same-bag s1 s2)
             (forall (x)
                     (iff (form:bag-member-count x s1)
                          (form:bag-member-count x s2)))))
//Insertion of elements into bags and bag membership
(forall (x s)
        (if (form:Bag s)
            (form:Bag (form:bag-insert x s))))
(forall (x y s)
        (iff (form:bag-member x (form:bag-insert y s))
             (or (= x y)
                 (form:bag-member x s))))
//Member count
(forall (x s)
        (if (form:Bag s)
            (buml:UnlimitedNatural (form:bag-member-count x s))))
(= (form:bag-member-count form:empty-bag) form:0)
(forall (x s)
        (exists (n)
           (and (form:add-one (form:bag-member-count x s) n)
                (= (form:bag-member-count x (form:bag-insert x s))
                   n))))
```

```
(forall (x y s)
        (if (not (= x y))
            (= (form:bag-member-count x (form:bag-insert y s))
               (form:bag-member-count x s))))
end

oms collection-types =
  sequences-insert and ordered-sets and sets and bags
then
//bag to set
(forall (b)
        (if (form:Bag s)
            (form:Set (form:bag2set b))))
(= (form:bag2set form:empty-bag) form:empty-set)
(forall (x b)
        (if (form:Bag b)
            (= (form:bag2set (form:set-insert x b))
               (form:bag-insert x (form:bag2set b)))))

//sequence to ordered set
(forall (s)
        (if (form:Sequence s)
            (form:Ordered-Set (form:seq2ordset s))))
(= (form:seq2ordset form:empty-sequence) form:empty-ordered-set)
(forall (x s)
        (if (form:Sequence s)
            (= (form:seq2ordset (form:sequence-insert x s))
               (form:ordered-set-insert x (form:seq2ordset s)))))

//sequence to bag
(forall (s)
        (if (form:Sequence s)
            (form:Bag (form:seq2bag s))))
(= (form:seq2bag form:empty-sequence) form:empty-bag)
(forall (x s)
        (if (form:Sequence s)
            (= (form:seq2bag (form:sequence-insert x s))
               (form:bag-insert x (form:seq2bag s)))))

//ordered-set to set
(forall (b)
        (if (form:Ordered-Set s)
            (form:Set (form:ordset2set b))))
(= (form:ordset2set form:empty-ordered-set) form:empty-set)
(forall (x b)
        (if (form:Ordered-Set b)
            (= (form:ordset2set (form:set-insert x b))
               (form:ordered-set-insert x (form:ordset2set b)))))
```

```
//sequence to set
(forall (s)
        (if (form:Sequence s)
            (form:Set (form:seq2set s))))
(forall (s) (= (form:seq2set s) (form:ordset2set (form:seq2ordset s))))

// leq
(forall (x y)
   (iff (buml:leq x y)
        (or (= x y)
            (buml:less-than x y))))
end


oms uml-cd-preliminaries =
  collection-types and pairs
end
```

Using this infrastructure, we obtain an institution for UML class diagrams as described in the following sections.

E.2. Signatures

Class/data type hierarchies. A *class/data type hierarchy* $(C, \leq_C)$ is given by a partial order where the set C contains the *class/data type names*, which are closed w.r.t. the *built-in data types* Boolean, UnlimitedNatural, Integer, Real, and String, i.e., $\{\mathsf{Boolean}, \mathsf{UnlimitedNatural}, \mathsf{Integer}, \mathsf{Real}, \mathsf{String}\} \subseteq C$; and the partial ordering relation $\leq_C$ represents a *generalisation relation* on C, where we say that c_1 is a *sub-class/data type* of c_2 if $c_1 \leq_C c_2$.

A *class/data type hierarchy map* $\gamma : (C, \leq_C) \to (D, \leq_D)$ is given by a monotone map from $(C, \leq_C)$ to $(D, \leq_D)$, i.e., $\gamma(c) \leq_D \gamma(c')$ if $c \leq_C c'$, such that $\gamma(c) = c$ for all $c \in \{\mathsf{Boolean}, \mathsf{UnlimitedNatural}, \mathsf{Integer}, \mathsf{Real}, \mathsf{String}\}$.

We use the *collection type constructors* OrderedSet, Set, Sequence, and Bag for representing the meta-attributes “ordered” and “unique” of MultiplicityElement according to the following table:[1]

	ordered	not ordered
unique	OrderedSet	Set
not unique	Sequence	Bag

The default is “not ordered” and “unique”.[2]

For a class/data type $c \in C$ of a class/data type-hierarchy $(C, \leq_C)$ and a collection type constructor $\tau \in \{\mathsf{OrderedSet}, \mathsf{Set}, \mathsf{Sequence}, \mathsf{Bag}\}$, we write $\tau[c]$ for the induced *collection type*.

Let $(C, \leq_C)$ be a class/data type hierarchy.

- An *attribute declaration*[3] over $(C, \leq_C)$ is of the form $c.p : \tau[c']$ with $c, c' \in C$, τ a collection type constructor, and p an *attribute name*.

[1] Cf. UML Superstructure Specification 2.4.1, p. 128; UML 2.5, p. 27.
[2] UML Superstructure Specification 2.4.1, p. 96; there does not seem to be default in UML 2.5.
[3] We separate attributes from association member ends due to their different uses. In UML, both are of class Property.

- A *query operation declaration* over $(C, \leq_C)$ is of the form $c.q(x_1 : \tau_1[c_1], \dots, x_r : \tau_r[c_r]) : \tau[c']$ with $c, c_1, \dots, c_r, c' \in C$, τ a collection type constructor, o an *operation name*, and $x_1, \dots, x_r$ *parameter names*.
- An *association declaration* over $(C, \leq_C)$ is of the form $a(p_1 : \tau_1[c_1], \dots, p_r : \tau_r[c_r])$ with $r \geq 2$, $c_1, \dots, c_r \in C$, $\tau_1, \dots, \tau_r$ classifier annotations, a an *association name*, and $p_1, \dots, p_r$ *member end names*.[4] An association declaration $\mathbf{a} = a(p_1 : \tau_1[c_1], \dots, p_r : \tau_r[c_r])$ *yields* the *property declarations* $\mathbf{a}.p_i : \tau_i[c_i]$ for $1 \leq i \leq r$. An association declaration is *binary* if $r = 2$.[5]
- A *composition declaration* over $(C, \leq_C)$ is of the form $m(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2])$ with $c_1, c_2 \in C$, τ_2 a collection type constructor, m a *composition name*, and p_1, p_2 *member end names*.[6] A composition declaration $\mathbf{m} = m(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2])$ *yields* the property declarations $\mathbf{m}.p_1 : \mathsf{Set}[c_1]$ and $\mathbf{m}.p_2 : \tau_2[c_2]$.

Class/data type nets (Signatures). A *class/data type net* $\Sigma = ((C, \leq_C), P, O, A, M)$ comprises a class/data type hierarchy $(C, \leq_C)$ and a set P of attribute declarations, a set O of operation declarations, a set A of association declarations over $(C, \leq_C)$, and a set M of composition declarations over $(C, \leq_C)$, such that the following properties are satisfied:

- attribute names are unique along the generalisation relation: if $c_1.p_1 : \tau_1[c'_1]$ and $c_2.p_2 : \tau_2[c'_2]$ are different property declararations in P and $c_1 \leq_C c_2$, then $p_1 \neq p_2$;
- association and composition names are unique: if d_1 and d_2 are the names of two different association or composition declarations in $M \cup A$, then $d_1 \neq d_2$;
- member end names are unique: if $p_1, \dots, p_r$ are the member end names of an association declaration in A or a composition declaration in M, then $p_i \neq p_j$ for $1 \leq i \neq j \leq r$;[7]
- the type of a member end[8] owned by a class/data type coincides with its declarations as attribute: We say that a property declaration $\mathbf{a}.p_i : \tau_i[c_i]$ yielded by a binary association $\mathbf{a} = a(p_1 : \tau_1[c_1], p_2 : \tau_2[c_2])$ is *owned by* $c_0 \in C$ if $c_{3-i} \leq_C c_0$ and there is an attribute declaration $c_0.p_i : \tau_i[c_i] \in P$; and similarly for property declarations yielded by composition

[4]The member ends are ordered according to the UML Superstructure Specification 2.4.1, p. 29; UML 2.5, p. 206; hence we use a tuple-like notation.

[5]Only binary association may show member ends that are properties not owned by the association (UML Superstructure Specification 2.4.1, p. 37; UML 2.5, p. 228). The propery declarations induced by a more than binary association result in a query operation.

[6]In UML, each Property may have AggregationKind composite. However, such an aggregation kind has no semantic meaning when the property is not a member end of an association: the UML Superstructure Specification 2.4.1 does not mention the aggregation kind in the description of the semantics of Property, and UML 2.5 explains the use of aggregations for Property as "to model circumstances in which *one instance* is used to group together a set of instances" (p. 112, our emphasis). Moreover, composite properties, i.e., properties with aggregation kind composite can only be member ends of binary associations (UML Superstructure Specification 2.4.1, p. 37; UML 2.5, p. 228) and their multiplicity must not exceed one (UML Superstructure Specification 2.4.1, p. 126; UML 2.5, p. 155). We thus separate composition declarations from general association declarations.

[7]In UML, member end names need not be unique. However, for (1) a simpler handling of selecting a particular member end in the sentences and avoiding the use of number selectors, and (2) making the notion of member ends "owned" by a class/data type, we add this constraint. An association declaration violating this uniqueness constraints can easily be transformed into an association declaration satisfying it by decorating member end names with the numbers $1, \dots, r$.

[8]All member ends are instances of Property; UML Superstructure Specification 2.4.1, p. 36; UML 2.5, p. 206.

declarations. (Note that by the uniqueness of attribute names along the generalisation hierarchy only a single attribute with name p_i may exist.)

A *class/data type net morphism* $\sigma = (\gamma, \varphi, \alpha, \mu) : \Sigma = ((C, \leq_C), P, A, M) \to \mathrm{T} = ((D, \leq_D), Q, B, N)$ is given by

- a class/data type hierarchy map $\gamma : (C, \leq_C) \to (D, \leq_D)$;
- an attribute declaration map $\varphi : P \to Q$ such that if $\varphi(c.p : \tau[c']) = d.q : \tau'[d'] \in Q$, then $d = \gamma(c)$, $d' = \gamma(c')$, and $\tau = \tau'$;
- a query operation declaration map $\rho : O \to R$ such that if $\rho(c.q(x_1 : \tau_1[c_1], \dots, x_r : \tau_r[c_r]) : \tau[c']) = d.r(x_1 : \tau'_1[d_1], \dots, x_r : \tau'_r[d_r]) : \tau[d'] \in R$, then $d = \gamma(c)$, $d_i = \gamma(c_i)$, $d' = \gamma(c')$, $\tau'_i = \tau_i$ and $\tau = \tau'$;
- an association declaration map $\alpha : A \to B$ such that if $\alpha(a(p_1 : \tau_1[c_1], \dots, p_r : \tau_r[c_r])) = b(q_1 : \tau'_1[d_1], \dots, q_s : \tau'_s[d_s]) \in B$, then $r = s$ and $d_i = \gamma(c_i)$ and $\tau_i = \tau'_i$ for $1 \leq i \leq r$, and member ends owned by the association are mapped into owned member ends;
- a composition declaration map $\mu : M \to N$ such that if $\mu(m(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2])) = n(q_1 : \mathsf{Set}[d_1], \blacklozenge q_2 : \tau'_2[d_2]) \in N$, then $d_1 = \gamma(c_1)$, $d_2 = \gamma(c_2)$, and $\tau_2 = \tau'_2$, and member ends owned by the composition are mapped into owned member ends.

Class/data type nets as objects and class/data type net morphisms as morphisms form the category of *class/data type nets*, denoted by Cl.

For the example in Fig. E.1 we have

Classes/data types: Net, Station, Line, Connector, Unit, Track, Point, Linear, Boolean, UnlimitedNatural, Integer, Real, String

Generalisations: Point $\leq$ Unit, Linear $\leq$ Unit

Properties: Line.linear : Set[Boolean], Track.linear : Set[Boolean],
Net.station : Set[Station], Net.line : Set[Line],
Station.net : Set[Net], Station.unit : Set[Unit], Station.track : Set[Track],
Line.net : Set[Net], Line.linear : Set[Linear],
Connector.unit : Set[Unit],
Unit.station : Set[Station], Unit.connector : Set[Connector],
Track.station : Set[Station], Track.linear : Set[Linear],
Linear.track : Set[Track], Linear.line : Set[Line]

Associations: l2l(line : Set[Line], linear : Set[Linear]),
l2t(linear : Set[Linear], track : Set[Track]),
c2u(connector : Set[Connector], unit : Set[Unit])

Compositions: n2s(net : Set[Net], ◆station : Set[Station]),
n2l(net : Set[Net], ◆line : Set[Line]),
s2u(station : Set[Station], ◆unit : Set[Unit]),
s2t(station : Set[Station], ◆track : Set[Track])

Here all member ends are owned by class/data types.

E.3. Models

As stated above, models (in the sense of the term model defined in clause 4) of UML class diagrams are obtained via a translation to Common Logic.

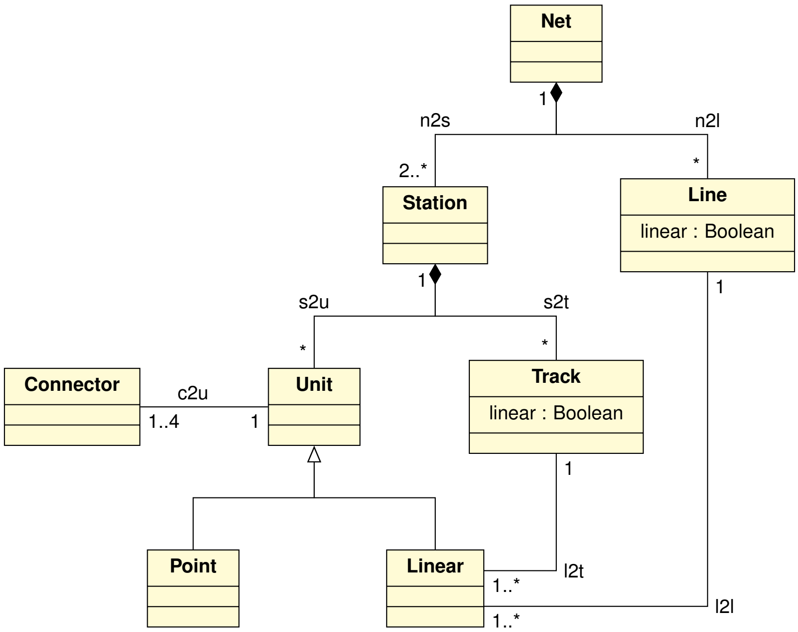


Figure E.1.: Sample UML class diagram.

For a classifier net $\Sigma = ((C, \leq_C), K, P, M, A)$, we define a Common Logic theory $CL(\Sigma)$ consisting of:

- for $c \in C$, a predicate[9] $\mathsf{CL}(c)$, such that
  - $\mathsf{CL}(\mathsf{Boolean}) =$ `buml:Boolean`,
  - $\mathsf{CL}(\mathsf{String}) =$ `buml:String`,
  - $\mathsf{CL}(\mathsf{Integer}) =$ `buml:Integer`,
  - $\mathsf{CL}(\mathsf{UnlimitedNatural}) =$ `form:NaturalNumber`,
  - $\mathsf{CL}(\mathsf{Real}) =$ `buml:Real`,
  - $\mathsf{CL}(\mathsf{c}) = c$, if c is an enumeration type with values $k_1, \ldots, k_n$. In this case, additionally, the Common Logic theory is augmented by `(not (=` $k_i \cdots k_j$`))` for $i \neq j$ and `(forall (x) (if (c x) (or (= x` k_1`)` $\cdots$ `(= x` k_n`))))`,
  - $\mathsf{CL}(\mathsf{List}[c]) =$ `form:Sequence`,
  - $\mathsf{CL}(\mathsf{Set}[c]) =$ `form:Set`,
  - $\mathsf{CL}(\mathsf{OrderedSet}[c]) =$ `form:OrderedSet`,
  - $\mathsf{CL}(\mathsf{Bag}[c]) =$ `form:Bag`,
  - $\mathsf{CL}(\mathsf{c}) = c$, if c a class name which is not one of the above.
- for each relation $c_1 \leq_C c_2$, an axiom `(forall (x) (if (`C_1` x) (`C_2` x)))`, where $C_1 = \mathsf{CL}(c_1)$, $C_2 = \mathsf{CL}(c_2)$,

[9]Strictly speaking, this is just a name.

- CL maps each attribute declaration $c.p : \tau[c'] \in P$ to a predicate $\mathsf{CL}(c.p)$ and axioms stating type-correctness and functionality:
  - `(forall (x y) (if (c.p x y) (c x)))`
  - `(forall (x y) (if (c.p x y) (`$\tau[c']$ y`)))` [10]
  - `(forall (x)`
    `(if (c x) (exists (y) (c.p x y))))`
  - `(forall (x y z)`
    `(if (and (c.p x y) (c.p x z))`
    `(= y z)))`
- CL maps each query operation declaration $c.q(x_1 : \tau_1[c_1], \ldots, x_r : \tau_n[c_r]) : \tau[c'] \in O$ to a predicate $\mathsf{CL}(c.q)$ and axioms stating type-correctness and functionality:[11]
  - `(forall (x` x_1 x_2 $\cdots$ x_n `y) (if (c.q x` x_1 x_2 $\cdots$ x_n `y) (`c x`)))`
  - `(forall (x` x_1 x_2 $\cdots$ x_n `y) (if (c.q x` x_1 x_2 $\cdots$ x_n `y) (`$\tau_i[c_i]$ x_i`)))` for each $i = 1 \ldots n$,[12]
  - `(forall (x` x_1 x_2 $\cdots$ x_n `y) (if (c.q x` x_1 x_2 $\cdots$ x_n `y) (`$\tau[c']$ y`)))`
  - `(forall (x` x_1 x_2 $\cdots$ x_n `y z)`
    `(if (and (c.q x` x_1 x_2 $\cdots$ x_n `y) (c.q x` x_1 x_2 $\cdots$ x_n `z))`
    `(= y z)))`
- CL maps each composition declaration $m(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2]) \in M$ to a constant $\mathsf{CL}(m)$ and axioms stating that $\mathsf{CL}(m)$ is a finite binary relation represented as a sequence of pairs of the correct type:

  ```
  (from:Sequence m)
  (forall (p) (if (form:sequence-member p m)
                  (and (form:Pair p) (c1 (form:first p)) (c2 (form:second p))))
  ```

  In case τ_2 is not present or $\tau_2 = \mathsf{Set}$, this is simplified to a binary relation directly represented as a binary predicate:
  `(forall (x y) (if (m x y) (and (`c_1 `x) (`c_2 `y))))`

- for any pair of composition declarations $m(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2]), m'(p_1' : \mathsf{Set}[c_1'], \blacklozenge p_2' : \tau_2'[c_2']) \in M$, an axiom stating "each instance has at most one owner":

  ```
  (forall (o o' i)
          (if (and (form:sequence-member (form:pair o i) m)
                   (form:sequence-member (form:pair o' i) m'))
              (= o o')))
  ```

  In case `m` is represented in the simplified way, `(form:sequence-member (form:pair o i) m)` is replaced by `(m o i)`, and analogously for `m'`.

[10] With `(`$\tau[c]$ `x)`, we abbreviate either (if τ is present)
`(and (`τ `x) (forall (m) (if (from:`τ`-member m x) (c' m)))).`
or (if τ is omitted) just `(c x)`.
[11] Query operations are modeled as partial functions: they may be undefined for certain arguments due to violation of multiplicity constraints.
[12] Note that the $\cdots$ here is meta notation, not a sequence marker!

- CL maps each association declaration $a(p_1 : \tau_1[c_1], \dots, p_r : \tau_r[c_r]) \in A$ to a predicate $\mathsf{CL}(a)$ and axioms stating that $\mathsf{CL}(a)$ is a finite relation represented as a sequence of tuples of the correct types (the latter again being represented as sequences)[13]:

  ```
  (from:Sequence a)
  (forall (t) (if (form:sequence-member t a)
            (exists (x1 ··· xr)
                   (and (c1 x1) ··· (cr xr)
  (= t (form:sequence-insert x1 (··· (form:sequence-insert xr form:empty-sequence)))))))
  ```

  In case that all the τ_i are omitted (or, equivalently, equal to Set), the representation is simplified to an n-ary predicate:

  ```
  (forall (x1 x2 ··· xn) (if (a x1 x2 ··· xn) (and (c1 x1) ··· (cn xn)))))
  ```

- the interpretation of a member end of a binary association declaration owned by a class/data type coincides with the interpretation of the attribute: if for $i \in \{1, 2\}$, $\mathbf{a}.p_i : \tau_i[c_i]$ for $\mathbf{a} = a(p_1 : \tau_1[c_1], p_2 : \tau_2[c_2]) \in A$ is owned by $c \in C$ with $c.p_i : \tau_i[c_i] \in P$, then

  ```
  (forall (o s)
       (if (c.p o s) (= s (form:seq2τi (form:selecti o a))))
  ```

  If $\mathbf{a}$ is represented in simplified form, then instead we use

  ```
  (forall (o s)
       (if (c.p o s) (forall (x) (iff (member x s) (a o x)))))
  ```

- the interpretation of a member end of a composition declaration owned by a class/data type coincides with the interpretation of the attribute: if for $i \in \{1, 2\}$, $\mathbf{m}.p : \tau_i[c_i]$ for $\mathbf{m} = m(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2]) \in M$ is owned by $c \in C$ with $c.p : \tau_i[c_i] \in P$, then

  ```
  (forall (o s)
       (if (c.p o s) (= s (form:seq2τi (form:selecti o m))))
  ```

  Again, if $\mathbf{m}$ is represented in simplified form, then instead we use

  ```
  (forall (o s)
       (if (c.p o s) (forall (x) (iff (member x s) (m o x)))))
  ```

It is straightforward to extend CL from signatures to signature morphisms.

Models. A Σ-model of the UML class diagram institution is just a $\mathsf{CL}(\Sigma)$-model in Common Logic. That is, the UML class diagram institution inherits models from Common Logic. Moreover, model reducts are inherited as well, using the action of CL on signature morphisms.

[13]Ignoring the annotations τ_i in the interpretation of an association is intentional, see OMG UML version 2.5 (ptc/2013-09-05) in section 11.5.3: "When one or more ends of the Association have isUnique =false, it is possible to have several links associating the same set of instances. In such a case, links carry an additional identifier apart from their end values. When one or more ends of the Association are ordered, links carry ordering information in addition to their end values." Similarly in UML Superstructure Specification 2.4.1, p. 37. We cover the additional information required for links by using sequences of tuples.

E.4. Sentences

The set of *multiplicity formulae* *Frm* is given by the following grammar:

$$
\begin{aligned}
\mathit{Frm} &::= \mathit{NumLiteral} \le \mathit{FunExpr} \\
&\mid \mathit{FunExpr} \le \mathit{NumLiteral} \\
\mathit{FunExpr} &::= \#\ \mathit{Attribute} \\
&\mid \#\ \mathit{Assocation}\ .\ \mathit{End} \\
&\mid \#\ \mathit{Composition}\ .\ \mathit{End} \\
&\mid \#\ \mathit{Operation}\ .\ \mathit{Param} \\
\mathit{Attribute} &::= \mathit{Classifier}\ .\ \mathit{End}{:}\mathit{Type} \\
\mathit{Association} &::= \mathit{Name}\ (\ \mathit{End} : \mathit{Type}(\ ,\ \mathit{End} : \mathit{Type})^*\ ) \\
\mathit{Composition} &::= \mathit{Name}\ (\ \mathit{End} : \mathsf{Set}\ [\ \mathit{Classifier}\ ], \blacklozenge\mathit{End} : \mathit{Type}\ ) \\
\mathit{Operation} &::= \mathit{Name}\ (\ (\ \mathit{NumLiteral} \le \mathit{Param} \le \mathit{NumLiteral}{:}\ \mathit{Type},\ )^*\ ) : \mathit{Type} \\
\mathit{Type} &::= \mathit{Annot}\ [\ \mathit{Classifier}\ ] \\
\mathit{Classifier} &::= \mathit{Name} \\
\mathit{End} &::= \mathit{Name} \\
\mathit{Param} &::= \mathit{Name} \\
\mathit{Annot} &::= \mathsf{OrderedSet} \mid \mathsf{Set} \mid \mathsf{Sequence} \mid \mathsf{Bag} \\
\mathit{NumLiteral} &::= \mathsf{0} \mid \mathsf{1} \mid \cdots
\end{aligned}
$$

where *Name* is a set of names and *NumLiteral* is assumed to be equipped with an appropriate function $[\![-]\!] : \mathit{NumLiteral} \to \mathbb{Z}$.

The set of Σ-*multiplicity constraints* $\mathit{Mult}(\Sigma)$ for a class/data type net Σ is given by the multiplicity formulae in *Frm* such that all mentioned elements of *Association* and *Composition* correspond to association declarations and composition declarations of Σ, respectively, and the member end name mentioned in the clauses of *FunExpr* occur in the mentioned association and composition, respectively.

The *translation* of a formula $\varphi \in \mathit{Mult}(\Sigma)$ along a class/data type net morphism σ, written as $\sigma(\varphi)$, is given by applying σ to associations, compositions, and member end names.

Example   For the example in Fig. E.1 we have

$$
\begin{aligned}
&2 \le \#\mathsf{n2s}(\mathsf{net} : \mathsf{Set}[\mathsf{Net}], \blacklozenge\mathsf{station} : \mathsf{Set}[\mathsf{Station}]).\mathsf{station} \\
&\#\mathsf{n2s}(\mathsf{net} : \mathsf{Set}[\mathsf{Net}], \blacklozenge\mathsf{station} : \mathsf{Set}[\mathsf{Station}]).\mathsf{net} = 1 \\
&\#\mathsf{n2l}(\mathsf{net} : \mathsf{Set}[\mathsf{Net}], \blacklozenge\mathsf{line} : \mathsf{Set}[\mathsf{Line}]).\mathsf{net} = 1 \\
&\#\mathsf{s2u}(\mathsf{station} : \mathsf{Set}[\mathsf{Station}], \blacklozenge\mathsf{unit} : \mathsf{Set}[\mathsf{Unit}]).\mathsf{station} = 1 \\
&\#\mathsf{s2t}(\mathsf{station} : \mathsf{Set}[\mathsf{Station}], \blacklozenge\mathsf{track} : \mathsf{Set}[\mathsf{Track}]).\mathsf{station} = 1 \\
&1 \le \#\mathsf{c2u}(\mathsf{connector} : \mathsf{Set}[\mathsf{Connector}], \mathsf{unit} : \mathsf{Set}[\mathsf{Unit}]).\mathsf{unit} \le 4 \\
&\#\mathsf{c2u}(\mathsf{connector} : \mathsf{Set}[\mathsf{Connector}], \mathsf{unit} : \mathsf{Set}[\mathsf{Unit}]).\mathsf{connector} = 1 \\
&1 \le \#\mathsf{l2t}(\mathsf{track} : \mathsf{Set}[\mathsf{Track}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]).\mathsf{track} \\
&\#\mathsf{l2t}(\mathsf{track} : \mathsf{Set}[\mathsf{Track}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]).\mathsf{linear} = 1 \\
&1 \le \#\mathsf{l2t}(\mathsf{line} : \mathsf{Set}[\mathsf{Line}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]).\mathsf{line} \\
&\#\mathsf{l2l}(\mathsf{line} : \mathsf{Set}[\mathsf{Line}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]).\mathsf{linear} = 1
\end{aligned}
$$

where we write "$=$" and "$- \le - \le -$" as respective abbreviations for two inequations using "$\le$".

E.5. Satisfaction Relation

The satisfaction relation is inherited from Common Logic, using a translation $\mathsf{CL}(\_)$ of multiplicity formulas to Common Logic. That is, given a UML class and object diagram Σ, a multiplicity formula φ and a Σ-model M (the latter amounts to a $\mathsf{CL}(\Sigma)$-model M in Common Logic), we define

$$M \models_\Sigma \varphi \text{ iff } M \models_{\mathsf{CL}(\Sigma)} \mathsf{CL}(\varphi)$$

The translation of multiplicity formulas to Common Logic is as follows:

- $\mathsf{CL}(\ell \leq \#c.p : \tau[c']) =$
```
(forall (x y n)
     (if (and (c.p x y) (form:τ-size y n)) (buml:leq [[ℓ]] n))
```
- $\mathsf{CL}(\ell \leq \#a(p_1 : \tau_1[c_1], \ldots, p_r : \tau_r[c_r]).p_i =$
```
(forall (x_1 ··· x_{i-1} x_{i+1} ··· x_r)
     (if (and (c_1 x_1) ··· (c_{i-1} x_{i-1}) (c_{i+1} x_{i+1}) ··· (c_r x_r)
               (form:sequence-size
                    (form:n-select a i [x_1 ··· x_{i-1} x_{i+1} ··· x_r]) n))
          (buml:leq [[ℓ]] n)))
```
  If a is represented in simplified form, we use instead:
  $\mathsf{CL}(\ell \leq \#a(p_1 : \tau_1[c_1], \ldots, p_r : \tau_r[c_r]).p_i =$
```
(forall (x_1 ··· x_{i-1} x_{i+1} ··· x_r)
     (if (and (c_1 x_1) ··· (c_{i-1} x_{i-1}) (c_{i+1} x_{i+1}) ··· (c_r x_r))
              (exists (y_1 ··· y_[[ℓ]])
                  (and (not (= (y_1 y_2))) ··· (not (= (y_[[ℓ]]-1 y_[[ℓ]])))
                        (a x_1 ··· x_{i-1} y_1 x_{i+1} ··· x_r)
                        ···
                        (a x_1 ··· x_{i-1} y_[[ℓ]] x_{i+1} ··· x_r) ))))
```
- $\mathsf{CL}(\ell \leq \#m(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2]).p_i) =$
```
(forall (x)
     (if (and (c_{3-i} x) (form:τ-size (form:select_i x m) n))
          (buml:leq [[ℓ]] n))
```
  If m is represented in simplified form, we use instead:
  $\mathsf{CL}(\ell \leq \#m(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2]).p_1) =$
```
(forall (x)
     (if (c_2 x)
              (exists (y_1 ··· y_[[ℓ]])
                  (and (not (= (y_1 y_2))) ··· (not (= (y_[[ℓ]]-1 y_[[ℓ]])))
                        (m y_1 x)
                        ···
                        (m y_[[ℓ]] x) ))))
```
  $\mathsf{CL}(\ell \leq \#m(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2]).p_2) =$
```
(forall (x)
     (if (c_1 x)
              (exists (y_1 ··· y_[[ℓ]])
                  (and (not (= (y_1 y_2))) ··· (not (= (y_[[ℓ]]-1 y_[[ℓ]])))
                        (m x y_1)
                        ···
                        (m x y_[[ℓ]]) ))))
```

- $\mathsf{CL}(\ell \leq \#c.q(\ell_1 \leq f_1 \leq \ell'_1 : \tau_1[c_1], \ldots, \ell_k \leq f_k \leq \ell'_k : \tau_k[c_k]) : \tau[c']) =$

```
(forall (x x_1 x_2 ··· x_n)
     (if (and (c.q x x_1 x_2 ··· x_n y)
               (form:τ-size x_1 n_1)
               ...
               (form:τ-size x_k n_k)
               (form:τ-size y n)
               (buml:leq ⟦ℓ_1⟧ n_1)
               (buml:leq n_1 ⟦ℓ'_1⟧)
               ...
               (buml:leq ⟦ℓ_k⟧ n_k)
               (buml:leq n_k ⟦ℓ'_k⟧) )
          (buml:leq ⟦ℓ⟧ n)))
```

where $[\![-]\!] : \mathit{NumLit} \to \mathbb{Z}$ maps a numerical literal to an integer, and $[x_1 \cdots x_n]$ abbreviates `(form:sequence-insert` x_1 $\cdots$ `(form:sequence-insert` x_n `form:empty-sequence))`. The translation for $\mathit{FunExpr} \leq \mathit{NumLiteral}$ is analogous. In case of simplified representation, we need to replace the existence of $[\![\ell]\!]$ distinct individuals with a statement expressing that if $[\![\ell]\!] + 1$ individuals have the specified property, at least two of them must be equal.

F. Annex (informative): Conformance of Essential MOF with DOL

Since essential MOF (EMOF) is a sublanguage of UML class diagrams, we can define a corresponding institution that is a subinstitution of that for UML class diagrams (see annex E).

G. Annex (informative): Conformance of TPTP with DOL

TPTP [62, 64, 63] is a language spoken by dozens of first-order theorem provers, and large libraries have been formalized in TPTP. The underlying logic is unsorted first-order logic. In [22], many-sorted first has been formalized as an institution; the single-sorted sublogic (using only a fixed set of sorts $\{s\}$ is isomorphic to unsorted first-order logic.

H. Annex (informative): Conformance of CASL with DOL

CASL [12] extends many-sorted first-order logic with partial functions and subsorting. It also provides induction sentences, expressing the (free) generation of datatypes. CASL has been presented as an institution in [51, 12]. We here only sketch this institution.

CASL signatures consist of a set S of sorts with a subsort relation $\leq$ between them together with families $\{PF_{w,s}\}_{w \in S^*, s \in S}$ of partial functions, $\{TF_{w,s}\}_{w \in S^*, s \in S}$ of total functions and $\{P_w\}_{w \in S^*}$ of predicate symbols. If Σ is a signature, two operation symbols with the same name f and with profiles $w \to s$ and $w' \to s'$, denoted $f_{w,s}$ and $f_{w',s'}$, are in the overloading relation if there are $w_0 \in S^*$ and $s_0 \in S$ such that $w_0 \leq w, w'$ and $s, s' \leq s_0$. Overloading of predicates is defined in a similar way. Signature morphisms consist of maps taking sort, function and predicate symbols respectively to a symbol of the same kind in the target signature, and they must preserve subsorting, typing of function and predicate symbols and totality of function symbols, and overloading.

For a signature Σ, terms are formed starting with variables from a sorted set X using applications of function symbols to terms of appropriate sorts, while sentences are partial first-order formulas extended with *sort generation constraints* which are triples (S', F', σ') such that $\sigma' : \Sigma' \to \Sigma$ and S' and F' are respectively sort and function symbols of Σ'. Partial first-order formulas are translated along a signature morphism $\varphi : \Sigma \to \Sigma''$ by replacing symbols as prescribed by φ while sort generation constraints are translated by composing the morphism σ' in their third component with φ.

Models interpret sorts as nonempty sets such that subsorts are injected into supersorts, partial/total function symbols as partial/total functions and predicate symbols as relations, such that the embeddings of subsorts into supersorts are monotone w.r.t. overloading.

The satisfaction relation is the expected one for partial first-order sentences. A sort generation constraint (S', F', σ') holds in a model M if the carriers of the reduct of M along σ' of the sorts in S' are generated by function symbols in F'.

I. Annex (informative): A Core Logic Graph

This annex provides a core heterogeneous environment that could be used as a basis for semantics of DOL as defined in Sec. 11.

I.1. Languages

The OMS languages that we selected are those whose conformance with DOL is established in the preceding, normative annexes (OWL 2 DL in annex B, Common Logic in annex C, RDFS in annex D, CASL in annex H, UML class diagrams in annex E and TPTP in annex G). The graph is shown in Figure I.1. Its nodes refer to the following OMS languages and profiles:

- RDF W3C/TR REC-rdf11-concepts:2014
- RDF Schema W3C/TR REC-rdf11-schema:2014
- EL, QL, RL (all being profiles of OWL) W3C/TR REC-owl2-profiles:2009
- OWL W3C/TR REC-owl2-syntax:2009
- CL (Common Logic) ISO/IEC 24707:2007
- UML class diagrams OMG Unified Modeling Language (UML) specification 2.4.1
- CASL [12] and its sublanguage classical first-order logic (FOL)
- TPTP

The list of chosen languages includes those ones required as mandatory ones in the RFP. Since these are only ontology and modeling languages, also a specification language is included, namely the Common Algebraic Specification Language (CASL). The list of language translations, given below, comprises standard translations from the literature, as well as further translations that are considered useful for logical interoperability:

- EL $\rightarrow$ OWL
- QL $\rightarrow$ OWL
- RL $\rightarrow$ OWL
- RDF $\rightarrow$ RDFS
- RDFS $\rightarrow$ OWL
- OWL $\rightarrow$ CASL.**FOL**
- CASL.**FOL** $\rightarrow TPTP$
- $TPTP \rightarrow$ CASL.**FOL**
- CASL.$FOL \rightarrow$ CL
- CASL.$FOL \rightarrow$ CASL
- $UML - CD \rightarrow$ CL.

The translations are specified in [50, 52]. Properties of translations have been introduced in section 11.1. All translations are marked as default translations.

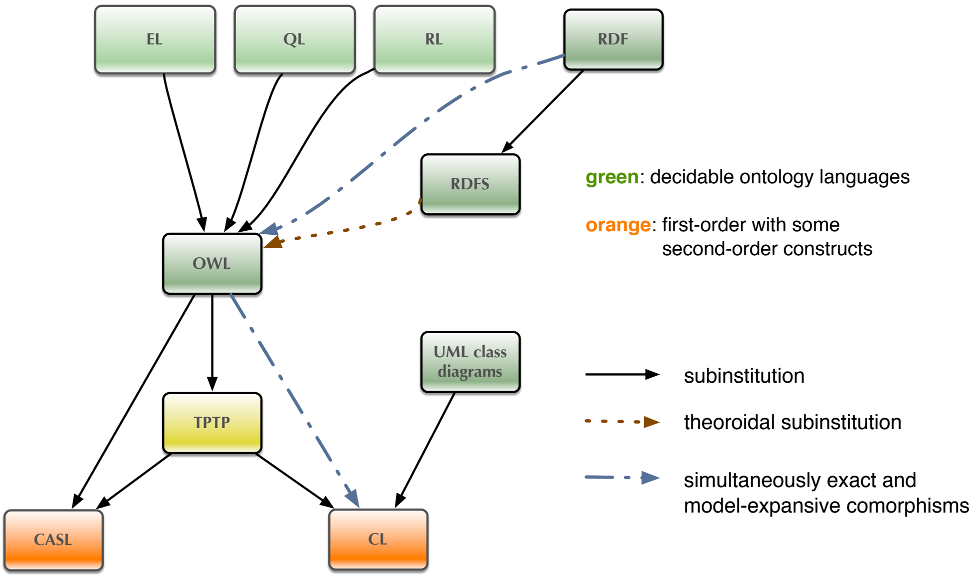


Figure I.1.: Translations between conforming OMS languages

I.2. Logics

The logics giving the semantics of these languages are listed below:

- RDF and RDFS, supported respectively by RDF and RDFS
- $\mathcal{EL}++$, supported by the language EL
- DL-Lite$_R$, supported by QL
- RL, supported by RL
- $\mathcal{SROIQ}(D)$, supported by OWL
- CL, supported by CL
- $SubPCFOL^{=}_{ms}$, supported by Casl
- **FOL**, supported by Casl.FOL and $TPTP$
- $UML - CD$, supported by $UML - CD$.

The institution comorphisms between these logics are

- $\mathcal{EL}++ \to \mathcal{SROIQ}(D)$
- DL-Lite$_R$ $\to \mathcal{SROIQ}(D)$
- RL $\to \mathcal{SROIQ}(D)$
- RDF $\to$ RDFS
- RDFS $\to \mathcal{SROIQ}(D)$
- $\mathcal{SROIQ}(D) \to$ Casl.**FOL**

- **FOL** $\rightarrow$ CL
- **FOL** $\rightarrow$ $SubPCFOL^{=}_{ms}$
- $UML - CD \rightarrow$ CL.

All of them are selected as default logic translations. There are no institution morphisms. The partial union operation between logics is given in the tables below, where $\perp$ denotes undefinedness:

Union	$\mathcal{EL}++$	DL-Lite$_R$	RL	RDF	RDFS
$\mathcal{EL}++$	$\mathcal{EL}++$	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$
DL-Lite$_R$	$\mathcal{SROIQ}(D)$	DL-Lite$_R$	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$
RL	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$	RL	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$
RDF	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$	RDF	RDFS
RDFS	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$	RDFS	RDFS
$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$
FOL	**FOL**	**FOL**	**FOL**	**FOL**	**FOL**
$SubPCFOL^{=}_{ms}$	$SubPCFOL^{=}_{ms}$	$SubPCFOL^{=}_{ms}$	$SubPCFOL^{=}_{ms}$	$SubPCFOL^{=}_{ms}$	$SubPCFOL^{=}_{ms}$
UML-CD	CL	CL	CL	CL	CL
CL	CL	CL	CL	CL	CL

Union	$\mathcal{SROIQ}(D)$	**FOL**	$SubPCFOL^{=}_{ms}$	UML-CD	CL
$\mathcal{EL}++$	$\mathcal{SROIQ}(D)$	**FOL**	$SubPCFOL^{=}_{ms}$	CL	CL
DL-Lite$_R$	$\mathcal{SROIQ}(D)$	**FOL**	$SubPCFOL^{=}_{ms}$	CL	CL
RL	$\mathcal{SROIQ}(D)$	**FOL**	$SubPCFOL^{=}_{ms}$	CL	CL
RDF	$\mathcal{SROIQ}(D)$	**FOL**	$SubPCFOL^{=}_{ms}$	CL	CL
RDFS	$\mathcal{SROIQ}(D)$	**FOL**	$SubPCFOL^{=}_{ms}$	CL	CL
$\mathcal{SROIQ}(D)$	$\mathcal{SROIQ}(D)$	**FOL**	$SubPCFOL^{=}_{ms}$	CL	CL
FOL	**FOL**	**FOL**	$SubPCFOL^{=}_{ms}$	CL	CL
$SubPCFOL^{=}_{ms}$	$\mathcal{SROIQ}(D)$	**FOL**	$SubPCFOL^{=}_{ms}$	$\perp$	$\perp$
UML-CD	CL	CL	$\perp$	UML-CD	CL
CL	CL	CL	$\perp$	CL	CL

The other assumptions on the logics in the heterogeneous logical environment hold in the expected way.

I.3. Serializations

The following syntaxes are part of the heterogeneous logical environments:

- Turtle, supported by OWL, EL, QL, RL, RDF, RDFS
- RDF-XML, supported by OWL, EL, QL, RL, RDF, RDFS
- OWL 2 XML, supported by OWL, EL, QL, RL
- Manchester Syntax, supported by OWL, EL, QL, RL
- TPTP, supported by TPTP
- CASL, supported by CASL
- XMI, supported by UML-CD
- XCL, supported by CL
- CLIF, supported by CL

I.4. Language and Logic Translations

I.4.1. EL → OWL and $\mathcal{EL}++ \to \mathcal{SROIQ}(D)$

EL → OWL is the sublanguage inclusion obtained by the syntactic restriction according to the definition of EL, see W3C/TR REC-owl2-profiles:2009. Since by definition, $\mathcal{EL}++$ is a syntactic restriction of $\mathcal{SROIQ}(D)$, $\mathcal{EL}++ \to \mathcal{SROIQ}(D)$ is the corresponding sublogic inclusion.

I.4.2. QL → OWL and DL-Lite$_R \to \mathcal{SROIQ}(D)$

QL → OWL is the sublanguage inclusion obtained by the syntactic restriction according to the definition of QL, see W3C/TR REC-owl2-profiles:2009. Since by definition, DL-Lite$_R$ is a syntactic restriction of $\mathcal{SROIQ}(D)$, DL-Lite$_R \to \mathcal{SROIQ}(D)$ is the corresponding sublogic inclusion.

I.4.3. RL → OWL and RL → $\mathcal{SROIQ}(D)$

RL → OWL is the sublanguage inclusion obtained by the syntactic restriction according to the definition of RL, see W3C/TR REC-owl2-profiles:2009. Since by definition, RL is a syntactic restriction of $\mathcal{SROIQ}(D)$, RL → $\mathcal{SROIQ}(D)$ is the corresponding sublogic inclusion.

I.4.4. SimpleRDF → RDF

SimpleRDF → RDF is an obvious inclusion, except that SimpleRDF resources need to be renamed if they happen to have a predefined meaning in RDF. The model translation needs to forget the fixed parts of RDF models, since this part can always reconstructed in a unique way, we get an isomorphic model translation.

I.4.5. RDF → RDFS

This is entirely analogous to SimpleRDF → RDF.

I.4.6. SimpleRDF → $\mathcal{SROIQ}(D)$

A SimpleRDF signature is translated to $\mathcal{SROIQ}(D)$ by providing a class P and three roles sub, $pred$ and obj (these reify the extension relation), and one individual per SimpleRDF resource. A SimpleRDF triple (s, p, o) is translated to the $\mathcal{SROIQ}$ (D) sentence

$$\top \sqsubseteq \exists U.(\exists sub.\{s\} \sqcap \exists pred.\{p\} \sqcap \exists obj.\{o\}).$$

From an $\mathcal{SROIQ}$ (D) model $\mathcal{I}$, obtain a SimpleRDF model by inheriting the universe and the interpretation of individuals (then turned into resources). The interpretation $P^{\mathcal{I}}$ of P gives P_m, and EXT_m is obtained by de-reifying, i.e.

$$EXT_m(x) := \{(y, z) | \exists u.(u, x) \in pred^{\mathcal{I}}, (u, y) \in sub^{\mathcal{I}}, (u, z,) \in obj^{\mathcal{I}}\}.$$

RDF → $\mathcal{SROIQ}(D)$ is defined similarly. The theory of RDF built-ins is (after translation to $\mathcal{SROIQ}$ (D)) added to any signature translation. This ensures that the model translation can add the built-ins.

I.4.7. OWL $\rightarrow FOL$

Translation of signatures

$\Phi((\mathbf{C}, \mathbf{R}, \mathbf{I})) = (F, P)$ with

- function symbols: $F = \{a^{(1)} | a \in \mathbf{I}\}$
- predicate symbols $P = \{A^{(1)} | A \in \mathbf{C}\} \cup \{R^{(2)} | R \in \mathbf{R}\}$

Translation of sentences

Concepts are translated as follows:

- $\alpha_x(A) = A(x)$
- $\alpha_x(\neg C) = \neg \alpha_x(C)$
- $\alpha_x(C \sqcap D) = \alpha_x(C) \wedge \alpha_x(D)$
- $\alpha_x(C \sqcup D) = \alpha_x(C) \vee \alpha_x(D)$
- $\alpha_x(\exists R.C) = \exists y.(R(x, y) \wedge \alpha_y(C))$
- $\alpha_x(\exists U.C) = \exists y.\alpha_y(C)$
- $\alpha_x(\forall R.C) = \forall y.(R(x, y) \rightarrow \alpha_y(C))$
- $\alpha_x(\forall U.C) = \forall y.\alpha_y(C)$
- $\alpha_x(\exists R.\mathrm{Self}) = R(x, x)$
- $\alpha_x(\leq nR.C) = \forall y_1, \ldots, y_{n+1}. \bigwedge_{i=1,\ldots,n+1}(R(x, y_i) \wedge \alpha_{y_i}(C)) \rightarrow \bigvee_{1 \leq i < j \leq n+1} y_i = y_j$
- $\alpha_x(\geq nR.C) = \exists y_1, \ldots, y_n. \bigwedge_{i=1,\ldots,n}(R(x, y_i) \wedge \alpha_{y_i}(C)) \wedge \bigwedge_{1 \leq i < j \leq n} y_i \neq y_j$
- $\alpha_x(\{a_1, \ldots a_n\}) = (x = a_1 \vee \ldots \vee x = a_n)$

For inverse roles R^-, $R^-(x, y)$ has to be replaced by $R(y, x)$, e.g.

$$\alpha_x(\exists R^-.C) = \exists y.(R(y, x) \wedge \alpha_y(C))$$

This rule also applies below.

Sentences are translated as follows:

- $\alpha_\Sigma(C \sqsubseteq D) = \forall x. (\alpha_x(C) \rightarrow \alpha_x(D))$
- $\alpha_\Sigma(a : C) = \alpha_x(C)[a/x]$[1]
- $\alpha_\Sigma(R(a, b)) = R(a, b)$
- $\alpha_\Sigma(R \sqsubseteq S) = \forall x, y.R(x, y) \rightarrow S(x, y)$
- $\alpha_\Sigma(R_1; \ldots; R_n \sqsubseteq R) =$
  $\forall x, y.(\exists z_1, \ldots z_{n-1}.R_1(x, z_1) \wedge R_2(z_1, z_2) \wedge \ldots \wedge R_n(z_{n-1}, y)) \rightarrow R(x, y)$
- $\alpha_\Sigma(\mathrm{Dis}(R_1, R_2)) = \neg \exists x, y.R_1(x, y) \wedge R_2(x, y)$
- $\alpha_\Sigma(\mathrm{Ref}(R)) = \forall x.R(x, x)$
- $\alpha_\Sigma(\mathrm{Irr}(R)) = \forall x.\neg R(x, x)$
- $\alpha_\Sigma(\mathrm{Asy}(R)) = \forall x, y.R(x, y) \rightarrow \neg R(y, x)$
- $\alpha_\Sigma(\mathrm{Tra}(R)) = \forall x, y, z.R(x, y) \wedge R(y, z) \rightarrow R(x, z)$

[1] $t[a/x]$ means "in t, replace x by a".

Translation of models

- For $M' \in \mathrm{Mod}^{FOL}(\Phi\Sigma)$ define $\beta_\Sigma(M') := (\Delta, \cdot^I)$ with $\Delta = |M'|$ and $A^I = M'_A, a^I = M'_a, R^I = M'_R$.

Proposition 16 $C^{\mathcal{I}} = \{m \in M'_{Thing} | M' + \{x \mapsto m\} \models \alpha_x(C)\}$

Proof. By Induction over the structure of C.

- $A^{\mathcal{I}} = M'_A = \{m \in M'_{Thing} | M' + \{x \mapsto m\} \models A(x)\}$
- $(\neg C)^{\mathcal{I}} = \Delta \backslash C^{\mathcal{I}} =^{I.H.} \Delta \backslash \{m \in M'_{Thing} | M' + \{x \mapsto m\} \models \alpha_x(C)\} = \{m \in M'_{Thing} | M' + \{x \mapsto m\} \models \neg\alpha_x(C)\}$

□

The satisfaction condition holds as well.

I.4.8. $FOL \to$ CL

This comorphism maps classical first-order logic (FOL) to Common Logic.

A FOL signature is translated to CL.Fol by turning all constants into discourse names, and all other function symbols and all predicate symbols into non-discourse names. A FOL sentence is translated to CL.Fol by a straightforward recursion, the base being translations of predications:

$$\alpha_\Sigma(P(t_1, \ldots, t_n)) = (P\ \alpha_\Sigma(t_1)\ \ldots\ \alpha_\Sigma(t_n))$$

Within terms, function applications are translated similarly:

$$\alpha_\Sigma(f(t_1, \ldots, t_n)) = (f\ \alpha_\Sigma(t_1)\ \ldots\ \alpha_\Sigma(t_n))$$

A CL.Fol model is translated to a FOL model by using the universe of discourse as FOL universe. The interpretation of constants is directly given by the interpretation of the corresponding names in CL.Fol. The interpretation of a predicate symbol P is given by using $rel^M(int^M(P))$ and restricting to the arity of P; similarly for function symbols (using fun^M). Both the satisfaction condition and model-expansiveness of the comorphism are straightforward.

I.4.9. OWL $\to$ CL

This comorphism is the composition of the comorphisms described in the previous two sections.

I.4.10. UML class diagrams $\to$ CL

This translation has been described in annex E. Translation of signatures is detailed in section E.3, translation of sentences in section E.5. Models are translated identically.

I.4.11. $FOL \to$ CASL

This is an obvious sublogic.

J. Annex (informative): Extended Logic Graph

This annex extends the graph of logics and translations given in annex I by a list of OMS language whose conformance with DOL will be established through the registry. The graph is shown in Figure J.1. Its nodes are included in the following list of OMS languages and profiles (in addition to those mentioned in annex I):

- PL (propositional logic)
- SimpleRDF (RDF triples without a reserved vocabulary)
- $\mathrm{OBO}^{\mathrm{OWL}}$ and OBO1.4
- RIF (Rule Interchange Format)
- EER (Enhanced Entity-Relationship Diagrams)
- Datalog
- ORM (object role modeling)
- the meta model of schema.org
- different diagram types of the UML (Unified Modeling Language), with possibly different logics according to different UML semantics
- SKOS (Simple Knowledge Organization System)
- $\mathrm{FOL}^{=}$ (untyped first-order logic, as used for the TPTP format)
- F-logic

The actual translations are specified in [50].

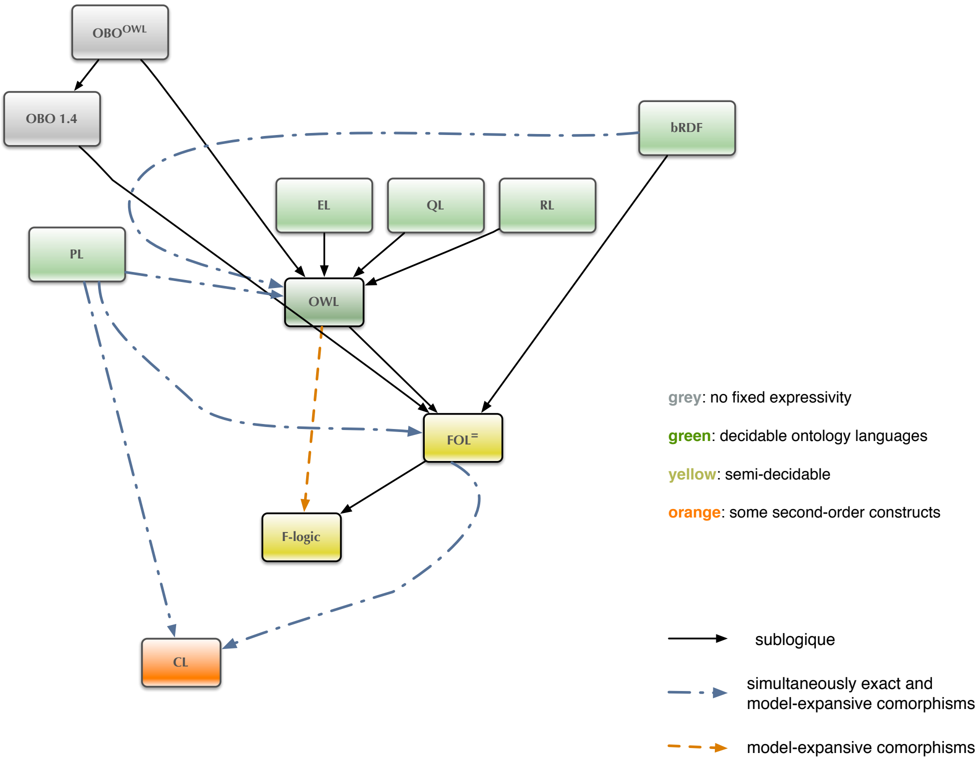


Figure J.1.: Translations between conforming OMS languages (extended)

K. Annex (informative): Example Uses of all DOL Constructs

Top-level declarations in libraries	
Top-level declaration	**Examples**
library ...	all examples
import IRI	Mereology
language IRI	Alignments, Publications
logic IRI	Alignments, Mereology
serialization IRI	Alignments, Mereology
PrefixMap	Mereology
oms IRI = OMS end	Alignments, Mereology
oms IRI = %consistent OMS end	PropositionalExamples, Mereology
oms IRI = %inconsistent OMS end	PropositionalExamples
oms IRI = %mono OMS end	section 7.7
oms IRI = %def OMS end	PropositionalExamples
network IRI = IRI, ..., IRI	Alignments
interpretation IRI : OMS to OMS = SymbolMap	Mereology
interpretation IRI : OMS to OMS = %cons SymbolMap	Engine
interpretation IRI : OMS to OMS = translation IRI	Mereology
refinement IRI = OMS refined via SymbolMap to OMS	section 7.7
refinement IRI = OMS refined via translation IRI to OMS	section 7.9
refinement IRI = IRI then IRI	section 7.7
refinement IRI = Network refined to Network	section 7.8
entailment IRI = OMS entails OMS	PropositionalExamples
entailment IRI = OMSName in Network entails OMS	section 7.8
entailment IRI = Network entails Network	section 7.8
equivalence IRI : OMS <-> OMS = OMS end	Algebra
module IRI : OMS of OMS for Symbols	section 7.3
alignment IRI : OMS to OMS = Correspondences	Alignments
alignment IRI : OMS to OMS = Correspondences assuming SingleDomain	[?]
alignment IRI : OMS to OMS = Correspondences assuming GlobalDomain	[?]
alignment IRI : OMS to OMS = Correspondences assuming ContextualizedDomain	[?]
query IRI = select ars where Sen in OMS	MyQuery
substitution IRI : OMS to OMS = SymbolMap	MyQuery
result IRI = IRIs for IRI	MyQuery

OMS	
OMS notation	**Examples**
BasicOMS	Alignments, Mereology
IRI	Alignments, Mereology
minimize { OMS }	BlocksWithCircumscription
OMS minimize Symbols var Symbols	BlocksWithCircumscription
OMS maximize Symbols var Symbols	BlocksWithCircumscription
free { OMS }	Datatypes
cofree { OMS }	Datatypes
OMS with SymbolMap	Alignments, section 7.7
OMS with translation IRI	Mereology
OMS hide SymbolItems	Algebra
OMS reveal Symbols	Datatypes
OMS hide along IRI	section 7.8
OMS extract Symbols	section 7.3
OMS remove Symbols	All_kinds_of_group_specifications
OMS forget Symbols	All_kinds_of_group_specifications
OMS keep Symbols	All_kinds_of_group_specifications
OMS select BasicOMS	All_kinds_of_group_specifications
OMS reject BasicOMS	All_kinds_of_group_specifications
OMS and OMS	Engine
OMS then OMS	Mereology
OMS then %ccons OMS	[44]
OMS then %mcons OMS	Propositional
OMS then %notccons OMS	[44]
OMS then %notmcons OMS	[44]
OMS then %mono OMS	Sorting
OMS then %def OMS	Persons
OMS then %implied OMS	BlocksWithCircumscription
logic IRI : OMS	all examples
language IRI : OMS	Mereology
serialization IRI : OMS	Mereology
combine NetworkElements	Alignments, Publications

K.1. Simple Examples in Propositional Logic

```
%prefix( :       <http://www.example.org/prop#>
         log:    <http://www.omg.org/spec/DOL/logics/> )%

                                         %% descriptions of logics ...

%% non-standard serialization built into Hets:
logic log:Propositional syntax ser:Prop/Hets

library PropositionalExamples

oms Consistent = %consistent
```

```
  props A, B
  . A => B
end

oms Inconsistent = %inconsistent
  props A
  . A /\ not A
end

oms SingleModel = %def
  props A, B
  . A /\ not B
end

entailment Ent = SingleModel entails { . not ( A=>B ) }
end


library PropositionalMereology

%% non-standard serialization built into Hets:
logic log:Propositional syntax ser:Prop/Hets

%% basic taxonomic information about mereology reused from DOLCE:
ontology Taxonomy = %conssistent
  props PT, T, S, AR, PD
  . S ∨ T ∨ AR ∨ PD ⟶ PT
%% PT is the top concept
  . S ∧ T  ⟶ ⊥              %% PD, S, T, AR are pairwise disjoint
  . T ∧ AR ⟶ ⊥
%% and so on
end
```

K.2. Engine Diagnosis and Repair

```
library Engine

logic Propositional

%% possible symptoms of an engine that is malfunctioning
spec EngineSymptoms =
  props black_exhaust, blue_exhaust, low_power, overheat,
        ping, incorrect_timing, low_compression
end

%% diagnosis derived from symptoms
spec EngineDiagnosis = EngineSymptoms
then %mcons
```

```
  props carbon_deposits,
        clogged_filter,
        clogged_radiator,
        defective_carburetor,
        worn_rings,
        worn_seals
  . overheat /\ not incorrect_timing => clogged_radiator
                          %(diagnosis1)%
  . ping /\ not incorrect_timing => carbon_deposits
                          %(diagnosis2)%
  . low_power /\ not incorrect_timing =>
              worn_rings \/ defective_carburetor \/ clogged_filter
                          %(diagnosis3)%
  . black_exhaust => defective_carburetor \/ clogged_filter
                          %(diagnosis4)%
  . blue_exhaust => worn_rings \/ worn_seals
                          %(diagnosis5)%
  . low_compression <=> worn_rings
                          %(diagnosis6)%
end

%% needed repair, derived from diagnosis
spec EngineRepair = EngineDiagnosis
then %cons
  props replace_auxiliary,
        repair_engine,
        replace_engine
  . worn_rings => replace_engine
                          %(rule_replace_engine)%
  . carbon_deposits \/ defective_carburetor \/ worn_seals =>
              repair_engine
                          %(rule_repair_engine)%
  . clogged_filter \/ clogged_radiator => replace_auxiliary
                          %(rule_replace_auxiliary)%
end

%% application to a specific case
spec MyObservedSymptoms =
  EngineSymptoms
then
  . overheat              %(symptom_overheat)%
  . not incorrect_timing  %(symptom_not_incorrect_timing)%
end

spec MyRepair =
  MyObservedSymptoms
and
  EngineRepair
end
```

```
spec Repair =
  prop repair
  . repair
end

interpretation repair1 : Repair to MyRepair = %cons
  repair |-> replace_engine end
interpretation repair2 : Repair to MyRepair = %cons
  repair |-> repair_engine end
interpretation repair3 : Repair to MyRepair = %cons
  repair |-> replace_auxiliary end
%% only repair3 is a valid interpretation. That is, 'replace_auxiliary'
%% is the required action
```

K.3. Mereology: Distributed and Heterogeneous Ontologies

```
%prefix( :      <http://www.example.org/mereology#>
         owl:   <http://www.w3.org/2002/07/owl#>
         log:   <http://www.omg.org/spec/DOL/logics/>
                                %% descriptions of logics ...
         trans: <http://www.omg.org/spec/DOL/translations/> )%
                                %% ... and translations

library Mereology

import PropositionalMereology

%% OWL Manchester syntax declaration:
language lang:OWL2 logic log:SROIQ syntax ser:OWL2/Manchester

%% Parthood in SROIQ, as far as easily expressible:
ontology BasicParthood =
  Class: ParticularCategory
        SubClassOf: Particular
                %% omitted similar declarations of the other classes
    DisjointUnionOf: SpaceRegion, TimeInterval, AbstractRegion, Perdurant
                %% pairwise disjointness more compact
                                %% thanks to an OWL built-in
  ObjectProperty: isPartOf
        Characteristics: Transitive
  ObjectProperty: isProperPartOf
        Characteristics: Asymmetric  SubPropertyOf: isPartOf
  Class: Atom
        EquivalentTo: inverse isProperPartOf only owl:Nothing
end             %% an atom has no proper parts

%% translate the logic, then rename the entities
```

```
interpretation TaxonomyToParthood : Taxonomy to BasicParthood =
  translation trans:PropositionalToSROIQ,
  PT ↦ Particular,       S ↦ SpaceRegion,
  T ↦ TimeInterval,      A ↦ AbstractRegion, %[ and so on ]%

logic log:CommonLogic syntax ser:CommonLogic/CLIF
                %% syntax: the Lisp-like CLIF dialect of Common Logic

%% ClassicalExtensionalParthood imports the OWL ontology from above,
%% translate it to Common Logic, then extend it there:
ontology ClassicalExtensionalParthood =
  BasicParthood with translation trans:SROIQtoCL
then
  . (forall (X) (if (or (= X S) (= X T) (= X AR) (= X PD))
                    (forall (x y z) (if (and (X x) (X y) (X z))
                                        (and
%% now list all the axioms:
        %% antisymmetry:
      (if (and (isPartOf x y) (isPartOf y x)) (= x y))
        %% transitivity; not combinable with asymmetry in OWL DL:
      (if (and (isProperPartOf x y) (isProperPartOf y z)) (isProperPartOf x z))
      (iff (overlaps x y) (exists (pt) (and (isPartOf pt x) (isPartOf pt y))))
      (iff (isAtomicPartOf x y) (and (isPartOf x y) (Atom x)))
      (iff (sum z x y)
           (forall (w) (iff
                          (overlaps w z)
                          (and (overlaps w x) (overlaps w y)))))
 %% existence of the sum:
      (exists (s) (sum s x y))
      )))))
%% definition of fusion
  . (forall (Set a) (iff (fusion Set a)
            (forall (b) (iff (overlaps b a)
                             (exists (c) (and (Set c) (overlaps c a)))))))
  }
```

K.4. Defined Concepts

```
library Persons
logic OWL

ontology Persons =
  Class Person
  Class Female
then %def
  Class: Woman  EquivalentTo: Person and Female
end
```

K.5. Blocks World: Minimization

```
library BlocksWithCircumscription
logic log:OWL

ontology Blocks =
  %% FIXED PART
  Class: Block
  Individual: B1 Types: Block
  Individual: B2 Types: Block DifferentFrom: B1
              %% B1 and B2 are different blocks
then
  %% CIRCUMSCRIBED PART
  minimize {
    Class: Abnormal
    Individual: B1 Types: Abnormal
       %% B1 is abnormal
  }
then
  %% VARYING PART
  Class: Ontable
  Class: BlockNotAbnormal
        EquivalentTo: Block and not Abnormal
        SubClassOf: Ontable
        %% Normally, a block is on the table
then %implied
  Individual: B2 Types: Ontable
     %% B2 is on the table
end

ontology Blocks_Alternative =
  Class: Block
  Class: Abnormal
  Individual: B1 Types: Block, Abnormal
  Individual: B2 Types: Block DifferentFrom: B1
              %% B1 and B2 are different blocks
              %% B1 is abnormal
  Class: Ontable
  Class: BlockNotAbnormal
        EquivalentTo: Block and not Abnormal
        SubClassOf: Ontable
        %% Normally, a block is on the table
  minimize Abnormal var Ontable, BlockNotAbnormal
then %implied
  Individual: B2 Types: Ontable
     %% B2 is on the table
end

ontology Blocks_Alternative2 =
```

```
  Class: Block
  Class: Normal
  Individual: B1 Types: Block, not Normal
  Individual: B2 Types: Block DifferentFrom: B1
              %% B1 and B2 are different blocks
              %% B1 is abnormal
  Class: Ontable
  Class: NormalBlock
        EquivalentTo: Block and Normal
        SubClassOf: Ontable
        %% Normally, a block is on the table
  maximize Normal var Ontable, BlockNotAbnormal
then %implied
  Individual: B2 Types: Ontable
     %% B2 is on the table
end
```

K.5.1. Alignments

```
%prefix( :     <http://www.example.org/alignment#>
         owl:   <http://www.w3.org/2002/07/owl#>
         log:   <http://www.omg.org/spec/DOL/logics/> %% descriptions of logics ...
         trans: <http://www.omg.org/spec/DOL/translations/> )% %% ... and translations

library Alignments

language lang:OWL2 logic log:SROIQ syntax ser:OWL2/Manchester

alignment Alignment1 : { Class: Woman } to { Class: Person } =
  Woman < Person
end

ontology AlignedOntology1 =
  combine Alignment1
end


ontology Onto1 =
  Class: Person
  Class: Woman SubClassOf: Person
  Class: Bank
end

ontology Onto2 =
  Class: HumanBeing
  Class: Woman SubClassOf: HumanBeing
  Class: Bank
end
```

```
alignment VAlignment : Onto1 to Onto2 =
  Person = HumanBeing,
  Woman = Woman
end

network N =
  1 : Onto1, 2 : Onto2, VAlignment
end

ontology VAlignedOntology =
  combine N
  %% 1:Person is identified with 2:HumanBeing
  %% 1:Woman is identified with 2:Woman
  %% 1:Bank and 2:Bank are kept distinct
end

ontology VAlignedOntologyRenamed =
  VAlignedOntology with 1:Bank |-> RiverBank, 2:Bank |-> FinancialBank
end
```

K.6. Distributed Description Logics

```
%prefix( :      <http://www.example.org/mereology#>
         owl:   <http://www.w3.org/2002/07/owl#>
         log:   <http://www.omg.org/spec/DOL/logics/> %% descriptions of logics ...
         trans: <http://www.omg.org/spec/DOL/translations/> )% %% ... and translations

library Publications

language lang:OWL2 logic log:SROIQ syntax ser:OWL2/Manchester

ontology Publications1 =
  Class: Publication
  Class: Article SubClassOf: Publication
  Class: InBook SubClassOf: Publication
  Class: Thesis  SubClassOf: Publication
  Class: MasterThesis  SubClassOf: Thesis
  Class: PhDThesis SubClassOf: Thesis
end

ontology Publications2 =
  Class: Thing
  Class: Article SubClassOf: Thing
  Class: BookArticle SubClassOf: Thing
  Class: Publication SubClassOf: Thing
  Class: Thesis  SubClassOf: Thing
end
```

```
ontology Publications_Combined =
combine
  1 : Publications1 with translation OWL2MS-OWL,
  2 : Publications2 with translation OWL2MS-OWL
  %% implicitly: Article ↦ 1:Article ...
  %%              Article ↦ 2:Article ...
  with translation MS-OWL2DDL
  %% implicitly added by translation MS-OWL2DDL:
  %% binary relation providing the bridge
then
  1:Publication ⊑⟶ 2:Publication
  1:PhdThesis ⊑⟶ 2:Thesis
  1:InBook ⊑⟶ 2:BookArticle
  1:Article ⊑⟶ 2:Article
  1:Article ⊒⟶ 2:Article
end


ontology Publications_Extended =
Publications with translation DDL2-ECO
  %% turns implicit domain-relation into default relation 'D'
  %% add E-connection style bridge rules on top
end


library Market


language lang:OWL2 logic log:SROIQ syntax ser:OWL2/Manchester
ontology Purchases =
combine
  1 : { Class: PurchaseOrder },
  2 : { ObjectProperty: Buyer
       ObjectProperty: Good
       ObjectProperty: BoughtBy }
  with translation OWL2DDLwithRoles
then
  1:PurchaseOrder -into-> 2:BoughtBy
%% means in FOL:
%% forall x 1PurchaseOrder(x) -> forall yz CR12(x,y,z) -> 2BoughtBy(y,z)
end
```

K.7. Algebra

```
%prefix( :     <http://www.example.org/alignment#>
         owl:   <http://www.w3.org/2002/07/owl#>
```

```
        log:   <http://www.omg.org/spec/DOL/logics/> %% descriptions of logics ...
        trans: <http://www.omg.org/spec/DOL/translations/> )% %% ... and translations

library Algebra

logic log:CommonLogic syntax ser:CommonLogic/CLIF

spec implicit_group =
(forall (x y z)
        (= (op x (op y z)) (op (op x y) z)))
(exists (e)
        (forall (x)
                (and    (= x (op e x))
                        (= x (op x e)))))
(forall (x)
        (exists (y)
                (and    (= x (op x (op x y)))
                        (= x (op x (op y x))))))
end

spec explicit_group =
(forall (x y z)
        (= (op x (op y z)) (op (op x y) z)))
(forall (x)     (and    (= x (op e x))
                        (= x (op x e)))))
(forall (x)
                (and    (= x (op x (op x (inv x))))
                        (= x (op x (op (inv x) x))))))
end

equivalence groups_equiv : implicit_group <-> { explicit_group hide e, inv }
end

equivalence e : algebra:BooleanAlgebra
                ↔ algebra:BooleanRing =
    x∧y = x·y
    x∨y = x+y+x·y
    ¬x = 1+x
    x·y = x∧y
    x+y = (x∨y) ∧ ¬(x∧y)
end

logic CASL

spec InterpolatedGroup =
  sort Elem
  ops 0:Elem; __+__:Elem*Elem->Elem; inv:Elem->Elem
  forall x,y,z:elem . x+0=x
                    . x+(y+z) = (x+y)+z
```

```
                    . x+inv(x) = 0
  forget inv
end

entailment ent = InterpolatedGroup
  entails { . forall x:Elem . exists y . Elem . x+y=0 }
end
```

K.7.1. Groups specified with different forms of hiding and forgetting

Groups and hiding

```
library All_kinds_of_group_specifications
logic CASL
spec Group_with_inverse =
  sort Elem
  ops 0:Elem; __+__:Elem*Elem->Elem; inv:Elem->Elem
  forall x,y,z:elem . x+0=x
                    . x+(y+z) = (x+y)+z
                    . x+inv(x)=0
end

spec Group_via_hiding =
  Group_with_inverse hide inv
end
```

The semantics of this specification is the class of all monoids that can be extended with an inverse, i.e. class of all groups. The effect is second-order quantification:

```
logic HasCASL
spec Group_in_second_order_logic =
  sort Elem
  ops 0:Elem; __+__:Elem*Elem->Elem;
  exists inv:Elem->Elem .
    forall x,y,z:elem . x+0=x
                      /\ x+(y+z) = (x+y)+z
                      /\ x+inv(x)=0
end
```

Groups and module extraction

```
logic CASL
spec Group_via_module_extraction_1 =
  Group_with_inverse remove inv
end
```

The semantics is just `Group_with_inverse`, since the module needs to be enlarged to the whole specification. This is of course unsatisfactory. A better use of module extraction is the following:

```
logic CASL
spec Group_with_implicit_inverse =
```

```
  sort Elem
  ops 0:Elem; __+__:Elem*Elem->Elem; inv:Elem->Elem
  forall x,y,z:elem . x+0=x
                    . x+(y+z) = (x+y)+z
                    . x+inv(x) = 0
                    . exists y:Elem . x+y=0
end

spec Group_via_module_extraction_2 =
  Group_with_implicit_inverse remove inv
end
```

The semantics of `Group_via_module_extraction_2` is just `Group_with_implicit_inverse`, because adding `inv` is conservative.

Groups via interpolation

```
logic CASL
spec Group_via_interpolation1 =
  Group_with_inverse forget inv
end
spec Group_via_interpolation2 =
  Group_with_inverse keep Elem, 0, __+__
end
```

Both specifications are equivalent, and they are equivalent to `Group_with_implicit_inverse`.

Groups and filtering

```
logic CASL
spec Group_via_Filtering_1 =
  Group_with_inverse reject inv
end
spec Group_via_Filtering_2 =
  Group_with_inverse select Elem, 0, __+__
end
```

Both specifications are equivalent, and they are equivalent to the following theory which just omits the inverse axioms (and hence does not specify groups):

```
logic CASL
spec Group_via_reject =
  sort Elem
  ops 0:Elem; __+__:Elem*Elem->Elem
  forall x,y,z:elem . x+0=x
                    . x+(y+z) = (x+y)+z
end
```

K.8. Queries

```
library MyQuery
logic CASL
spec Person =
  sort s
  pred Person:s
  op max,peter:Person
end
query MyQuery = select x where Person(x) in Person
end
substitution MySubst : { Person then op x:Person } to Person = x |-> max
end
result MyResult = MySubst for MyQuery
```

K.9. Datatypes

```
library Datatypes
logic CASL

spec Bag =
  sort Elem
  then free {
     sort Bag
     ops mt:Bag;
         __union__:Bag*Bag->Bag, assoc, comm, unit mt
           }
end

spec Stream =
  sort Elem
  then cofree {
     sort Stream
     ops head:Stream->Elem;
         tail:Stream->Stream
           }
end

spec Finite =
  sort Elem
  free type Nat ::= 0 | suc(Nat)
  op f: Nat ->? Elem
  . forall x:Elem . exists n:Nat . f(n)=x           %(f_surjective)%
  . exists n:Nat . forall m:Nat . def f(m) => m<n   %(f_bounded)%
  reveal Elem
end
```

L. Annex (informative): Use cases

This annex sketches scenarios that outline how DOL is intended to be applied. For each scenario, we list its status of implementation, the DOL features it makes use of, and provide a brief description.

L.1. Generating multilingual labels for menus in a user interface

Status exists (but not yet DOL-based)

Features Aligning (multiple OWL ontologies), Annotation

DO-ROAM (**D**ata and **O**ntology driven **R**oute-finding **Of** **A**ctivity-oriented **M**obility[1]) is a web service with an interactive frontend that extends OpenStreetMap by an ontology-based search for located activities and opening hours [10]. The service is driven by a set of different OWL ontologies that have been aligned to each other using the Falcon matching tool [32]. The user interface of the DO-ROAM web frontend offers multilingual labels, which are maintained in close connection to the underlying ontologies.

Porting DO-ROAM to DOL would enable the coherent representation of the aligned ontologies as one OMS network, and it would enable the maintenance of the user interface labels as annotations inside the ontology.

L.2. Connecting devices of differing complexity in an Ambient Assisted Living setting

Status core ontology (not DOL-based) and service environment exists – the DOL-based extensions not yet

Features Logical OMS mappings across different logics, connection to linked open datasets

Consider the following ambient assisted living (AAL) scenario:

> Clara instructs her **wheelchair** to get her to the **kitchen** (<u>**next door**</u> to the **living room**. For **dinner**, she would like to take a *pizza* from the **freezer** and bake it in the **oven**. (Her diet is *vegetarian*.) <u>**Afterwards**</u> she needs to rest in **bed**.

Existing ontologies for ambient assisted living (e.g. the OpenAAL[2] OWL ontology) cover the *core* of these concepts; they provide at least classes (or generic superclasses) corresponding to the concepts highlighted in **bold**. However, that does not cover the scenario completely:

[1] `http://www.do-roam.org`
[2] `http://openaal.org`

- Some concepts (here: food and its properties, *italicized*) are not covered. There are separate ontologies for that (such as the Pizza ontology[3]), whereas information about concrete products (here: information about the concrete pizza in Clara's oven) would rather come from Linked Open Datasets than from formal ontologies.
- Not all concepts (here: space and time, underlined) are covered at the required level of complexity. OpenAAL says that appointments have a date and that rooms can be connected to each other, but not what exactly that means. Foundational ontologies and spatial calculi, often formalized in first-order logic, cover space and time at the level of complexity required by a central controller of an apartment and by an autonomously navigating wheelchair.
- Thirdly, even description logic might be too complex for very simple devices involved into the scenario, such as the kitchen light switch, for which propositional logic may be sufficient.

Thus, an adequate formalization of this scenario has to be heterogeneous. For example, one could imagine the following axioms:

light switch "light is switched on if and only if someone is in the room and it is dark outside" – this could be formalized in propositional logic as $\mathrm{light\_on} \equiv \mathrm{person\_in\_room} \wedge \mathrm{dark\_outside}$.

freezer "a vegetarian pizza is a pizza whose toppings are all vegetarian" – this could be formalized in description logic as $\mathrm{VegetarianPizza} \equiv \mathrm{Pizza} \sqcap \forall \mathrm{hasTopping.Vegetarian}$

wheelchair "two areas in a house (e.g. a working area in a room) are either the same, or intersecting, or bordering, or separated, or one is part of the other" – this could be formalized as an RCC-style spatial calculus in first-order logic as

$$\begin{aligned}\forall a_1, a_2. \quad & \mathrm{equal}(a_1, a_2) \veebar \mathrm{overlapping}(a_1, a_2) \veebar \mathrm{bordering}(a_1, a_2) \veebar \mathrm{disconnected}(a_1, a_2) \\ & \veebar \mathrm{part\_of}(a_1, a_2) \veebar \mathrm{part\_of}(a_2, a_1).\end{aligned}$$

DOL would be capable of expressing all that within one library of heterogeneous ontologies arranged around an OWL core (here: the OpenAAL ontology), including OMS mappings from OpenAAL to the other ontologies, as well as a re-declaration of a concrete pizza product from a product dataset as an instance of the Pizza OWL class.

L.3. Interpreting the OWL formalization of the DOLCE foundational ontology in First-order logic

Status potential use case

Features Logical OMS mappings

DOLCE is a foundational ontology that has primarily been formalized in the first-order logic ontology language KIF (a predecessor of Common Logic), but also in OWL ("DOLCE Lite") [47]. This 'OWLized' version was targeting use in semantic web services and domain ontology interoperability, and to provide the generic categories and relationships to aid domain ontology development. DOLCE has been used also for semantic middleware, and in OWL-formalized

[3]This is not a fully comprehensive food ontology, but rather a well-known sample OWL ontology; cf. `http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/`

ontologies of neuroimaging, computing, ecology, and data mining and optimization. Given the differences in expressivity, DOLCE Lite had to simplify certain notions. For example, the DOLCE Lite formalization of "temporary parthood" (something is part of something else at a certain point or interval in time) omits any information about the time, as OWL only supports binary predicates (a.k.a. "properties"). That leaves ambiguities for modeling a view from DOLCE Lite to the first-order DOLCE, as such a view would have to reintroduce the third (temporal) component of such predicates:

- Should a relation asserted in terms of DOLCE Lite be assumed to hold for *all* possible points/intervals in time, i.e. should it be universally quantified?
- Or should such a relation be assumed to hold for *some* points/intervals in time, i.e. should it be existentially quantified?
- Or should a concrete value for the temporal component be assumed, e.g. "0" or "now"?

DOL would support the formalization of all of these views and, given suitable consistency checking tools, the analysis of whether any such view would satisfy all further axioms that the first-order DOLCE states about temporal parthood.

L.4. Extending the OWL Time ontology to a more comprehensive coverage of time

Status potential use case

Features Logical OMS mappings

The OWL Time ontology[4] covers temporal concepts such as instants and intervals and has been designed for describing the temporal content of Web pages and the temporal properties of Web services. While OWL is suitable for these intended applications, only a first-order axiomatization is capable of faithfully capturing all relevant notions, such as the trichotomy of the "before" relation: One instant is either before another one, or at the same time, or after. Moreover, a relationship between facts expressed in terms of instants and facts expressed in terms of intervals (both of which is, independently, possible in OWL), can only be established via first-order logic, e.g. by declaring an interval of length zero equivalent to an instant.

A separate first-order axiomatization of OWL Time exists [[30],[55]]. DOL would instead provide the mechanism of modeling OWL Time as one coherent heterogeneous ontology, using OWL and, e.g., Common Logic.[5] For the temporal description logic $\mathcal{DLR}_{\mathcal{US}}$ for knowledge bases and logic-based temporal conceptual data modeling [[2],[3]]; $\mathcal{DLR}_{\mathcal{US}}$ combines the propositional temporal logic with the *Since* and *Until* operators and the (non-temporal) description logic $\mathcal{DLR}$ and can be regarded as an expressive fragment of the first-order temporal logic $L^{since,until}$. Within DOL, this would enable one to have 'lightweight' time aspects with OWL Time, which are then properly formalized with $\mathcal{DLR}_{\mathcal{US}}$ or a leaner variant TDL-Lite [[5]], where notions such as (some time) "before" are given a formal semantics of the intended meaning that the plain OWL Times human-readable object property does not have. The latter, then, would enable the modeler to represent the meaning—hence, restrict the possible models—and check the consistency of the temporal constraints and so-called 'evolution constraints' in the ontology (evolution constraints constrain membership of an object or an individual relation to a concept or relationship over time). For instance, that each divorcee

[4] `http://www.w3.org/TR/2006/WD-owl-time-20060927/`

[5] This is also a use case for multiple namespaces: OWL supports namespaces, CL does not.

must have been a participant in a marriage before, that boarding only may occur after checking in, and that any employee must obtain a salary increase after two years of employment. It also can be used to differentiate between essential and immutable parthood, therewith being precise in the ontology about, e.g., the distinction how a human brain is part of a human (humans cannot live without it), versus how a hand is part of a human (humans can live without it), versus how the hand is part of, say, a boxer, which is essential to the boxer but only for has long as he is a boxer [[4]].

L.5. Metadata in COLORE (Common Logic Repository)

Status exists (but not yet DOL-based)

Features Annotation, Metadata vocabularies

COLORE, the Common Logic Repository[6] is an open repository of more than 150 ontologies as of December 2011, all formalized in Common Logic. COLORE stores metadata about its ontologies, which are represented using a custom XML schema that covers the following aspects[7], without specifying a formal semantics for them:

module provenance author, date, version, description, keyword, parent ontology[8]

axiom source provenance name, author, year[9]

direct relations maps (signature morphisms), definitional extension, conservative extension, inconsistency between ontologies, imports, relative interpretation, faithful interpretation, definable equivalence

DOL provides built-in support for a subset of the "direct relations" and specifies a formal semantics for them. In addition, it supports the implementation of the remainder of the COLORE metadata vocabulary as an ontology, reusing suitable existing metadata vocabularies such as OMV, and it supports the implementation of one or multiple Common Logic ontologies plus their annotations as one coherent library.

[6] `http://stl.mie.utoronto.ca/colore/`

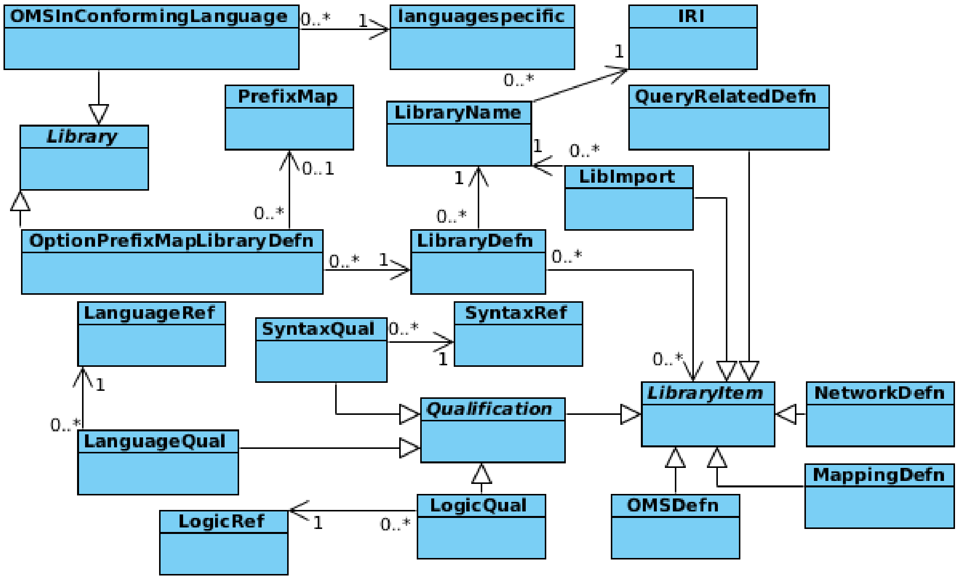
[7] `http://stl.mie.utoronto.ca/colore/metadata.html`

[8] Note that this use of the term "module" in COLORE corresponds to the term structured OMS in this OMG Specification

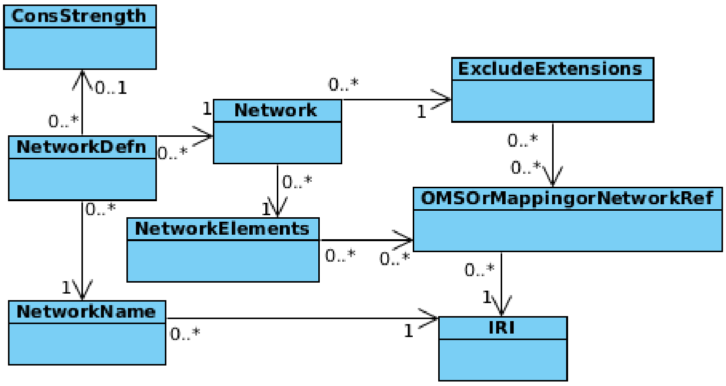
[9] Note that this may cover any sentences in the sense of this OMG Specification

M. Annex (informative): Abstract syntax specified as an SMOF meta model

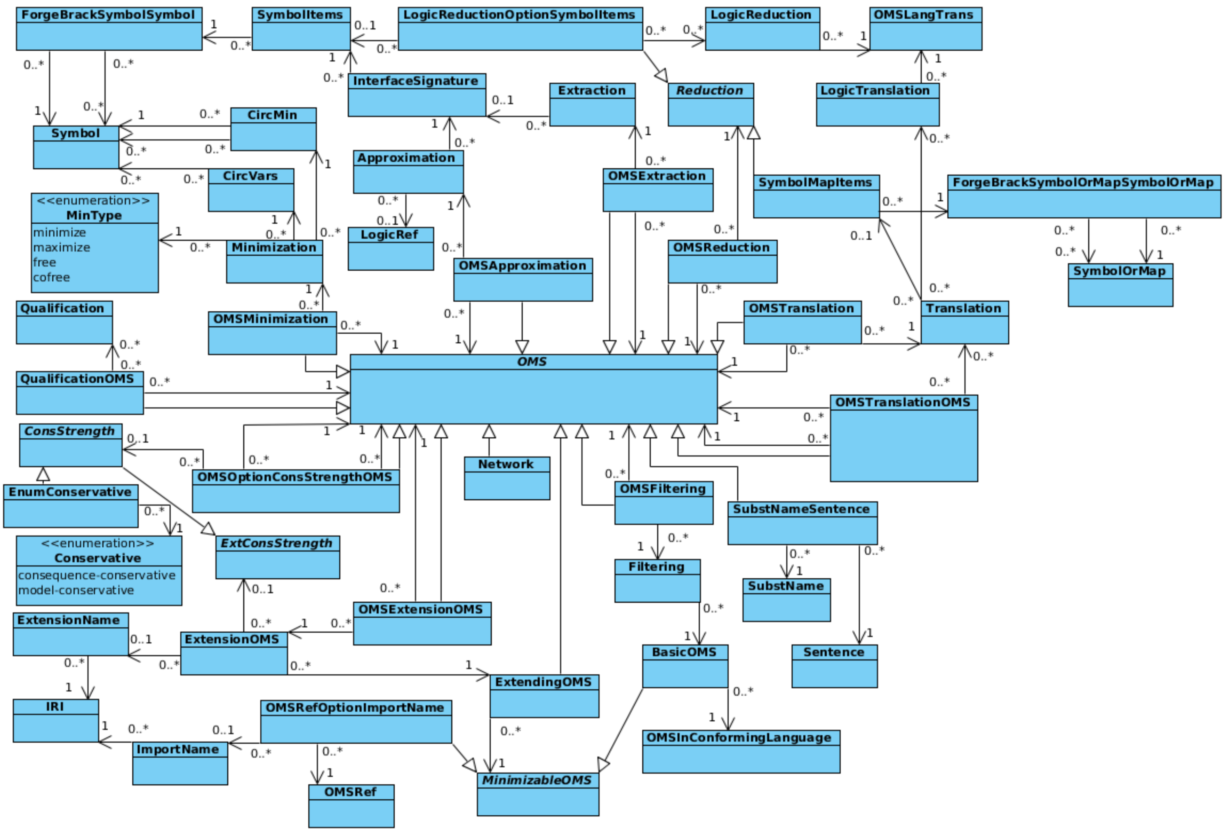
M.1. Libraries



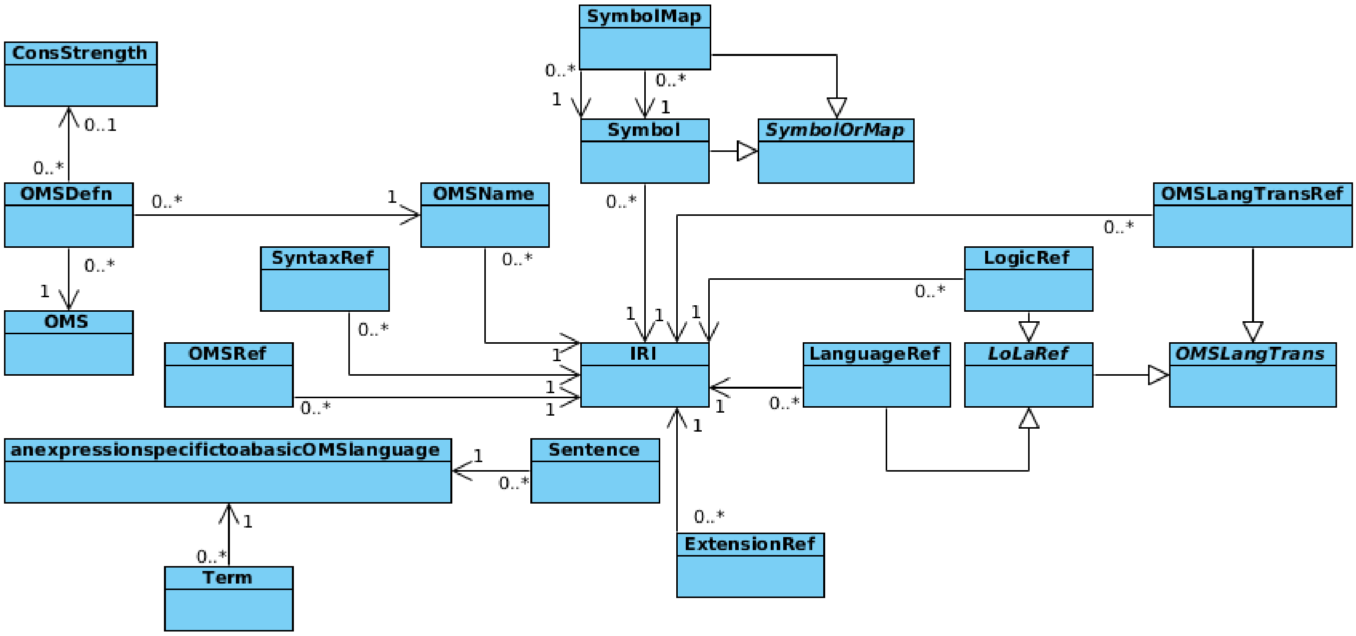
M.2. Networks



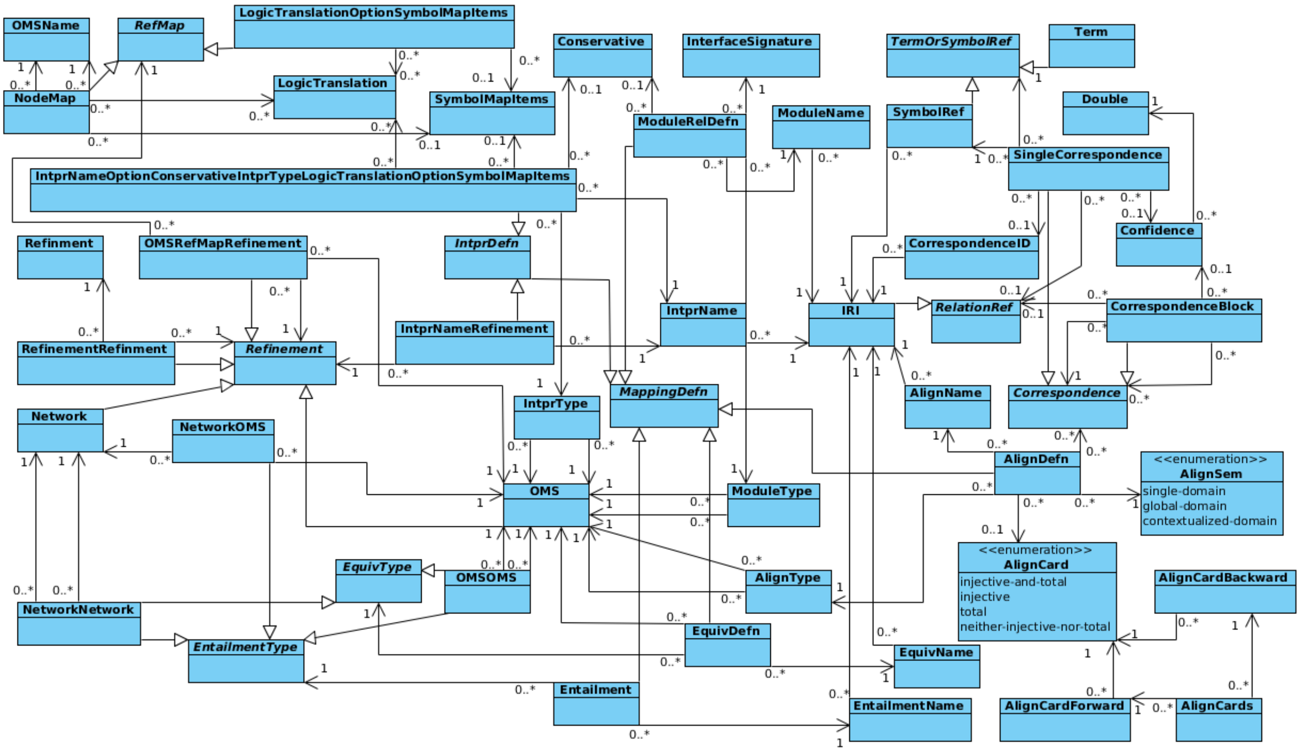
M.3. OMS



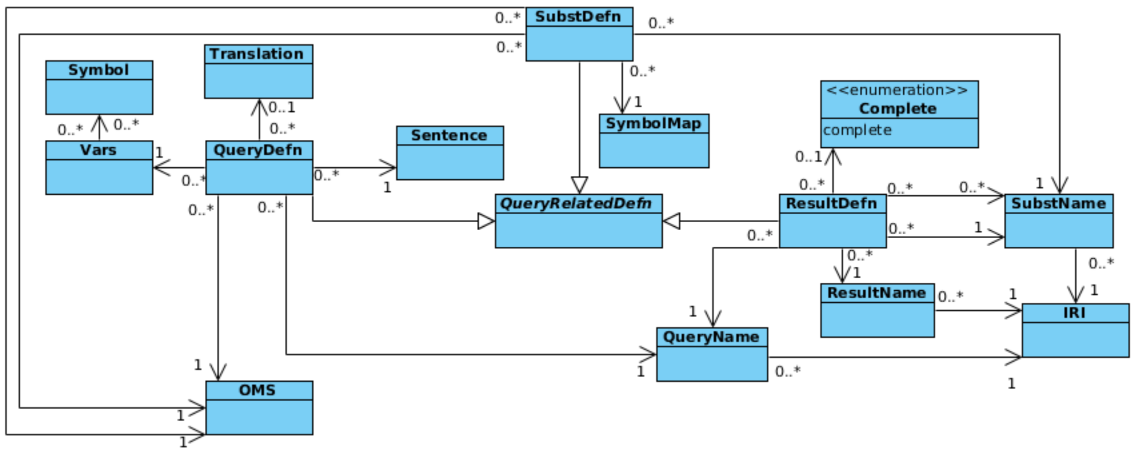
M.4. OMS Definitions



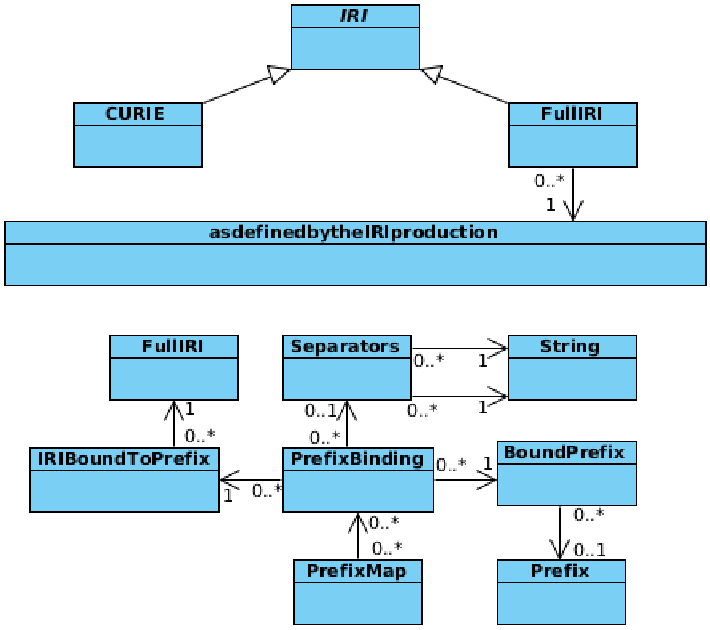
M.5. OMS Mappings



M.6. Queries



M.7. IRIs and Prefixes



N. Annex (informative): Tools for DOL

N.1. The Heterogeneous Tool Set (Hets)

The Heterogeneous Tool Set (Hets) is the reference implementation of DOL. Hets a parsing, analysis and proof tool for OMS, OMS networks and OMS mappings written in DOL and DOL-conforming languages. It supports a wide range of OMS languages and language translations, in particular OWL, RDF, Common Logic, first-order logic and CASL. Support for MOF, UML class diagrams and state machines is in preparation. Hets has been co-developed together with the DOL language presented in this standard, and has been used to test the examples. Hets has been connected to considerable number of proof tools like theorem provers, supporting various logics. Logics that are not directly supported by any proof tool can be supported indirectly, through a logic mapping into a tool-supported logic. [1]

Hets is open source, licensed under GPLv2 or higher. The sources are available at the following URL `https://github.com/spechub/hets`.

N.2. Ontohub, Modelhub, Spechub

Ontohub/Modelhub/Spechub is the second reference implementation of DOL. It is a repository engine for managing OMS, OMS networks and OMS mappings written in DOL and DOL-conforming languages. It supports the same range of OMS languages and language translations as Hets (indeed, Hets is used for analyzing DOL files). The novel aspect w.r.t. Hets is the provision of git-based repositories and IRIs for libraries, OMS, symbols and mappings (see also Annex O).

Users of Ontohub/Modelhub/Spechub can upload, browse, search and annotate OMS in various languages via a web frontend, see `https://ontohub.org`, `https://model-hub.org` and `https://spechub.org`. Ontohub/Modelhub/Spechub is open source under GNU AGPL 3.0 license, the sources are available at the following URL `https://github.com/ontohub/ontohub`.

Ontohub/Modelhub/Spechub enjoys the following distinctive features:

- OMS can be organized in multiple repositories, each with its own management of editing and ownership rights,
- private repositories are possible,
- version control of OMS is supported via interfacing the Git version control system,
- OMS can be edited both via the browser and locally with any editor (and in the latter case pushed via Git); Git will synchronize both editing approaches,

[1] While the Hets parser should support the current version of DOL as presented in this standard, it can happen that the most recent changes to the DOL syntax are not fully supported by the Hets static analysis and proof support yet. This will be fixed in the future.

- one and the same URL is used for referencing an OMS, downloading it (for use with tools), and for user-friendly presentation in the browser (i.e. Ontohub/Modelhub/Spechub is fully linked-data compliant)
- modular and heterogeneous OMS are specially supported,
- OMS can not only be aligned (as in BioPortal and NeOn), but also be combined along alignments (using DOL's `combine` construct),
- logical relations between OMS (interpretation of theories, conservative extensions etc.) are supported,
- support for a variety of OMS languages,
- OMS can be translated to other OMS languages, and compared with OMS in other languages,
- heterogeneous OMS involving several languages can be built,
- OMS languages and OMS language translations are first-class citizens and are available as linked data.

Ontohub/Modelhub/Spechub is not a repository, but a semantic repository engine. This means that Ontohub/Modelhub/Spechub OMS are organized into repositories. The organization into repositories has several advantages:

- Firstly, repositories provide a certain structuring of OMS, let it be thematically or organizational. Access rights can be given to users or teams of users per repository. Typically, read access is given to everyone, and write access only to a restricted set of users and teams. However, also completely open, i.e. world-writeable repositories are possible, as well as private repositories visible only to a restricted set of users and teams. Since creation of repositories is done easily with a few clicks, this supports a policy of many but small repositories (which of course does not preclude the existence of very large repositories). Note that also structuring within repositories is possible, since each repository is a complete file system tree.
- Secondly, repositories are git repositories. Git is a popular decentralized version control system. With any git client, the user can clone a repository to her local hard disk, edit it with any editor, and push the changes back to Ontohub/Modelhub/Spechub. Alternatively, the web frontend can be used directly to edit OMS; pushing will then be done automatically in the background. Parallel edits of the same file are synchronized and merged via git; handling of merge conflicts can be done with git merge tools.
- Thirdly, OMS can be searched globally in Ontohub/Modelhub/Spechub, or in specific repositories. Additionally, user-supplied metadata like categories, formality levels and purposes can be used for searching.

Ontohub/Modelhub/Spechub is linked-data compliant. This means that OMS are referenced by a unique URL of the form `https://ontohub.org/name-of-repository/path-within-repository`. Depending on the MIME type of the request, under this URL, the raw OMS file will be available, but also a HTML version for display in a browser, an XML and a JSON version for processing with tools.

N.3. APIs

Both Hets and Ontohub/Modelhub/Specshub provide APIs for the interchange with other tools[2]. Ontohub/Modelhub/Specshub also provides an API for exchange with other instances, so that e.g. Ontohub and Modelhub can exchange information about available repositories and their OMS.

In the future, these APIs shall be aligned with OMG's standardization effort API4KB.

[2]See `https://github.com/spechub/Hets/wiki/RESTful-Interface` and `https://github.com/ontohub/ontohub/wiki/`.

O. Annex (informative): Ontohub loc/id v2

This annex describes the way how Ontohub assigns IRIs to DOL libraries, OMS, symbols etc. Ontohub[1] is a reference implementation for DOL, and it is suggested that other tools supporting DOL should adopt the same or a similar scheme for IRIs.

O.1. Concept

Generally an Ontohub loc/id (locator/identifier) is just an IRI of a library, an OMS or one of its members (symbols, sentences, mappings). However Ontohub loc/ids are generated by the Ontohub application and assigned to an OMS. We try to infer them from the path of the repository, the path of the OMS and the specific name. Additionally we ensure that this specific IRI is actually a locator and not *just* an identifier.

This is quite important as the IRI of an OMS is the general starting interface a user has with the given OMS. When she evaluates the OMS in her tool of choice she'll use the IRI to reference the given OMS. When she wants to work on ontohub with the given OMS she'll point her browser at the given IRI. As one familiarity with the Ontohub application increases one will more often want to use the IRI instead of just searching or even browsing for something. This is further intensified if the IRI-schema follows a schema that is easily understood by a user.

O.2. Ontohub-Style

Identifying OMS and their members in Ontohub is a hierarchical task. An OMS library belongs to a repository. An OMS may belong directly to a repository, or indirectly through a library. Mappings, symbols and sentences in turn belong to an OMS. So we could use the hierarchical portion of an IRI instead of the query string. This would mean using a forward slash (/) as separator.

Ontohub loc/ids are specific to an instance of the Ontohub application. However such an instance might be reachable via multiple multiple FQDNs (fully qualified domain name) and ports. So instead we should expect a *qualified loc/id* to be a tuple consisting of the specific application instance, represented by the set of their schema-fqdn-port tuples, and the actual identifying portion beginning with the hierarchical forward slash (/).

O.2.1. qualified loc/id structure

1. Set of Schema + FQDNs + Port for an instance: *INSTANCE*, e.g. { `http://ontohub.org`, `http://model-hub.org`, `http://spechub.org` }
2. Identifying portion loc/id with leading forward slash (/)
    - The identifying portion is split into three parts.

[1] In this annex, "Ontohub" could equally well be substituted by "Modelhub" and "Spechub".

- *HIERARCHY*: is the `path/to/OMS-file`, with elements split by a forward slash (/).
- *MEMBER*: is the element of the OMS at the specific position. It is being separated from the *HIERARCHY* by two forward slashes (//). These forward slashes are also being used to separate members inside of *MEMBER* (e.g. in the case of an OMS which contains a symbol).
- *COMMAND*: is not really an element or part of an OMS, but a command the user wishes to execute on the object selected by the previous sections of the loc/id. It is denoted and separated from the rest of the IRI by the use of three consecutive forward slashes (///).

O.2.2. Examples

OMS library

OMS library	`/dol-testing/double_mapped_blendoid`
OMS	`/dol-testing/double_mapped_blendoid//DMB-CommonSource`
Mapping	`/dol-testing/double_mapped_blendoid//SomeMapping`
Symbol	`/dol-testing/double_mapped_blendoid//DMB-CommonSource/ /KitchenTable`
Sentence	`/dol-testing/double_mapped_blendoid//DMB-CommonSource/ /Ax02`

OMS

OMS library	`/dol-testing/double_mapped_blendoid`
OMS	`/default/pizza`
Mapping	`/default/pizza//SomeMapping`
Symbol	`/default/pizza//Veneziana`
Sentence	`/default/pizza//Ax02`

Fully qualified symbols (e.g. $+ : Nat \times Nat \mapsto Nat$) will need to be escaped but will be supported.

O.3. Specification

We can specify qualified loc/id IRIs as a special case of RFC 3987 (IRI, [18]). Code-excerpt O.1 on page 154 contains this specification of qualified loc/ids in Augmented Backus-Naur Form (ABNF, [15]). We use ABNF here, because RFC 3987 itself specifies IRIs using ABNF and we wanted to be able to reference rules from the RFC in our specification. Such rules can be easily identified by the `i`-prefix that was used when writing the IRI-rules.

`<Loc-Id-IRI>` represents the start rule for a qualified loc/id and `<Loc-Id>` would be the starting non-terminal for a loc/id without its *INSTANCE* qualifier. The following symbols are non-terminal symbols that represent rules from the IRI-RFC.

- `<iquery>`
- `<ifragment>`
- `<scheme>`
- `<iauthority>`
- `<isegment-nz>`

One should take note that the `<scheme>` rule does not include a `i`-prefix. This is because `<scheme>` is actually taken from RFC 3986[7], which defines the URI.

```
; Author: Tim Reddehase
; E-Mail: robustus AT rightsrestricted DOT com
; Last-Changed: 2015-02-22
; Version: 0.1.2
;
; This ABNF for Loc/Ids is based on the definition
; of IRIs and as such uses Rules from the RFC-Definition
; of IRIs: http://tools.ietf.org/html/rfc3987#section-2.2
; Rules that represent an IRI-rule usually start with an
; i char.

Loc-Id-IRI = li-instance [ li-ref ] Loc-Id [ "?" iquery ] [ "#"
    ifragment ]

; Represents an Ontohub-Application instance.
; Semantically multiple <li-instance> values
; can be equivalent and thus forming the
; set of INSTANCE. <scheme> is a rule inside
; of the IRI RFC.
li-instance = scheme "://" iauthority

; a lone repository is also a Loc/Id
Loc-Id = "/" li-repository [ li-hierarchy [ li-member ] ] [
    li-command ]

; Represents the path/directory name of the repository
li-repository = isegment-nz

; Represents a ref/ special form
li-ref = "/" "ref/" isegment-nz

; Represents the path inside the Repository to the ontology
li-hierarchy = *( "/" isegment-nz )

; Represents internal 'path' inside of the ontology
; where child-ontologies, mappings, symbols and sentences
; are first-class members.
li-member = *2( "//" isegment-nz )

; Represents a command to be 'executed' on the
; specific resource
li-command = *( "///" isegment-nz )
```

Figure O.1.: Specification of loc/id IRIs in ABNF

O.4. ref/ special form loc/ids

There is one additional syntax-element that we haven't covered yet. One of the main features that Ontohub provides in its role as an *Open OMS Repository* is versioning of OMS by backing the repositories with git. It is quite important that we can access such versions and other related files inside of a repository, which can be basically viewed as a directory in a file system. `ref/`-style IRIs accomplish this task.

The `ref/argument`-form is a prefix of the *HIERARCHY*, *MEMBER* and *COMMAND* components – otherwise referred to as unqualified loc/id, or in short: loc/id.

- Version: `/ref/2/default/pizza//SomeMapping`
- Commit: `/ref/def3ab/default/pizza//SomeMapping`
- Branch: `/ref/master/default/pizza//SomeMapping`
- Date: `/ref/2014-09-07/default/pizza//SomeMapping`
  - would take the latest commit which applies to the Date range.
- MMT: `/ref/mmt/default/pizza?SomeMapping`
  - Does not refer to a specifically designated version of the element, but always refers to the current one instead. This version allows to use MMT-style IRIs [56], which should guarantee basic support for tools which expect the MMT-style.

O.4.1. References inside of the tree

Additionally we need to provide a way to reference files inside a repository, This especially applies to files that do not represent OMS. This will be accomplished by the `tree/` special form. Additionally we will support a `treeref` special form which allows to reference a specific version of a files using the *Commit*, *Branch* and *Date* references. MMT is for obvious reasons not supported.

- File: `/tree/default/some_directory/some_child_dir/Foo.txt`
  - applies to HEAD commit of main branch (currently always *master*)
- File at reference: `/treeref/{REF}/default/tree/some_directory/some_child_dir/Foo.txt`
  - where {REF} is any of the above possible ref-types: Commit, Branch or Date

O.5. Disambiguating

If the `path/to/an-OMS` can actually also be a path to a directory – which would be possible if there were a directory named **pizza** and an ontology named **pizza.owl** – will the loc/id be resolved to a disambiguating page.

This page will contain a link to the tree for the directory, e.g. `/tree/default/pizza`, and a link to a `ref/` special form version of the OMS, e.g. `/ref/master/default/pizza`.

If however the loc/id is requested with a *text/plain* content type we will always serve the OMS. This is in part because there is no reasonable representation of a directory that we would want to support. Another reason is that Ontohub serves OMS as its main objects. And as *text/plain* is the MIME-type that was chosen to always return the textual content of an OMS (the raw file), we will need to serve that, even if the loc/id would be ambiguous in a normal request.

References

[1] J. Adámek, H. Herrlich, and G. Strecker. *Abstract and Concrete Categories.* Wiley, New York, 1990.

[2] A. Artale, E. Franconi, F. Wolter, and M. Zakharyaschev. A temporal description logic for reasoning about conceptual schemas and queries. *In: S. Flesca and S. Greco and N. Leone and G. Ianni, Proceedings of the 8th Joint European Conference on Logics in Artificial Intelligence (JELIA-02), Springer LNAI 2424, 98-110, 2002*, 2002.

[3] A. Artale, C. Parent, and S. Spaccapietra. Evolving objects in temporal information systems. *Annals of Mathematics and Artificial Intelligence 50:5-38, 2007*, 2002.

[4] Alessandro Artale, Nicola Guarino, and C. Maria Keet. Formalising temporal constraints on part-whole relations. *In: Gerhard Brewka and Jerome Lang, 11th International Conference on Principles of Knowledge Representation and Reasoning (KR'08), 673-683, AAAI Press 2008*, 2008.

[5] Alessandro Artale, Roman Kontchakov, Carsten Lutz, Frank Wolter, and Michael Zakharyaschev. Temporalising tractable description logic. *Proc. of the 14th International Symposium on Temporal Representation and Reasoning (TIME-07), 2007*, 2007.

[6] Kenneth J. Bagstad, Ferdinando Villa, Gary W. Johnson, and B. Voigt. ARIES – artificial intelligence for ecosystem services: A guide to models and data , version 1.0. Technical Report 1, ARIES, 2011. `http://ariesonline.org/docs/ARIESModelingGuide1.0.pdf`.

[7] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (INTERNET STANDARD), January 2005. Updated by RFCs 6874, 7320.

[8] C. Bizer and T. Heath. Linked data—the story so far. *International journal on semantic web and information systems*, 5(3):1–22, 2009.

[9] W.A. Carnielli, M. Coniglio, D.M. Gabbay, P. Gouveia, and C. Sernadas. Analysis and synthesis of logics: how to cut and paste reasoning systems. *Applied logic series, Springer, 2008.*, 2008.

[10] M. Codescu, G. Horsinka, Oliver Kutz, T. Mossakowski, and R. Rau. Do-roam: Activity-oriented search and navigation with openstreetmap. *In: Fourth International Conference on GeoSpatial Semantics, C. Claramunt, S. Levashkin, M. Bertolotto (eds.), pp. 88–107, no. 6631 in LNCS, Springer, 2011.*, 2011.

[11] M. Codescu and T. Mossakowski. Heterogeneous colimits. In F. Boulanger, C. Gaston, and P.-Y. Schobbens, editors, *MoVaH'08 Workshop on Modeling, Validation and Heterogeneity.* IEEE press, 2008.

[12] CoFI (The Common Framework Initiative). CASL *Reference Manual.* Lect. Notes Comp. Sci.2960 (IFIP Series). Springer, 2004.

[13] European Comission. INSPIRE Geoportal: Enhancing access to European spatial data, 2014. `http://inspire-geoportal.ec.europa.eu/`.

[14] Information technology — Common Logic (CL): a framework for a family of logic-based languages. Technical Report 24707:2007, ISO/IEC, 2007. `http://iso-commonlogic.org`.

[15] D. Crocker and P. Overell. Augmented BNF for Syntax Specifications: ABNF. RFC 5234 (INTERNET STANDARD), January 2008. Updated by RFC 7405.

[16] Jérôme David, Jérôme Euzenat, François Scharffe, and Cássia Trojahn dos Santos. The alignment API 4.0. *Semantic Web*, 2(1):3–10, 2011.

[17] Răzvan Diaconescu, Joseph Goguen, and Petros Stefaneas. Logical support for modularisation. In Gerard Huet and Gordon Plotkin, editors, *Proceedings of a Workshop on Logical Frameworks*, 1991.

[18] M. Duerst and M. Suignard. Internationalized Resource Identifiers (IRIs). RFC 3987 (Proposed Standard), January 2005.

[19] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching, Second Edition.* Springer, 2013.

[20] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585.

[21] J. A. Goguen and R.M. Burstall. A study in the foundations of programming methodology: Specifications, institutions, charters and parchments. *In: Category Theory and Computer Programming, D. Pitt et al. (eds.), pp. 313–333, no. 240 in LNCS, Springer, 1985.*, 1985.

[22] Joseph A. Goguen and Rod M. Burstall. Institutions: Abstract model theory for specification and programming. *J. ACM*, 39:95–146, 1992.

[23] Joseph A. Goguen and Grigore Rosu. Composing hidden information modules over inclusive institutions. In Olaf Owe, Stein Krogdahl, and Tom Lyche, editors, *From Object-Orientation to Formal Methods, Essays in Memory of Ole-Johan Dahl*, volume 2635 of *Lecture Notes in Computer Science*, pages 96–123. Springer, 2004.

[24] B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Just the right amount: Extracting modules from ontologies. In *Proc. 16th Int. World Wide Web Conf. (WWW)*, pages 717–726, 2007.

[25] Object Management Group. Ontology, model and specification integration and interoperability request for proposal. *OMG Document Number ad/13-12-02*, 2013.

[26] W3C OWL Working Group. Owl 2 web ontology language manchester syntax. *W3C Working Group Note, 27 October 2009.* *`http://www.w3.org/TR/2009/NOTE-owl2-manchester-syntax-20091027/`*, 2009.

[27] W3C OWL Working Group. Owl 2 web ontology language: Rdf-based semantics. *W3C Recommendation, 27 October 2009.* *`http://www.w3.org/TR/2009/REC-owl2-rdf-based-semantics-20091027/`*, 2009.

[28] W3C RDF Core Working Group. RDF 1.1 XML syntax. *W3C Recommendation, 25 February 2014.* *`http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/`*, 2014.

[29] W3C Semantic Web Deployment Working Group. Best practice recipes for publishing rdf vocabularies. *W3C Working Group Note, 28 August 2008.* `http://www.w3.org/TR/2008/NOTE-swbp-vocab-pub-20080828/`, 2008.

[30] J.R. Hobbs and F. Pan. An ontology of time for the semantic web. *ACM Transactions on Asian Language Processing (TALIP): Special issue on Temporal Information Processing, Vol. 3, No. 1, March 2004, pp. 66–85.*, 2004.

[31] I. Horrocks, Oliver Kutz, and U. Sattler. The even more irresistible sroiq. *In Knowledge representation 2006, p. 57–67*, 2006.

[32] W. Hu and Y. Qu. Falcon-ao: A practical ontology matching system. *In: Proceedings of WWW-07, pp. 237–239, 2008.*, 2008.

[33] Yazmin Angelica Ibanez, Till Mossakowski, Andrzej Tarlecki, and Don Sannella. Notions of modularity for ontologies and specifications. In Narciso Marti-Oliet et al., editor, *Logic, Rewriting, and Concurrency: A Festschrift Symposium in Honor of José Meseguer*, Lecture Notes in Computer Science. Springer, 2015. To appear.

[34] Z. Khan and C.M. Keet. Addressing issues in foundational ontology mediation. In J. Filipe and J. Dietz, editors, *5th International Conference on Knowledge Engineering and Ontology Development (KEOD'13), Vilamoura, Portugal, 19-22 September*, pages 5–16. SCITEPRESS, 2013.

[35] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages,. *Journal of the ACM*, 42:741–843, 1995.

[36] Alexander Knapp, Till Mossakowski, and Markus Roggenbach. An Institutional Framework for Heterogeneous Formal Development in UML, 2014. CoRR abs/1403.7747.

[37] Alexander Knapp and Jochen Wuttke. Model checking of UML 2.0 interactions. In Thomas Kühne, editor, *Reports Rev. Sel. Papers Wsh.s Symp.s MoDELS 2006*, number 4364 in Lect. Notes Comp. Sci., pages 42–51. Springer, 2007.

[38] Roman Kontchakov, Frank Wolter, and Michael Zakharyaschev. Logic-based ontology comparison and module extraction, with an application to DL-Lite. *Artif. Intell.*, 174(15):1093–1141, 2010.

[39] Oliver Kutz, I. Normann, Till Mossakowski, and D. Walther. Chinese whispers and connected alignments. In *Proc. of the 5th Int. Workshop on Ontology Matching (OM 2010)*, 2010. `http://disi.unitn.it/~p2p/OM-2010/om2010_Tpaper3.pdf`.

[40] Christoph Lange, Till Mossakowski, and Oliver Kutz. LoLa: A modular ontology of logics, languages, and translations. In Thomas Schneider and Dirk Walther, editors, *Modular Ontologies*, number 875 in CEUR Workshop Proceedings, Aachen, 2012.

[41] Berners-T. Lee. Design issues: Linked data. *27 July 2006.* `http://www.w3.org/DesignIssues/LinkedData.html`, 2006.

[42] V. Lifschitz. Circumscription. In *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3*, pages 297–352. Oxford University Press, 1994.

[43] D. Lucanu, Y.-F. Li, and J. Song Dong. Semantic web languages — towards an institutional perspective. *In: Essays Dedicated to Joseph A. Goguen, Lecture Notes in Computer Science 4060 Springer 2006, p. 99–123*, 2006.

[44] Carsten Lutz, Dirk Walther, and Frank Wolter. Conservative extensions in expressive description logics. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 453–458, 2007.

[45] Carsten Lutz and Frank Wolter. Foundations for uniform interpolation and forgetting in expressive description logics. In Toby Walsh, editor, *IJCAI*, pages 989–995. IJCAI/AAAI, 2011.

[46] S. Mac Lane. *Categories for the Working Mathematician*. Springer, 1998. Second edition.

[47] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. Ontology library. *WonderWeb Deliverable, Report No. 18 of the Laboratory for Applied Ontology – ISTC-CNR, Dec. 2003.* `http://www.loa-cnr.it/Papers/D18.pdf`, 2003.

[48] J. McCarthy. Circumscription — a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[49] José Meseguer. General logics. In *Logic Colloquium 87*, pages 275–329, 1989.

[50] T. Mossakowski and Oliver Kutz. The onto-logical translation graph. *In: 5th International Workshop on Modular Ontologies (WoMO) at the 23rd European Summer School in Logic, Language and Information (ESSLLI) (Ljubljana, Slovenia, Aug. 8–12, 2011), Oliver Kutz, T. Schneider (eds.), IOS Press 2011.*, 2011.

[51] Till Mossakowski. Relating CASL with Other Specification Languages: the Institution Level. *Theoretical Computer Science*, 286:367–475, 2002.

[52] Till Mossakowski, Mihai Codescu, Oliver Kutz, Christoph Lange, and Michael Gruninger. Proof support for common logic. In *Proc. Wsh. Automated Reasoning for Quantified Non-Classical Logic (ARQNL)*, 2014.

[53] Till Mossakowski, Ulf Krumnack, and Tom Maibaum. What is a derived signature morphism? In R. Diaconescu, M. Codescu, and I. Tutu, editors, *WADT 2014*, Lecture Notes in Computer Science. Springer, 2015. To appear.

[54] Till Mossakowski and Andrzej Tarlecki. Heterogeneous logical environments for distributed specifications. In Andrea Corradini and Ugo Montanari, editors, *WADT 2008*, number 5486 in Lecture Notes in Computer Science, pages 266–289. Springer, 2009.

[55] Feng Pan and Jerry R. Hobbs. Time in OWL-S. In *Proc. of the AAAI Spring Symp. on Semantic Web Services*, pages 29–36, 2004.

[56] Florian Rabe and Michael Kohlhase. A scalable module system. *Information & Computation*, pages 1–95, 2013.

[57] D. Sannella and A. Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2012.

[58] Michael Schneider, Sebastian Rudolph, and Geoff Sutcliffe. Modeling in OWL 2 without restrictions. In Mariano Rodriguez-Muro, Simon Jupp, and Kavitha Srinivas, editors, *Proceedings of the 10th International Workshop on OWL: Experiences and Directions (OWLED 2013) co-located with 10th Extended Semantic Web Conference (ESWC 2013), Montpellier, France, May 26-27, 2013.*, volume 1080 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[59] Ed Seidewitz. Model Semantics and Mathematical Logic. *Syntax And Semantics*, pages 1–13, August 2008.

[60] Heiner Stuckenschmidt, Christine Parent, and Stefano Spaccapietra, editors. *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *Lecture Notes in Computer Science*. Springer, 2009.

[61] MC. Suarez-Figueroa and A. Gomez-Perez. First attempt towards a standard glossary of ontology engineering terminology. *In: The 8th International Conference on Terminology and Knowledge Engineering, 18–21 August 2008, Copenhagen, Denmark.*, 2008.

[62] Geoff Sutcliffe. The tptp problem library for automated theorem proving. `http://www.tptp.org`.

[63] Geoff Sutcliffe. The TPTP problem library and associated infrastructure. *J. Autom. Reasoning*, 43(4):337–362, 2009.

[64] Geoff Sutcliffe, Christian B. Suttner, and Theodor Yemenis. The TPTP problem library. In Alan Bundy, editor, *Automated Deduction - CADE-12, 12th International Conference on Automated Deduction, Nancy, France, June 26 - July 1, 1994, Proceedings*, volume 814 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 1994.

[65] ed. Wikimedia Foundation. Linked data. *from Wikipedia, the free encyclopedia, 9 November 2011. http://en.wikipedia.org/w/index.php?title=Linked_Data&oldid=459835053*, 2011.

[66] A. Zimmermann, M. Krötzsch, J. Euzenat, and P. Hitzler. Formalizing ontology alignment and its operations with category theory. *In: B. Bennett and C. Fellbaum: Proceedings of the Fourth International Conference on Formal Ontology in Information Systems (FOIS 2006), pp. 277–288,Frontiers in Artificial Intelligence and Applications 150, IOS Press 2006*, 2006.