An OMG® Distributed Ontology, Modeling, and Specification Language™ Publication

OBJECT MANAGEMENT GROUP®

The Distributed Ontology, Modeling, and Specification Language™

Version 1.0

OMG Document Number: formal/18-09-02

Release Date: October 2018

Standard document URL: https://www.omg.org/spec/DOL/1.0

Normative Machine Consumable Files:

https://www.omg.org/spec/DOL/20151109/DOL-metamodel.xmi

Informative Machine Consumable Files:

https://www.omg.org/spec/DOL/20171106/DOL-terms.rdf

https://www.omg.org/spec/DOL/20171106/DOL-terms-ODM-UML.xml

https://www.omg.org/spec/DOL/20171106/DOL-terms-ODM.xml

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.

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

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
109 Highland Avenue
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.

Issues

The reader is encouraged to report any technical or editing issues/problems with this specification by completing the Issue Reporting Form listed on the main web page http://www.omg.org, under Documents, Report a Bug/Issue.

This page intentionally left blank.

1 Scope

1.1 General

This OMG Specification specifies the Distributed Ontology, Modeling 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 [22].

1.2 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).

1.3 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 written in different languages);

2) mappings between OMS (which map OMS symbols to OMS symbols);

3) OMS networks (involving several OMS and mappings between them);

4) translations between different OMS languages conforming with DOL (translating a whole OMS to another language);

5) structuring constructs for modeling non-monotonic behavior;

6) annotation and documentation of OMS, mappings between OMS, symbols, and sentences;

7) recommendations of vocabularies for annotating and documenting OMS;

8) a syntax for embedding the constructs mentioned under (1)–(6) as annotations into existing OMS;

9) a syntax for expressing (1)–(5) as standoff markup that points into existing OMS;

10) a formal semantics of (1)–(5);

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, Modeling and Specification Language (DOL) by defining an abstract syntax and an associated model-theoretic semantics for DOL.

[1)] Only monotonic logics are within scope of this specification. Conformance criteria for non-monotonic logics are still under development. However, closure (i.e., employing a closed-world assumption) provides non-monotonic reasoning in DOL. It is also possible to include non-monotonic logics by construing entailments between formulas as sentences of the logic (formalized as an institution).

2 Conformance

2.1 General

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 models, TPTP, CASL) as well as translations among these is discussed in informative annexes of this OMG Specification.

2.2 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 related to its own abstract syntax and formal semantics, 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 be compatible with DOL's own structuring mechanisms

— its annotation language aspect must be compatible with DOL's meta-language constructs.

Several conformance levels are defined. They differ 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) **abstract syntax conformance:** its abstract syntax is conformant. This means that a) it is specified as an SMOF compliant meta model or as an EBNF grammar. Moreover, b) an SMOF metaclass or an EBNF non-terminal has to be declared to be a subclass of `NativeDocument`, and optionally another metaclass or non-terminal may be declared to be a subclass of `BasicOMS` (see clause 9.2);

2) **serialization conformance:** it has at least one serialization in the sense of section 2.3;

3) **semantic conformance:** 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

   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 in the sense of Definition 2 in clause 10 , 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.

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

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.

Concerning item 1. in the definition of DOL conformance of OMS languages above, the following levels of conformance of the abstract syntax of an OMS language with DOL are defined, 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 2.4.

2.2.1 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 that the following diagram commutes (i.e., following both possible paths from L_1 to I_2 leads to the same result):

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

Figure 2.1: Language Translation

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 (respectively symbols) expressing the translation, to at least one of them).

2.3 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 does not 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, DOL utilizes the annotation or commenting features that the OMS language supports, in order to place such identifiers inside annotations/ comments. Depending on the nature of a given concrete serialization of the OMS language (be it plain text, some serialization of RDF, XML, or some other structured text format), one 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 it is not possible to inject something into an OMS language fragment, because the OMS language serialization does not enable the addition of suitable comments. In this case the solution is to point into the OMS language fragment 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, several levels of conformance of a serialization are distinguished. They differ with respect to their means of conveniently abbreviating long IRI identifiers.

There are seven levels of conformance of a serialization of an OMS language with DOL.

XMI conformance: An XMI serialization for OMS written in the OMS language has been automatically derived from the SMOF specification of the abstract syntax, using the canonical MOF 2 XMI Mapping.

XML conformance: The given serialization has to be specified as an XML schema that satisfies all of the following conditions:

1) The elements of the schema belong to one or more non-empty XML namespaces.

2) The serialization shall use XML *elements* to represent all structural elements of an OMS.

3) XML elements that represent structural elements of an OMS shall support identifier injection in at least one of the following two ways:

   a) Such elements shall be able to carry annotations that comprise at least an object (the value of the annotation) and a IRI-valued predicate (the type of annotation), where the structural element is the subject. The value of the predicate shall either be full IRI according to **NR11**, or the serialization shall specify a way of interpreting the value of the predicate as a full IRI – for example if it is a relative URI or if an abbreviating notation is used. Analogously, the serialization shall permit the object to be a full IRI or anything that can be interpreted as a full IRI.

   b) 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 such elements.

   This requirement is necessary because either an annotation or an attribute or a child element is used to inject identifiers into elements of the XML serialization; cf. clause 9.9.

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

1) The elements of the vocabulary belong to one or more RDF namespaces identified by absolute URIs.

2) The serialization shall specify ways of giving IRIs or URIs to all structural elements of an OMS. (The rationale is that RDF syntax supports the identification of 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.)

3) 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.

See annex C for an example.

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 **NR9**, section 4.1.3.

— The serialization shall provide a designated comment construct that can be placed sufficiently flexibly 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.

Standoff markup conformance: The given serialization has to satisfy at least one of the following conditions:

1) The serialization conforms with the requirements for the *text/plain* media type specified in **NR9**, section 4.1.3. Note that conformance with *text/plain* is a prerequisite for using, for example, fragment URIs in the style of **NR12** for identifying text ranges.

2) The serialization conforms with XML **NR4**, which is a prerequisite for using XPointer fragment URIs for addressing subresources of an XML resource (cf. **NR13**).

Independently from the conformance levels given above, there is the following hierarchy of conformance with respect to 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 with respect to prefixed CURIE 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 **must** be declared, as this is the only possibility of mapping the identifiers of the native OMS to IRIs that are located in one flat namespace.

Any conforming serialization of an OMS language shall have a machine-processable description as detailed in clause 2.4.

2.4 Machine-Processable Description of Conforming Languages, Logics, and Serializations

Rationale: When a parser processes a DOL OMS found somewhere that 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, all languages/logics/serializations that conform with DOL are required to describe themselves in a machine-processable way and to be registered in the DOL registry.

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 is retrievable by dereferencing this IRI; this requirement follows the linked data principles **NR1**. Further, it is required that the language/logic/serialization is registered in the DOL registry (see annex A).

There may be an RDF description conforming to the vocabulary specified in annex B. That description may 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.

2.5 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 must 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,

— any 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.

2.6 Conformance of an Application With DOL

In the following, "DOL abstract syntax" means an XMI document that conforms to the DOL metamodel. Optionally, further representations (e.g., as JSON) can be supported.

— A *parser* is DOL-conformant if it can parse the DOL textual syntax and produce the corresponding DOL abstract syntax.

— A *printer* is DOL-conformant if it can read DOL abstract syntax and produce DOL textual syntax.

— DOL-conformant software that is used to *edit, format or manage* DOL libraries **must** be capable of reading and writing DOL abstract syntax. Moreover, it **must** meet the requirements for a DOL-conformant parser if it is able to read in DOL textual input. It **must** meet the requirements of a DOL-conformant printer if it is able to generate DOL textual output. However, it is also possible that a software for DOL management will work on the abstract syntax only, delegating the reading and generation of DOL text to external parsers and/or printers.

— a *static analyzer* is DOL-conformant if it can compute the logic and the signature of an OMS according to the semantics defined in section 10. In more detail, a static analyzer can have the following capabilities:

  — *simple analysis*: static analysis of DOL excluding networks and alignments;

  — *full analysis*: static analysis of full DOL.

— a *transformation tool* is DOL-conformant if it operates on DOL (abstract) syntax and implements one (or more) language translations, logic translations, language projections and/or logic projections.

— Software that implements machine *reasoning* about OMS (e.g., theorem proving, approximation) complies with this specification if and only if it interprets DOL documents according to the semantics defined in section 10. In more detail, a reasoning tool can have the following capabilities:

  — *simple logical consequence*, i.e., checking whether all sentences that are marked as `%implied` within basic OMS and extensions are logical consequences of the enclosing OMS;

  — *structured logical consequence*, i.e., checking whether all sentences that are marked as `%implied` are logical consequences of the enclosing OMS and whether all entailments in a DOL document have a defined semantics;

  — *interpretation*, i.e., checking whether all interpretations in a DOL document have a defined semantics;

  — *simple refinement*, i.e., checking whether all refinements of OMS in a DOL document have a defined semantics;

  — *full refinement*, i.e., checking whether all refinements (both of OMS and networks) in a DOL document have a defined semantics;

  — *simple conservativity*, i.e., checking whether all conservativity statements in a DOL document have a defined semantics;

  — *full conservativity*, i.e., checking whether all statements about conservative, monomorphic, definitional and weakly definitional extensions in a DOL document have a defined semantics;

  — *module extraction*, i.e., the ability to compute modules (typically, a given tool will provide this only for some logics);

  — *approximation*, i.e., the ability to compute approximations (typically, a given tool will provide this only for some logics and logic projections);

  — *full* DOL *reasoning*, i.e., checking whether an DOL document has a defined semantics.

In practice, DOL-aware *applications* may also deal with documents that are not conforming with DOL according to the criteria established in clause 2.5. However, an application only *conforms* with DOL if it is capable of producing DOL-conforming documents as its output when requested.

DOL-aware applications **shall** 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 DOL documents that mix OWL and Common Logic ontologies, as well as native OWL and Common Logic documents.

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

This page intentionally left blank.

3 Normative References

NR1 W3C/TR ldp Linked Data Platform. W3C Recommendation, 26 February 2015.
http://www.w3.org/TR/ldp/

NR2 W3C/TR owl2-syntax OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. W3C Recommendation, 11 December 2012.
http://www.w3.org/TR/owl2-syntax/

NR3 ISO/IEC 14977:1996 Information technology – Syntactic metalanguage – Extended BNF.

NR4 W3C/TR xml Extensible Markup Language (XML). W3C Recommendation, 26 November 2008.
http://www.w3.org/TR/xml/

NR5 W3C/TR owl2-primer OWL 2 Web Ontology Language: Primer. W3C Recommendation, 11 December 2012.
http://www.w3.org/TR/owl2-primer/

NR6 W3C/TR owl2-profiles OWL 2 Web Ontology Language: Profiles. W3C Recommendation, 11 December 2012.
http://www.w3.org/TR/owl2-profiles/

NR7 ISO/IEC 24707:2007 Information technology – Common Logic (CL): a framework for a family of logic-based languages.

NR8 OMG Document ptc/2013-09-05 OMG Unified Modeling Language (OMG UML).
http://www.omg.org/spec/UML/Current

NR9 IETF/RFC 2046 Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types.
https://www.ietf.org/rfc/rfc2046.txt

NR10 IETF/RFC 3986 Uniform Resource Identifier (URI): Generic Syntax. January 2005.
http://tools.ietf.org/html/rfc3986

NR11 IETF/RFC 3987 Internationalized Resource Identifiers (IRIs). January 2005.
http://tools.ietf.org/html/rfc3987

NR12 IETF/RFC 5147 URI Fragment Identifiers for the text/plain Media Type. April 2008.
http://tools.ietf.org/html/rfc5147

NR13 W3C/TR xptr-framework XPointer Framework. W3C Recommendation, 25 March 2003.
http://www.w3.org/TR/xptr-framework/

NR14 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/

NR15 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/

NR16 W3C/TR REC-rdfa-core:2015 RDFa Core 1.1 – Third Edition. Syntax and processing rules for embedding RDF through attributes. W3C Recommendation, 17 March 2015.
http://www.w3.org/TR/2015/REC-rdfa-core-20150317/

NR17 ISO/IEC 10646 Information technology – Universal Multiple-Octet coded Character Set (UCS).

NR18 W3C/TR rdf-schema RDF Schema. W3C Recommendation, 25 February 2014.
http://www.w3.org/TR/rdf-schema/

NR19 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/

NR20 W3C/TR REC-owl2-mapping-to-rdf:2012 OWL 2 Web Ontology Language Mapping to RDF Graphs (Second Edition). W3C Recommendation, 11 December 2012.
http://www.w3.org/TR/2012/REC-owl2-mapping-to-rdf-20121211/

NR21 DCMI Metadata Terms:2012 DCMI Metadata Terms, DCMI Recommendation, DCMI Usage Board, 14 July 2012.
http://dublincore.org/documents/2012/06/14/dcmi-terms/

NR22 W3C/TR skos-reference SKOS Simple Knowledge Organization System Reference. W3C Recommendation, 18 August 2009.
http://www.w3.org/TR/skos-reference/

NR23 OMG Specification Metadata:2014 Specification Metadata (SM) Vocabulary. OMG, 18 August 2014.
http://www.omg.org/techprocess/ab/20140801/SpecificationMetadata.rdf

NR24 ODM Ontology Definition Metamodel, 2 September 2014.
http://www.omg.org/spec/ODM/1.1/

NR25 MOF Meta Object Facility.
http://www.omg.org/spec/MOF/

NR26 SMOF Support for Semantic Structure, April 2013.
http://www.omg.org/spec/SMOF/1.0/

NR27 XMI Metadata Interchange (XMI) – using MOF 2 XMI, April 2014.
http://www.omg.org/spec/XMI/

NR28 SBVR Semantics Of Business Vocabulary And Rules, November 2013.
http://www.omg.org/spec/SBVR/

NR29 DTV Date-Time Vocabulary, August 2013.
http://www.omg.org/spec/DTV/1.0/

NR30 RIF Rule Interchange Format, February 2013.
http://www.w3.org/TR/rif-overview/

4 Terms and Definitions

4.1 Distributed Ontology, Modeling and Specification Language

Distributed Ontology, Modeling and Specification Language
DOL unified metalanguage for the structured and heterogeneous expression of ontologies, specifications, and models, using DOL libraries of OMS, OMS mappings and OMS networks, whose syntax and semantics are specified in this OMG Specification.

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

native document document containing a native OMS.

DOL document document containing a DOL library.

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 models, RDF Schema, and OBO.
Note An OMS language is used for the formal specification of native OMS.
Note An OMS language has a logical language aspect, a structuring language aspect, and an annotation language aspect.

DOL structured OMS syntactically valid DOL expression denoting an OMS that is built from smaller OMS as building blocks.
Note DOL structured OMS, typically, use basic OMS as building blocks for defining other structured OMS, OMS mappings or OMS networks.
Note All DOL structured OMS are structured OMS.

ontology explicit and shared formal representation of the entities and their interrelationships of a given domain of discourse or of fundamental notions
Note The explicit and shared formal representation is materialised in some OMS language (or several such languages).
Note Ontologies also include definitions and explanations in natural language that capture the intended meaning of the formal expressions.
Note Ontologies typically include a taxonomy and, frequently, a partonomy.

model representation of (the development of) a system (e.g., hardware, software or information system or organisation), or a domain related to a system, used in model-driven engineering (MDE)
Note Not to be confused with the term model in the sense of logic (model theory). In this document, we use the term realization for models in the sense of model theory in logic.

specification formal representation of (requirements of) a data structure, an algorithm or a hardware or software system used in systems analysis, requirements analysis and systems design

OMS (ontology, specification or model) collection of expressions (like non-logical symbols, sentences and structuring elements) in a given OMS language (or several such languages) and denoting a class of realizations and, possibly, a logical theory.
NOTE An OMS is either a basic OMS (which is always a native OMS, and can occur as a text fragment in a DOL document) or a structured OMS (which can be either a native structured OMS contained in some native document, or a DOL structured OMS contained in a DOL document).
NOTE An OMS has a single signature and class of realizations over that signature as its model-theoretic semantics.

basic OMS
flat OMS native OMS that does not utilize any elements from the structuring language aspect of its language.
NOTE Basic OMS are self-contained in the sense that their semantics does not depend on some other OMS. In particular, a basic OMS does not involve any imports.
NOTE Since a basic OMS has no structuring elements, it consists of (or at least denotes) a signature equipped with a set of sentences and annotations.
NOTE In signature-free logics like Common Logic or TPTP, a basic OMS only consists of sentences. A signature can be obtained a posteriori by collecting all non-logical symbols occuring in the sentences.

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

EXAMPLE Non-logical symbols in OWL **NR2** (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).

These non-logical symbols are distinguished from logical symbols in OWL, e.g., those for intersection and union of classes.

EXAMPLE Non-logical symbols in Common Logic **NR7** comprise

— names (denoting objects from the domain of discourse),

— sequence markers (denoting sequences of objects).

These non-logical symbols are distinguished from 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 notion of signature depends on the OMS language or logic.
NOTE The signature of an OMS is usually unequivocally determinable.

realization semantic interpretation of all non-logical symbols of a signature.
NOTE A realization of an OMS is a realization of the signature of the OMS that also satisfies all the additional

constraints expressed by the OMS. In case of flattenable OMS, these constraints are expressed by the axioms of the OMS.
NOTE The notion of realization depends on the OMS language or logic.
NOTE In logical model theory, a realization is called “model”. However, we have reserved the term “model” for models in the sense of model-driven engineering.

expression a finite combination of symbols that are well-formed according to applicable rules (depending on the language)

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 realization, i.e., which is assigned a truth value in this realization.
NOTE In a realization, 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. For example, a sentence can be: an axiom, if postulated to be true; a theorem, if proven from other axioms and theorems; or a conjecture, if expecting to be proven from other axioms and theorems.
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.
NOTE The notion of sentence depends on the OMS language or logic.

satisfaction relation relation between realizations and sentences indicating which sentences hold true in the realization.
NOTE The satisfaction relation depends on the OMS language or logic.

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

NOTE Each logical theory can also be written a basic OMS, and conversely each basic OMS has as its semantics a logical theory.

entailment
logical consequence
specialization relation between two OMS (or an OMS and a sentence, or two OMS networks, or an OMS network and an OMS) expressing that the second item (the conclusion) is logically implied by the first one (the premise).
NOTE Entailment expresses that each realization satisfying the premise also satisfies the conclusion.

NOTE The converse is generalization.

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

theorem sentence that has been proven from other axioms and theorems and therefore has been demonstrated to be a logical consequence of the axioms.

tool software for processing DOL libraries and OMS.

theorem proving process of demonstraing that a sentence (or OMS) is the logical consequence of some OMS.

theorem prover tool implementing theorem proving.

4.3 Structured OMS

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

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, approximation, filtering, or reference of named OMS involving only flattenable OMS.

elusive OMS OMS that is not flattenable.

subOMS OMS whose associated 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 verbatim included; also import of DOL 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 DOL library into another DOL library is also called import.

union DOL structured OMS expressing the aggregation of several OMS to a new OMS, without any renaming.

OMS translation DOL structured OMS expressing the assignment of new names to some non-logical symbols of an OMS, or translation of an OMS along a language translation.
NOTE An OMS translation results in an OMS mapping between the original and the renamed OMS.
NOTE Typically, the resulting OMS mapping of a translation is surjective: the symbols of the original OMS can be identified by the renaming, but no new symbols are added.

OMS reduction DOL structured OMS expressing the restriction of an OMS to a smaller signature.

local environment context for an OMS, being the signature built from all previously-declared symbols and axioms.

extension structured OMS extending a given OMS with new symbols and sentences.
NOTE The new symbols and sentences are interpreted relative to the local envorinment, which is the signature of the "given 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.

conservative extension extension that does not add new logical properties with respect to the signature of the extended OMS.
NOTE An extension is a consequence-theoretic or model-theoretic conservative extension. If used without qualification, the model-theoretic version is meant.

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 realizations of an OMS.
NOTE An extension O_2 of an OMS O_1 is a model-theoretic conservative extension, if each realization of O_1 can be expanded to a realization of O_2.
NOTE Each model-theoretic conservative extension is also a consequence-theoretic one, but not vice versa.

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 realization of O_1 can be expanded to a realization 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 realization of O_1 can be uniquely expanded to a realization of O_2.
NOTE O_2 being a definitional extension of O_1 implies a bijective correspondence between the classes of realizations 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 realization of O_1 can be expanded to at most one realization 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 class of realizations of O_2 is the class of realizations of O_1.
NOTE Each implied extension is also a definitional extension but not vice versa.

consistency property of an OMS expressing that it has a non-trivial set of logical consequences in the sense that not every sentence follows from the OMS.
NOTE The opposite is inconsistency.
NOTE In many (but not all) logics, consistency of an OMS equivalently can be defined as *false* not being a logical consequence of the OMS. However, this does not work for logics that e.g., do not feature a *false*. See [59] for a more detailed discussion.

satisfiability property of an OMS expressing that it is satisfied by least one realization.
NOTE The opposite is unsatisfiability.
NOTE Any satisfiable OMS is consistent, but there are some logics where the converse does not hold.

model finding process that finds realizations (models) of an OMS and thus proves it to be satisfiable.

model finder tool that implements model finding.

module structured OMS expressing a subOMS that conservatively extends to the whole OMS.
NOTE The conservative extension can be either model-theoretic or consequence-theoretic; without qualification, the model-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 [70].
NOTE The goal of module extraction is "decomposing an OMS into smaller, more manageable modules with appropriate dependencies" [69].

EXAMPLE Assume one extracts a module about white wines from an OWL DL ontology about wines of any kind. 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 logically implied theory (possibly after suitable translation) of an OMS in a smaller signature or a sublanguage.

maximum approximant best possible approximant of an OMS in a smaller signature or a sublanguage.
NOTE Technically, a maximum approximant is a uniform interpolant, see [46].

approximation structured OMS that expresses a maximum approximant.

filtering structured OMS expressing the verbatim removal of symbols or axioms from an OMS.
NOTE If a symbol is removed, all axioms containing that symbol are removed, too.

closed world assumption assumption that facts whose status is unknown are true.

closure
circumscription structured OMS expressing a variant of the closed world assumption by restricting the realizations to those that are minimal, maximal, free or cofree (with respect to the local environment).
NOTE Symbols from the local environment are assumed to have a fixed interpretation. Only the symbols newly declared in the closure are forced to have minimal or maximal interpretation.
NOTE DOL supports four different forms of closure: minimization and maximization as well as freeness and cofreeness (explained below).

NOTE See [50], [43].

minimization form of closure that restricts the realizations to those that are minimal (with respect to the local environment). **maximization** form of closure that restricts the realizations to those that are maximal (with respect to the local environment).

freeness special type of closure, restriction of realizations to those that are free (with respect to the local environment).
NOTE In first-order logic (and similar logics), freeness means minimal interpretation of predicates and minimal equality among data values. Freeness can be used for the specification of inductive datatypes like numbers, lists, trees, bags etc. In order to specify e.g., lists over some elements, the specification of the elements should be in the local environment.

cofreeness special type of closure, restriction of realizations to those that are cofree (with respect to the local environment).
NOTE In first-order logic (and similar logics), cofreeness means maximal interpretation of predicates and equality being observable equivalence. Cofreeness can be used for the specification of coinductive datatypes like infinite lists and streams.

combination structured OMS expressing the aggregation of all the OMS in an OMS network, where non-logical symbols are shared according to the OMS mappings in the OMS network.
EXAMPLE Consider an ontology involving a concept `Person`, and another one involving `Human being`, and an alignment that relates these two 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.

4.4 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.
NOTE Similar to correspondence.

signature morphism mapping between two signatures, preserving the structure of the source signature within the target signature
NOTE Each signature morphism has an underlying list of symbol map items. Conversely, a list of symbol map items may induce a signature morphism (but generally, it does not so in all cases).

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 [21].

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, with a confidence level.
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. [40] 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*).
NOTE A default correspondence can be used for stating that all symbols with the same name in the two ontologies are equivalent. A correspondence block can be used for specifying the relation and/or the confidence value of several single correspondences in the same time: if the relation or the confidence value of a single correspondence in a block are missing, they will be replaced with those specified as parameters of the block.

matching algorithmic procedure that generates an alignment for two given OMS.
NOTE For both matching and alignment, see [16] [32].

matcher tool that implements matching.

OMS network
distributed OMS
hyperontology graph with OMS as nodes and OMS mappings as edges, showing how the OMS are interlinked.
NOTE In [60], 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 engineering.
NOTE The links between the nodes of a network can be given using interpretations or alignments. Imports between the nodes of a network are automatically included in the network. 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 network 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.

category a collection of objects with suitable morphisms between them.
NOTE In this standard, objects of a category are usually signatures or OMS, and morphisms are signature morphisms, or OMS mappings. In principle, no assumption about the exact nature of objects and morphisms is made.
NOTE The morphisms determine which part of the structure of the objects is relevant, i.e., preserved by morphisms. Hence, objects can be seen as "sets with structure", and morphisms as "structure-preserving maps". However note that not all categories can be obtained in this way.

4.5 Features of OMS Languages

mapping
function relation between a set of inputs and a set of permissible outputs with the property that each input is related to exactly one output.
NOTE In some cases is a morphism, as in category theory.

language mapping mapping between languages
NOTE This is a general term, subsuming OMS language translation, logic translation and logic reduction below.

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.

graph set of objects (nodes) that are connected by links (edges).

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 The support relations specify which language supports which logics and which serializations, and which language translation supports which logic translation or reduction. Moreover, each language has a default logic and a default serialization.

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 meta classes (abstract syntax) and terminal and nonterminal symbols and grammar rules (concrete syntax).

language aspect a 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 interpreted 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.

4.6 Logic

logic specification of valid reasoning that comprises signatures (user defined vocabularies), realizations (interpretations of these), sentences (constraints on realizations), and a satisfaction relation between realizations and sentences.
NOTE Most OMS languages have an underlying logic.
EXAMPLE $\mathcal{SROIQ}(D)$ is the logic underlying OWL 2 DL.
NOTE See annex I 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, and one betwwen language translations and logic translations/reductions.

exact logical expressivity strengthening of the supports relation between languages and logics, stating that the language has exactly the expressivity of the logic.

institution metaframework mathematically formalizing the notion of a logic in terms of notions of signature, realization, sentence and satisfaction.
NOTE In order to support a broad range of OMS languages and enable interoperability between them, the DOL semantics has to abstract from the differences of the logic language aspects of OMS languages. Institutions provide a formal framework that enables this abstraction.
NOTE The notion of institution uses category theory for providing formal interfaces for the notions of signature, realization, sentence and satisfaction.

NOTE See Definition 2 in clause 10 for a formal definition.

plain mapping logic mapping that maps signatures to signatures and therefore does not use infrastructure axioms.

translation mapping between languages or logics representing all structure, in contrast to reduction.

reduction mapping between languages or logics forgetting parts of the structure, projection to a smaller language or logic.

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

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

simple theoroidal logic translation translation that maps signatures of the source logic to theories (i.e., signatures and sets of sentences, playing the role of infrastructure axioms) 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.

infrastructure axiom axiom that is used in the target of a logic translation in order to encode a signature of the source 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. These first order axioms are infrastructure axioms.

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 The opposite of heterogeneous OMS.

heterogeneous OMS OMS whose parts are formulated in different logics.
NOTE The opposite of homogeneous OMS.
EXAMPLE See section M.4.

faithful mapping logic mapping that preserves and reflects logical consequence.

model-expansive mapping logic mapping that has a surjective translation of realizations (ensuring faithfulness of the mapping).

model-bijective mapping logic mapping that has a bijective mapping of realizations.

exact mapping logic mapping that is compatible with certain DOL structuring constructs, e.g., union, OMS translation and OMS reduction.

weakly exact mapping logic mapping that is weakly compatible with certain DOL structuring constructs, e.g., union, OMS translation and OMS reduction.

embedding logic mapping that embeds the source into the target logic, using components that are embeddings and (in the case of translations of realizations) isomorphism.

sublogic logic embedding that is "syntactic" in the sense that signature and sentence translations are inclusions.

adjointness relation between a logic translation and a logic reduction, expressing that they share their sentence and translations of realizations, while the signature translations are adjoint to each other (in the sense of category theory).

4.7 Interoperability

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, an model is a representation of some aspect of a domain of interest using a normative modeling facility and modeling 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.

4.8 Abstract and Concrete Syntax

concrete syntax
serialization specific syntactic encoding of a given OMS language or of DOL.
NOTE Serializations serve as standard formats for exchanging DOL documents and OMS between human beings and 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, W3C specifications only require an RDF/XML implementation for OWL2 tools.
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 or DOL library using a given serialization.

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

abstract syntax
parse tree term language for representing documents in a machine-processable way
NOTE An abstract syntax can be specified as a MOF metamodel **NR25**. Then abstract abstract syntax documents can be represented as XMI **NR27** documents.

4.9 Semantics

formalization precise mathematical entity capturing an informal or semi-formal entity.

formal semantics assignment of a mathematical meaning to a language by mapping the abstract syntax to suitable semantic domains.
NOTE A formal semantics is a formalization of the meaning of a language.

semantic domain mathematically-defined set of values that can represent the intended meanings of language constructs.

semantic rule specification of a mapping from expressions for some meta class in the abstract syntax to a semantic domain.

global environment mapping from identifiers (IRIs) to values in semantics domains representing representing semantic information about a set of documents (the latter typically being distributed over the internet).

4.10 Semantic Web

resource something that can be globally identified.
NOTE **NR10**, 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 **NR11** 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). See **NR10**, 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 **NR1**[3)].
NOTE The linked data principles (adapted from **NR1** and its paraphrase at [75]) 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. (I.e., the IRI is treated as a URL (uniform resource locator).)

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 [26], is the most common format for publishing linked data. However, its usage is not mandatory.
NOTE Using HTTP content negotiation [17] it is possible to serve representations in different formats from the same URL.

4.11 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 $(\mathrm{subject}, \mathrm{predicate}, \mathrm{object})$ triple as defined by **NR14**, Section 3.1.

3) The original source is widely accepted but not formally a standard [42].

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 the preceding note, it is possible to interpret annotations under an RDF semantics. “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 [70].

This page intentionally left blank.

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
CURIE	Compact URI expression
DDL	Distributed description logic [4]
DOL	Distributed Ontology, Modeling and Specification Language
DTV	Date-Time Vocabulary
EBNF	Extended Backus-Naur Form
E-connections	a modular ontology language (closely related to DDL) [37]
F-logic	frame logic, an object-oriented ontology language
IRI	Internationalized Resource Identifier
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 [24]
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/XML	XML-based serialization of the OWL 2 language
P-DL	Package-based description logic
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
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
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

This page intentionally left blank.

6 Additional Information

(Informative)

6.1 How to Read This Specification

The initial five clauses of this specification describe the scope of the specification, determine conformance criteria, provide normative references, define terms and definitions, and introduce symbols that are used in the specification. The next three clauses are *informative*. This clause provides some background information, the next two provide a high-level summary of usage scenarios and goals (clause 7) and an overview over the design of DOL (clause 8).

Clause 9 defines the abstract syntax of DOL (*normative*) as an SMOF **NR26** compliant meta model. Further, the same clause also provides a human friendly text serialization of the abstract syntax of DOL (*normative*). Annex K contains the abstract syntax specified using Extended Backus–Naur Form (EBNF) (*informative*).

Clause 10 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 is about the DOL registry, which allows to register DOL conforming languages and translations (*normative*).

Annex B specifies an RDF vocabulary for the terms in clause 4, and for OMS languages and translation that conform with DOL (*informative*).

Various languages are shown to conform to DOL in informative annexes: OWL2 (annex C), Common Logic (annex D), RDF and RDF Schema (annex E), UML class models (annex F), TPTP (annex G), 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 L discusses an extension of DOL by queries. This extension is needed to support query languages (e.g., SQL or SPARQL) in DOL and to enable query related constructs for OMS in other DOL conformant languages (*informative*).

Annex M provides textual examples for all DOL constructs, which are specified in the abstract syntax (*informative*).

Annex N gives an overview of available 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 (Annex Q) contains references to the literature that is cited in this document (*informative*).

6.2 Acknowledgments

6.2.1 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

6.2.2 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, Free University of Bozen-Bolzano, Italy

— 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, Germany

— 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 Bozen-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

— Bernd Reichel, Otto-von-Guericke University Magdeburg, Germany

— Madhura Thosar, Otto-von-Guericke University Magdeburg, Germany

This page intentionally left blank.

7 Goals and Usage Scenarios

(Informative)

7.1 General

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 models, 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 is it possible to support shareability and reusability of OMS within the same domain?
- How is it possible to 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, this clause presents 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 are illustrated by the usage scenarios in this clause.

Most of the following subsections are illustrated with sample DOL libraries. These are always written in DOL, see the DOL Text Serialization in clause 9. Naturally, they also contain parts written in different OMS languages (e.g., OWL), the syntax of which is not described in this standard, but in other standard documents.

7.2 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 [31], Dolce Lite [48], 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; **NR22**) 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 checking 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 transitive.

```
%prefix( lang:  <http://purl.net/DOL/languages/>
                %% descriptions of languages ...
         trans: <http://purl.net/DOL/translations/> )%
                %% ... and translations

language lang:OWL2_DL
ontology Parthood_OWL =
  ObjectProperty: isProperPartOf
  Characteristics: Asymmetric
  SubPropertyOf: isPartOf
end
language lang:CommonLogic
ontology Parthood_CL =
  Parthood_OWL
  with translation trans:SROIQtoCL
  then
    (if (and (isProperPartOf x y) (isProperPartOf y z))
          (isProperPartOf x z))
end
```

OWL can express transitivity, but not together with asymmetry.

7.3 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 interplay between foundational and domain ontologies in various languages is depicted in Figure 8.2 below.

DOL provides the framework for integrating different domain ontologies, aligning these to foundational ontologies [16],[13] 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) [4] contains alignments between a number of foundational ontologies, expressing semantic relations between the aligned entities. For this use-case three such ontologies are considered, containing spatial and temporal concepts: DOLCE[5], GFO[6] and BFO[7], 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/>
           lang:  <http://purl.net/DOL/languages/>

        )%

language lang: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
```

DOL can be used to combine ontologies, while taking into account the semantic dependencies given by the alignments. In the following example the ontology `Space` is defined as a combination of three different ontologies (BFO, GFO, DolceLite) along three alignments.

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

7.4 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

[4] See http://www.thezfiles.co.za/ROMULUS/home.html

[5] See http://www.loa.istc.cnr.it/DOLCE.html

[6] See http://www.onto-med.de/ontologies/gfo/

[7] See http://www.ifomis.org/bfo/

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.

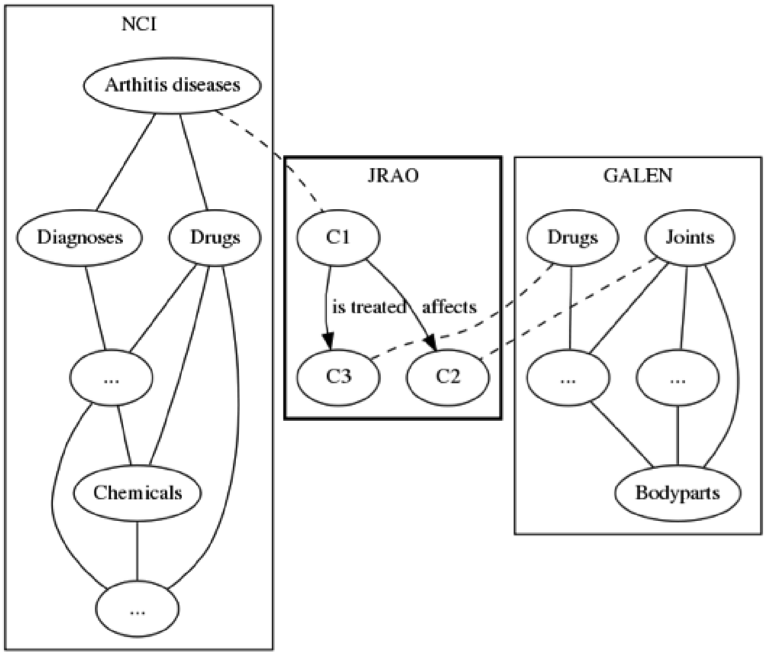


Figure 7.1: JRAO – Example for Module Extraction

```
%prefix( lang:  <http://purl.net/DOL/languages/> )%
library GalenModule
language lang:OWL
ontology myGalen =
  <http://purl.bioontology.org/ontology/GALEN> extract Drugs, Joints, Bodyparts
end

cons-ext myGalenIsAModule : <http://purl.bioontology.org/ontology/GALEN>
  of  myGalen
  for Drugs, Joints, Bodyparts
end
```

7.5 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 [2] require manual annotation of datasets with the information relevant for their processes. While there have been attempts to standardize such information [11], 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 [5].

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.

7.6 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. subclause 7.2). 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[8)] 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.[9)] 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.

7.7 Use Case Onto-6: Maintaining Different Versions of an Ontology in Languages with Different Expressivity

Often it is useful to maintain different versions of an ontology within languages, which differ in their expressivity.

8) http://www.w3.org/TR/2013/NOTE-prov-dc-20130430/

9) E.g., http://www.w3.org/TR/2013/NOTE-prov-dc-20130430/#dct-creator

For example, 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") [49]. 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 ontologies of different domains, including neuroimaging, computing, and ecology. Given the differences in expressivity between KIF and OWL, 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 supports the formalization of all of these views. Given suitable consistency checking tools, DOL enables the analysis of whether any such view satisfies all further axioms that the first-order DOLCE states about temporal parthood.

7.8 Use Case Onto-7: Metadata within OMS Repositories

DOL provides a language for the metadata within OMS Repositories. For example, the Common Logic Repository (COLORE) [10)] 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[11)], without specifying a formal semantics for them:

module provenance: author, date, version, description, keyword, parent ontology[12)]

axiom source provenance: name, author, year[13)]

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 DOL library.

7.9 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

[10)] http://stl.mie.utoronto.ca/colore/

[11)] http://stl.mie.utoronto.ca/colore/metadata.html

[12)] Note that this use of the term "module" in COLORE corresponds to the term structured OMS in this OMG Specification.

[13)] Note that this may cover any sentences in the sense of this OMG Specification.

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:

```
%prefix( lang:  <http://purl.net/DOL/languages/> )%

library Sorting

language lang: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: predicates `is_ordered` and `permutation` are hidden, because they are only auxiliary and need not be implemented.

7.10 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 [64], 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 7.9 into a specification of a particular sorting algorithm (for simplicity, insert sort is used for demonstration):

```
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 from abstract sorting to insert sort
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 the predicates `is_ordered` and `permutation` had not been hidden 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 R1 =
 Monoid refined via Elem |-> Nat to Nat
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 refined to R2
end

refinement R3''' = R1 refined to R2
```

It can be useful to also consider refinement of networks of OMS. Suppose that the specification `Nat` is extended in different ways: by a specification `Int` of integers, as well as by a specification `List` of lists. These three specifications form a network, see Figure 7.2. The network expresses the distributed character of the development: some people might use only `Int`, others only `List`, so there is no necessity to unite all specifications into one large specification.

```
spec Int = %mono
  Nat
then %mono
     generated type Int ::= __ - __(Nat;Nat)
     forall a,b,c,d: Nat
     .  a - b = c - d <=> a + d = c + b    %(equality_Int)%
     sort  Nat < Int
     forall a: Nat . a = a - 0             %(Nat2Int_embedding)%
end

spec List =
  Nat
then
  sort Elem
  free type List ::=  [] |  __ :: __ (Elem; List)
     op   #__: List -> Nat;
     forall x,: Elem; L: List
     . # [] = 0                            %(numberOf_nil_List)%
     . # (x :: L) = suc( # L )             %(numberOf_NeList_List)%
end

network NatIntList = Nat, Int, List
end
```

The network in Figure 7.2 can be refined to a network that is closer to implementation. This amounts to refining

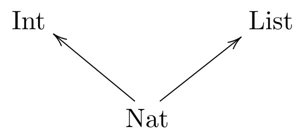


Figure 7.2: An OMS network consisting of three specifications

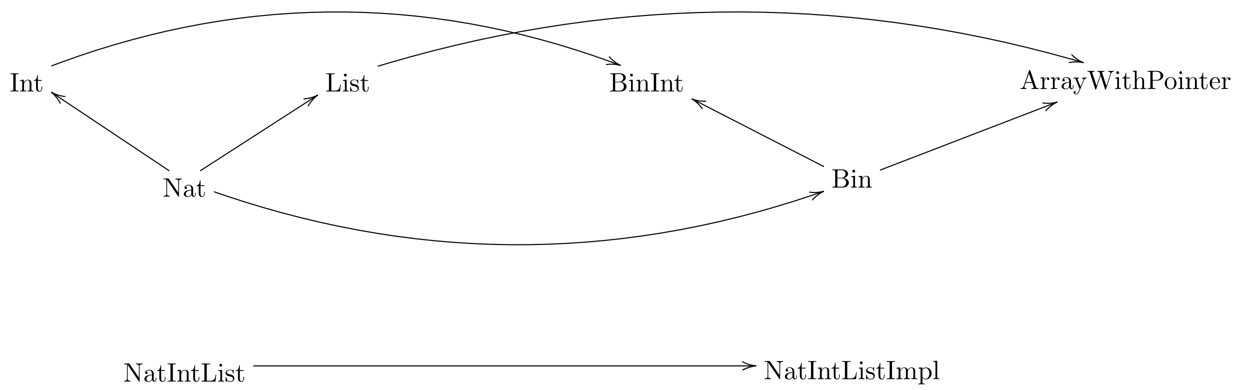


Figure 7.3: A refinement between OMS networks

the specifications of the network individually, see Figure 7.3. This needs to be done in such a way that both the refinement of `Int` and that of `List` are to built over the refinement of `Nat`.

```
spec IntBin = NatBin then ...
end
spec ArrayWithPointer = NatBin then ...
end
network NatIntListImpl = NatBin, IntBin, ArrayWithPointer
end
refinement NetworkRefinement =
  NatIntList refined via
      R2, %% Nat is refined to NatBin, see above
      Int refined via sort Int |-> BinInt to IntBin,
      List via sort List |-> Array to ArrayWithPointer
    to NatIntListImpl
end
```

7.11 Use Case Model-1: Consistency Among UML Models of Different Types

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

— whether the multiplicities in a class model are semantically consistent with each other;

— whether the sequential composition of actions in an interaction diagram is justified by an accompanying OCL specification;

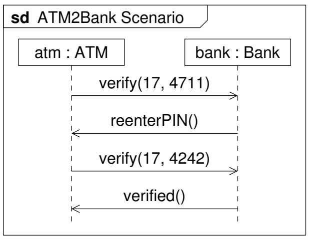
— whether cooperating state machines comply with pre-/post-conditions and invariants;

— whether the behavior prescribed in an interaction model is realizable by several state machines cooperating according to a composite structure model.

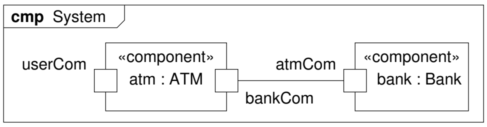
Such questions are currently hard to answer in a systematic manner. One method to answer these questions and find such errors is a check for semantic consistency. Under some restrictions, the proof of semantic consistency can be (at least partially) performed using model-checking tools like Hugo/RT [35]. Once a formal semantics for the different model types has been chosen (see, e.g., [34]), it is possible to use DOL to specify in which sense the models need to be consistent, and check this by suitable tools.

7.11.1 The ATM Example

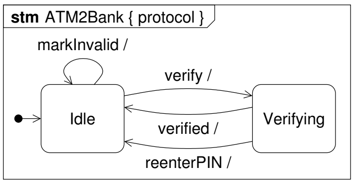
The ATM example, which illustrates model-driven development using UML, is taken from [34]. The example involves the design of a traditional automatic teller machine (ATM) connected to a bank. For simplicity, the example focuses on the ATM's processing of card and PIN entry actions. After entering the card, one has three trials for entering the correct PIN (which is checked by the bank). After three unsuccessful trials the card is kept.



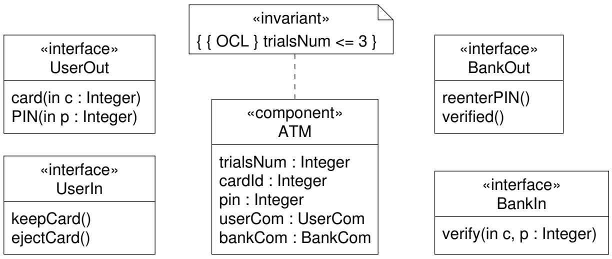
(a) Interaction



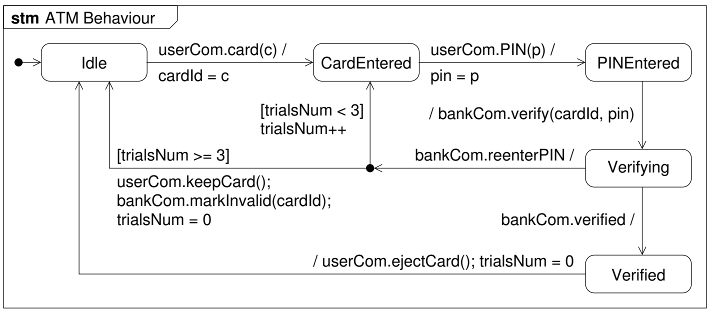
(b) Composite structure



(c) Protocol state machine



(d) Interfaces and components



(e) State machine

Figure 7.4: ATM example

Figure 7.4(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 model*, see Figure 7.4(b), which – among other things – specifies the objects in the initial system state. Furthermore, it specifies that the 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 Figure 7.4(c). The protocol state machine fixes in which order the messages verify, verified, reenterPIN, and markInvalid between atm and bank may occur. Figure 7.4(d) provides structural information in form of interfaces specifying what is provided and required at the userCom port and the bankCom port of the atm instance. An interface is a set of operations that other model elements have to implement. In our case, the interfaces are described in a *class model*. Its component type ATM is further enriched with the OCL constraint trialsNum <= 3, which refines its semantics requiring that trialsNum must not exceed three.

Finally, the dynamic behavior of the atm object is specified by the *behavioral state machine* shown in Figure 7.4(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 Figure 7.4 consists of five different UML models, which naturally form a network. Coherence of this network is expressed as its consistency. It is assumed that XMI **NR27** representations of the relevant UML models have been stored at `http://www.example.org/uml/` in a single xmi-file `http://www.example.org/uml/atm.xmi` that contains a `uml:Model` element for each UML model whose `xmi:id` has a prefix `xxx` followed by an underscore. `xxx` is determined as follows:

Figure	**xxx**	**diagram type**
Figure 7.4(a)	sd	sequence diagram
Figure 7.4(b)	cmp	composite structure diagram
Figure 7.4(c)	psm	protocol state machine
Figure 7.4(d)	cd	class diagram
Figure 7.4(e)	stm	state machine

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

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

7.12 Use Case Model-2: Refinements Between UML Models 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 models and OCL constraints. Assume further that this UML 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.

This can be illustrated based on the state machine of the atm, shown in Figure 7.4(e), which is a refinement of the protocol state machine in Figure 7.4(c). This can be stated as follows in DOL. [14]

```
refinement abstract_to_concrete_atm =
  psm refined via translation psm2sm to { atm and bank }
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.

7.13 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, **NR24**), which provides a number of syntactic transformations between such languages. One example is the OMG Date-Time Vocabulary (DTV, **NR29**). DTV has been formulated in different languages, each of which addresses different audiences:

— SBVR **NR28**: business users

— UML **NR8** (class models and OCL): software implementors

— OWL **NR2**: ontology developers and users

— Common Logic **NR7**: (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),

[14] It is assumed 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

— 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 UML model like the ATM model developed in section 7.11.1 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
```

To relate this ontology with the ATM model, various aspects need to be taken care of:

— Translating into shared language (in this case, 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> end

model clStateModel = xmiStateModel with
                     translation UMLState2CL
end

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

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

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

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
```

7.14 Conclusion

In this section, several use cases have been introduced. They illustrate many aspects of DOL and its usefulness in many situations in which different OMS artifacts might be leveraged and augmented to produce broader or more tractable models, ontologies, and specifications.

DOL has been designed to support of a wide range of formalisms and 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 DOL documents, 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.

The next sections present the metalanguage DOL; in particular, the syntax and the model-theoretic semantics. Further, various features of DOL will be discussed, which are based on best practices of modularity across the three areas of ontology design, formal specification, and model-driven development.

This page intentionally left blank.

8 Design Overview

(Informative)

8.1 General

The purpose of this clause is to briefly describe the overall guiding principles and constraints of DOL's syntax and semantics. It provides an overview of the most important and innovative language constructs of DOL. Details can be found in clause 9.

8.2 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. To address this diversity this specification does **not** propose a "universal" language that is intended to subsume all the others. Quite the opposite, the authors of this specification embrace the pluralism of OMS languages, and the purpose of DOL is to provide 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 models, CASL, OWL, Common Logic) without requiring any changes. These are called *native OMS*. A native OMS is serialized in a *native document*.
- DOL provides for defining new, *structured OMS* based on existing OMS.[15)] 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 10.

Each of these functions corresponds to a syntactic category in DOL: native OMS, structured OMS, OMS mappings, and OMS networks. They (together with imports) form the items in a DOL *library*, and are, in this sense, the most important metaclasses of DOL.

8.3 Features of DOL

DOL is a language enabling OMS interoperability. DOL is

15) Native OMS can also 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.

free: DOL is freely available for unrestricted use (as any OMG specification is).

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 (including 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.

8.4 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 again 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.

8.5 DOL in the Metamodeling Hierarchy

DOL uses the metamodeling hierarchy known from model-driven engineering (see Figure ??). 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.

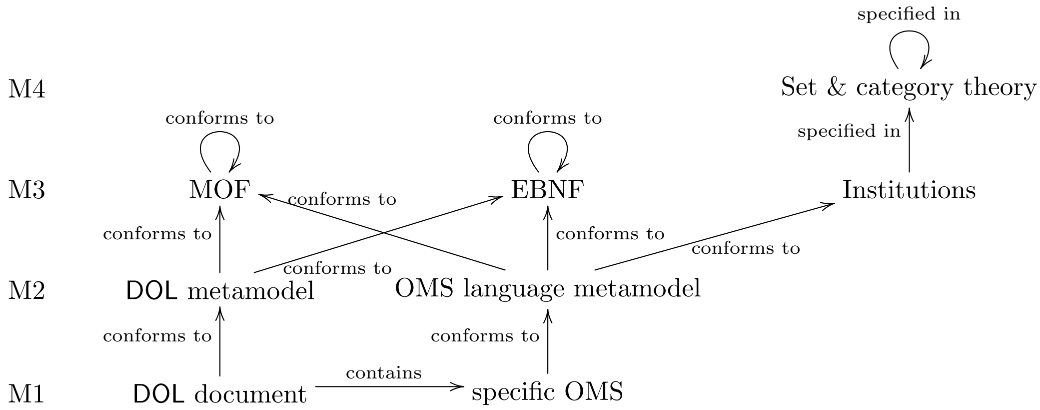


Figure 8.1: DOL **in the Metamodeling Hierarchy**

In the future, it may be possible to specify the semantics of a DOL conformant language using a semantics-based logical framework such as LF or MMT. Since LF can be specified in LF itself, this would close the loop already at M3 also for the semantics.

8.6 Semantic Foundations of DOL

A large variety of OMS languages in use can be captured at an abstract level using the concept of *institutions* [18]. This allows the development of DOL independently of the particularities of a logical system and to use the notions of institution and logical language interchangeably. 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, DOL provides 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 realizations 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* [51]. 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.

8.7 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 MOF metaclass `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 10). 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. However, DOL provides default translations, which are applied unless the user specifies a translation that deviates from the default. Both default and non-default translations may be combined to multi-step translations.

8.8 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 8.2. OMS mappings supported by DOL include:

— imports (particularly including imports that lead to conservative extensions), see the MOF metaclasses `OMSReference` and `ExtensionOMS` in clause 9.

— interpretations (both between OMS and OMS networks), see the MOF metaclass `InterpretationDefinition` in clause 9.

— alignments between OMS, see the MOF metaclass `AlignmentDefinition` in clause 9.

— conservative extensions, e.g., mappings between OMS and their modules, see the MOF metaclass `Conservative-ExtensionDefinition` in clause 9.

DOL uses symbol maps to express signature translations in such OMS mappings; see the MOF metaclass `SymbolMap` 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 MOF metaclass `OMS` in clause 9.

8.9 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 4.11.

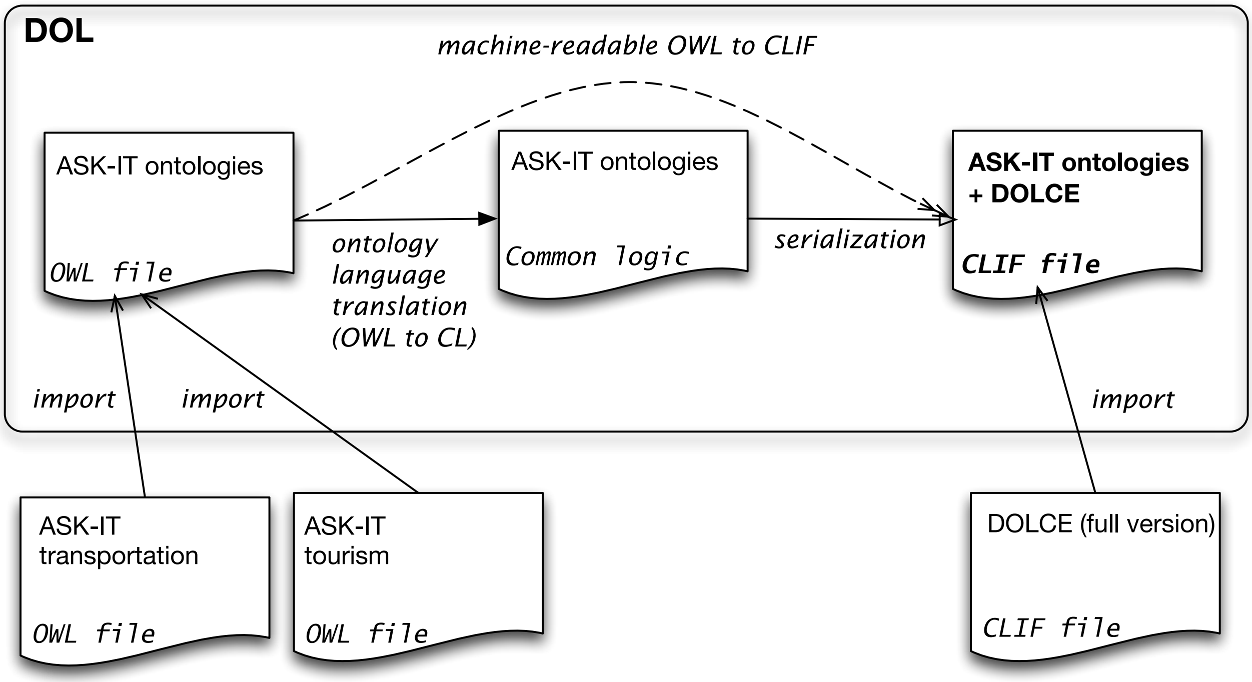


Figure 8.2: Mapping between two OMS formulated in different OMS languages

This page intentionally left blank.

9 DOL Syntax

9.1 General

This clause specifies the DOL abstract syntax as a MOF **NR25** metamodel. In annex K, the same abstract syntax is specified using EBNF. We further include the DOL concrete syntax, which uses the metaclasses of the abstract syntax as non-terminals of an EBNF grammar.

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, the `'end'` is not marked as optional, because it may be needed to effectively disambiguate heterogeneous texts.

The concrete syntax in EBNF relates to the abstract syntax in MOF via a simple scheme. Each non-terminal in the EBNF conforms to either a class or an attribute of a class in MOF. By default non-terminals are represented as classes. Non-terminals are represented as attributes in MOF only if in the corresponding EBNF production rule a non-terminal (a) produces a single non-terminal (e.g., IRI or String) and (b) is not part of an alternative in another rule. Each generalization in MOF yields a EBNF rule that consists solely of alternatives of single non-terminals. The properties of a MOF class form a ENBF rule for the corresponding non-terminal, which produces the concatenation of the property types of the class with syntactic terminals. Cardinality '0..1' in MOF is represented as options in EBNF. Analogously, the cardinality '0..*' in MOF is represented by the repetition symbol '*' in EBNF.

The DOL document types are as follows

MIME type: *application/dol+text*

Filename extension: .dol

9.2 MOF Metaclasses

DOL provides MOF metaclasses for (among others):

— OMS (which can be native OMS in some OMS language, or unions, translations, closures, combinations, approximations of OMS, among others)

— OMS mappings

— OMS networks

— DOL libraries (items in these are: definitions of OMS, OMS mappings, and OMS networks, as well as qualifications choosing (1) the logic, (2) the OMS language and/or (3) the serialization)

— identifiers

— annotations

The DOL metaclasses `NativeDocument` and `BasicOMS` are abstract metaclasses without any instances within the normative DOL metamodel. In order to use DOL with some specific conforming OMS language, the top-level MOF metaclass of the abstract syntax of this language (cf. clause 2.2) has to be a subclass (in the sense of SMOF multiple classification) of the DOL metaclass `NativeDocument`, see Figure 9.1. Likewise, if the conforming OMS language has a metaclass for basic OMS, this has to be a subclass of the metaclass `BasicOMS`, see Figure 9.2. See the informative annexes C to H for details.

9.3 Documents

9.3.1 Abstract Syntax

The DOL metamodel for documents and libraries is shown in Figure 9.3. A document (`Document`) can be a

— a DOL library, or

— a `NativeDocument`, which is the verbatim inclusion of an OMS written in an OMS language that conforms with DOL; cf. 2.2).

A DOL library consists of a collection of (named) OMS, OMS networks, and mappings between these. More specifically, a DOL library consists of a name, followed by a list of `LibraryItems`. A `LibraryItem` is either a `Definition`, an import of another DOL library (`LibraryImport`), or a `Qualification` selecting a specific OMS language, logic and/or syntax that is used to interpret the subsequent `LibraryItems`. A `LibraryImport` leads to the inclusion of all `LibraryItems` of the imported DOL library into the importing one. A `Definition` assigns an IRI to an OMS (`OMSDefinition`), to a mapping between OMS (`MappingDefinition`), or an OMS network (`NetworkDefinition`). Moreover, annex L informatively introduces `QueryRelatedDefinition`.

At the beginning of a DOL library, one can declare a `PrefixMap` for abbreviating long IRIs using CURIEs; see clause 9.7 for further details. Examples of the use of DOL library can be found in Appendix M and Section 7.

9.3.2 Concrete Syntax

9.3.2.1 Documents

```
Document           ::= DOLLibrary | NativeDocument
DOLLibrary         ::= [PrefixMap] 'library' LibraryName
                           Qualification LibraryItem*
NativeDocument     ::= <language and serialization specific >
LibraryItem        ::= LibraryImport | Definition | Qualification
Definition         ::= OMSDefinition
                     | NetworkDefinition
                     | MappingDefinition
LibraryImport      ::= 'import' LibraryName
```

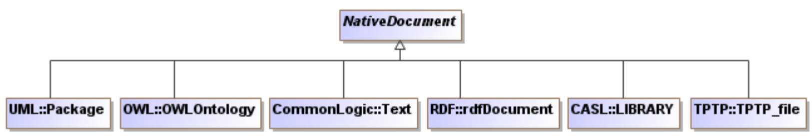


Figure 9.1: Informative diagram showing subclasses of `NativeDocument`

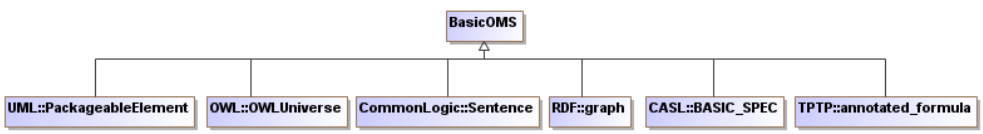


Figure 9.2: Informative diagram showing subclasses of `BasicOMS`

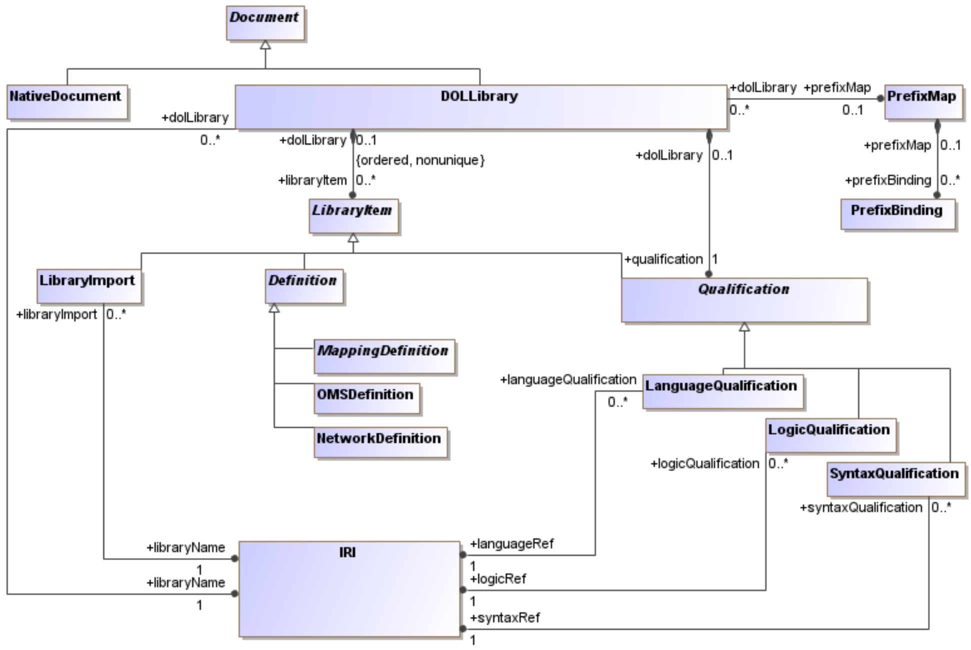


Figure 9.3: DOL metamodel: Documents and libraries

```
Qualification       ::= LanguageQualification
                      | LogicQualification
                      | SyntaxQualification
LanguageQualification ::= 'language' LanguageRef
LogicQualification ::= 'logic' LogicRef
SyntaxQualification ::= 'serialization' SyntaxRef
LibraryName         ::= IRI
LanguageRef         ::= IRI
LogicRef            ::= IRI
SyntaxRef           ::= IRI
```

```
PrefixMap       ::= '%prefix(' PrefixBinding* ')%'
PrefixBinding   ::= BoundPrefix IRIBoundToPrefix [Separators]
BoundPrefix     ::= ':' | Prefix <see definition in clause 9.7.2>
IRIBoundToPrefix ::= '<' FullIRI '>'
Separators      ::= 'separators' SeparatorString SeparatorString
SeparatorString ::= SeparatorChar SeparatorChar*
SeparatorChar   ::= ipchar | gen-delims - '#'<as defined in NR11>
```

Note that the empty prefix (called “no prefix” in **NR16**, Section 6) is denoted by a colon inside the prefix map, but it is omitted in CURIEs. This is the style of the OWL Manchester syntax [23] but differs from the RDFa Core 1.1 syntax.

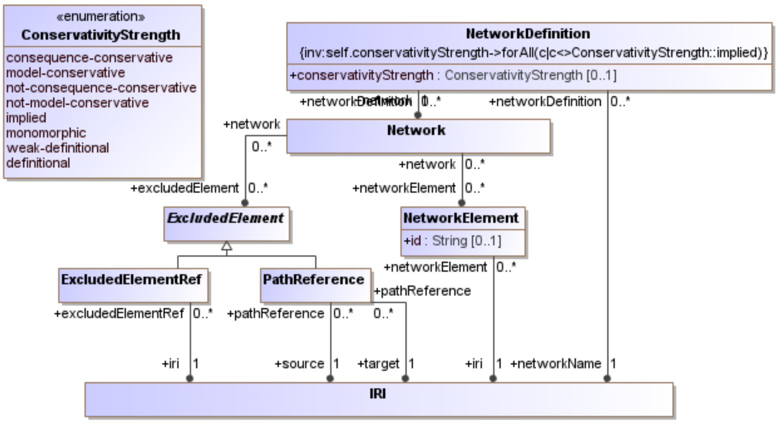


Figure 9.4: DOL metamodel: Networks

9.4 OMS Networks

9.4.1 Abstract Syntax

The DOL metamodel for documents and libraries is shown in Figure 9.3. Inside a DOL library, with a `NetworkDefinition`, one can define OMS networks (also called distributed OMS). OMS networks are typically used for complex viewpoint specifications; they also can be used in `combinations` (see clause 9.5 below). A `NetworkDefinition` names an OMS network consisting of `NetworkElements`. These can be `ElementRefs`, i.e., IRIs that name OMS, OMS mappings, or previously-defined OMS networks. `ElementRefs` that are OMS can be prefixed with an `Id`; this is then used for disambiguation in a `combination`. An optional `ConservativityStrength` specifies e.g., consistency of the network (analogously to `OMSDefinitions`, see clause 9.5 below for details).

An OMS network by default also includes all inclusions (between the extended and the extending OMS of an `ExtensionOMS`) between the involved OMS—unless these are explicitly excluded. The latter can be achieved using `ExcludingElements`. They consist of `ElementRefs` naming OMS or OMS mappings, and of `PathReferences`. A `PathReference` refers to an unnamed OMS mapping (e.g., one generated by an `Extension`) by specifying its source and target OMS. See Clauses 7.10, 7.11.1 and Appendix M.7 for an example network and of the use of `combination`.

9.4.2 Concrete Syntax

```
NetworkDefinition ::= 'network' NetworkName '='
                       [ConservativityStrength] Network
NetworkName      ::= IRI
Network          ::= NetworkElements [ExcludedElements]
NetworkElements ::= NetworkElement ( ',' NetworkElement )*
NetworkElement   ::= [Id ':'] ElementRef
ExcludedElements ::= 'excluding' ExcludedElement ( ',' ExcludedElement )*
ExcludedElement ::= PathReference | ElementRef
PathReference    ::= IRI '->' IRI
ElementRef       ::= IRI
Id               ::= Letter LetterOrDigit*
```

9.5 OMS

9.5.1 Abstract Syntax

The DOL metamodel for OMS is shown in Figure 9.5. DOL provides a rich structuring language for OMS, providing extension, translation, unions of OMS and many more. For each of these alternatives, a subclass is introduced. An OMS can be

— a `TranslationOMS` involving both an OMS (to be translated), and a specification of the translation, which is covered by the class `OMSTranslation` (see Appendix M.4, M.8, for examples);

— a `UnionsOMS`, uniting two given OMS (see Appendix M.3 for an example);

— a `ClosureOMS`, applying a closure operator (given by a `Closure`) to an OMS (see Appendix M.5 and M.11 for examples);

— an `ExtensionOMS`, extending a given OMS with another OMS (given by the `Extension`). 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 (see Appendix M.3, M.4, M.5 for examples);

— an `ExtendingOMS`, which is a very simple form of OMS, namely a basic OMS or an OMS reference (see below);

— a `FilteringOMS`, applying a filtering operator (given by a `Filtering`) to an OMS (see Appendix M.9.1.4 for an example);

— an `ApproximationOMS`, applying an approximation operator (given by an `Approximation`) to an OMS (see Appendix M.9 for an example);

— a `CombinationOMS`, giving a combination of (the OMS contained in) an OMS network (technically, this is a colimit, see [77]) (see Appendix M.7 for an example of the use of `combination`);

— a `ReductionOMS`, applying a reduction (given by an `Reduction`) to an OMS (see use cases 7.9, 7.10 and 7.11 and Appendix M.9 and M.10 for examples);

— a `ExtractionOMS`, applying a module extraction operator (given by an `Extraction`) to an OMS (see use case 7.4 for an example);

— a `QualifiedOMS`, which is an OMS qualified with the OMS language that is used to express it.

Moreover, annex L informatively introduces `Applications`, which apply a substitution to an OMS.

A `ConservativityStrength` specifies additional relations that may hold between an OMS and its extension (or union with other OMS), like conservative or definitional extension. The rationale is that the extension should not have impact on the original OMS that is being extended.

An OMS definition `OMSDefinition` names an OMS. It can be optionally marked as inconsistent, consistent, monomorphic or having a unique realization using `ConservativityStrength`. More precisely, `'consequence-conserva` here requires the OMS to have only tautologies as signature-free logical consequences, while `'notconsequence-conservati` expresses that this is not the case. `'model-conservative'` requires satisfiability of the OMS, `'not-model-conservativ` its unsatisfiability. `'definitional'` expresses that the OMS has a unique realization (see Appendix M.5 for an example); this may be interesting for characterizing OMS (e.g., returned by model finders) that are used to describe single realizations.

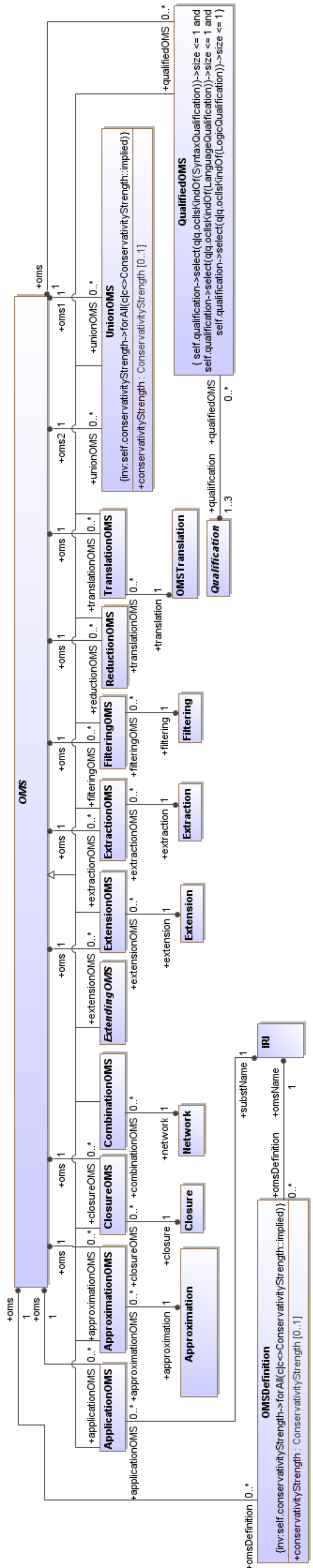


Figure 9.5: DOL metamodel: OMS

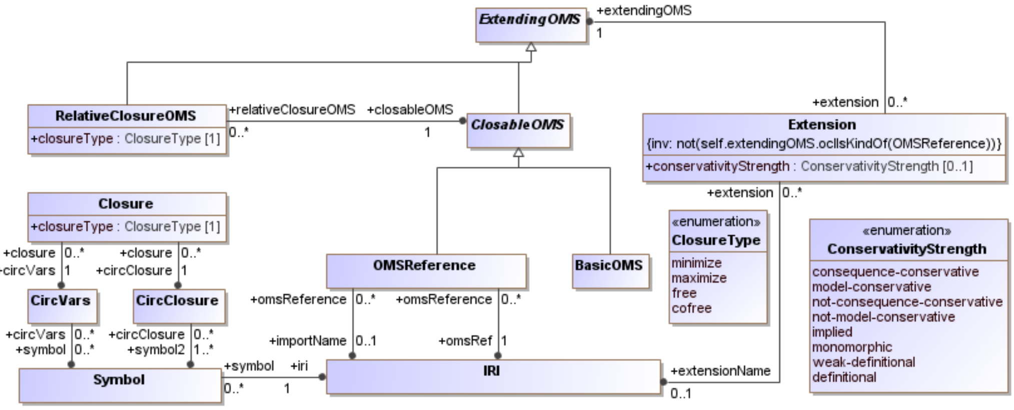


Figure 9.6: DOL metamodel: Extension and closure OMS

The DOL metamodel for extension OMS is shown in Figure 9.6. `ExtendingOMS` is a subclass of `OMS`, containing those OMS that may be used to extend a given OMS within an `ExtensionOMS`. An `ExtendingOMS` 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; practically every example uses basic OMS)[16]. Note that a basic OMS used inside a DOL document may not use any of the DOL keywords (see clause 9.8.1); otherwise, it needs to be enclosed in curly braces[17];

— a reference (through an IRI) to an OMS (`OMSReference`, many examples illustrate this); or

— a `RelativeClosureOMS`, applying a closure operator to a basic OMS or OMS reference (these two are hence joined into `ClosableOMS`). A closure forces the subsequently declared non-logical symbols to be interpreted in a minimal or maximal way, while the non-logical symbols declared in the local environment are fixed.[18] Variants of closure are minimization, maximization, freeness (minimizing also data sets and equalities on these, which enables the inductive definition of relations and datatypes), and cofreeness (enabling the coinductive definition of relations and datatypes). See Annex M.6 for examples of the former two, and Annex M.11 for examples of the latter two.

Recall that the local environment is the OMS built from all previously-declared symbols and axioms.

Using `ExtendingOMS`, extensions of an OMS with an `ExtendingOMS` can be built. The latter can optionally be named and/or marked as conservative, monomorphic, definitional, weakly definitional or implied (using a `ConservativityStrength`, see clause 4.3 for details). Note that an `ExtendingOMS` used in an extension must not be an `OMSReference`.

Furthermore, OMS can be constructed using

— closures of an OMS with a `Closure`. This is similar to a `RelativeClosureOMS`, but the non-logical symbols to be minimized/maximized and to be varied are explicitly declared here (while a `RelativeClosureOMS` takes the local environment to be fixed, i.e., not varied);

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

[17] This restriction applies to DOL documents only, not to native documents.

[18] Note that if applied to algebraic signatures (sorts and operation symbols), minimization can be used to express reachability (i.e., term-generatedness) of algebraic (first-order) realizations.

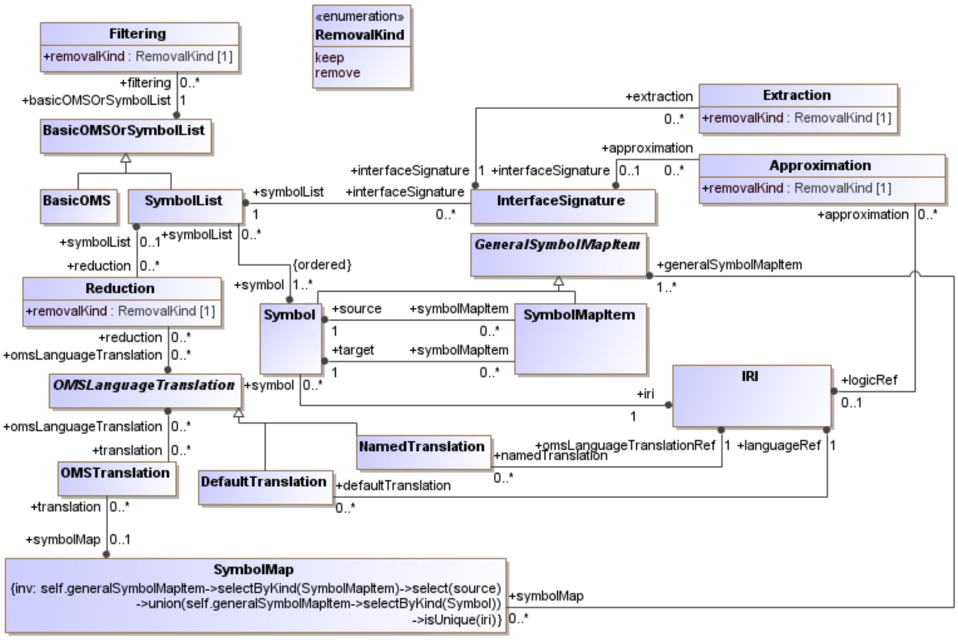


Figure 9.7: DOL metamodel: Translation and reduction OMS

— a translation `OMSTranslation` of an OMS into a different signature or OMS language. The former is done using a `SymbolMap`, specifying a map of symbols to symbols. The latter is done using an OMS language translation `OMSLanguageTranslation` 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);

— a `Reduction` of an OMS to a smaller signature and/or less expressive logic (that is, some non-logical symbols and/or some parts of the structure of the realization are hidden, but the semantic effect of sentences involving these is kept). The former is done using a `SymbolList`, which is a list of non-logical symbols that are to be hidden. The latter uses an `OMSLanguageTranslation` denoting a logic projection that is used as logic reduction to a less expressive OMS language.

— an `Approximation` of an OMS, in a subsignature (`InterfaceSignature`) or sublogic, with the effect that sentences not expressible in the subsignature respectively sublogic are replaced with a suitable approximation,

— a `Filtering` of an OMS, with the effect that some signature symbols and axioms (specified by a `BasicOMS`) are removed from the OMS,

— a module `Extraction` of an OMS, using a restriction signature (`InterfaceSignature`).

In all of these cases except for translation, a `RemovalKind` specifies whether the listed symbols are removed from the OMS, or whether they are kept (and the other ones are removed).

The DOL metamodel for closure OMS is shown in Figure 9.6, that for translation and reduction OMS in Figure 9.7.

9.5.2 Concrete Syntax

While in most cases the translation from concrete to abstract syntax is obvious (the structure is largely the same),

— both `%satisfiable`, `%cons` and `%mcons` are translated to `model-conservative`,

— both `%consistent` and `%ccons` are translated to `consequence-conservative`,

— both `%unsatisfiable` and `%notmcons` are translated to `not-model-conservative`,

— both `%inconsistent` and `%notccons` are translated to `not-consequence-conservative`,

— moreover, both `closed-world` and `minimize` are translated to `minimize`.

Note that the MOF abstract syntax subsumes all these elements except from those in the last line under the enumeration class `ConservativityStrength`. Not all elements of the enumeration can be used at any position; the corresponding restrictions are expressed as OCL constraints. By contrast, the concrete syntax features a more fine-grained structure of non-terminals (`Conservative`, `ConservativityStrength` and `ExtConservativityStrength`) in order to express the same constraints via the EBNF grammar.

```
BasicOMS             ::= <language and serialization specific>
ClosableOMS          ::= BasicOMS | '{' BasicOMS '}' | OMSRef [ImportName]
ExtendingOMS         ::= ClosableOMS | RelativeClosureOMS
RelativeClosureOMS ::= ClosureType '{' ClosableOMS '}'
OMS                  ::= ExtendingOMS
                       | OMS Closure
                       | OMS OMSTranslation
                       | OMS Reduction
                       | OMS Extraction
                       | OMS Approximation
                       | OMS Filtering
                       | OMS 'and' [ConservativityStrength] OMS
                       | OMS 'then' ExtensionOMS
                       | Qualification* ':' GroupOMS
                       | 'combine' NetworkElements [ExcludeExtensions]
                       | GroupOMS
Closure              ::= ClosureType CircMin [CircVars]
ClosureType          ::= 'minimize'
                       | 'closed-world'
                       | 'maximize'
                       | 'free'
                       | 'cofree'
CircMin              ::= Symbol Symbol*
CircVars             ::= 'vars' (Symbol Symbol*)
GroupOMS             ::= '{' OMS '}' | OMSRef
OMSTranslation       ::= 'with' LanguageTranslation* SymbolMap
                       | 'with' LanguageTranslation+
LanguageTranslation ::= 'translation' OMSLanguageTranslation
Reduction            ::= 'hide' LogicReduction* SymbolList
                       | 'hide' LogicReduction+
                       | 'reveal' SymbolList
LogicReduction       ::= 'along' OMSLanguageTranslation
SymbolList           ::= Symbol ( ',' Symbol )*
SymbolMap            ::= GeneralSymbolMapItem ( ',' GeneralSymbolMapItem )*
Extraction           ::= 'extract' InterfaceSignature
                       | 'remove' InterfaceSignature
Approximation        ::= 'forget' InterfaceSignature ['keep' LogicRef]
                       | 'keep' InterfaceSignature ['keep' LogicRef]
                       | 'keep' LogicRef
Filtering            ::= RemovalKind BasicOMSOrSymbolList
RemovalKind          ::= 'reject' | 'select'
BasicOMSOrSymbolList ::= '{' BasicOMS '}' | SymbolList
ExtensionOMS         ::= [ExtConservativityStrength]
                         [ExtensionName]
                         ExtendingOMS
```

```
ConservativityStrength ::= Conservative | '%mono' | '%wdef' | '%def'
ExtConservativityStrength ::= ConservativityStrength | '%implied'
Conservative       ::= '%cons'
                     | '%ccons'
                     | '%mcons'
                     | '%notccons'
                     | '%notmcons'
                     | '%consistent'
                     | '%inconsistent'
                     | '%satisfiable'
                     | '%unsatisfiable'
InterfaceSignature ::= SymbolList
ImportName         ::= '%(' IRI ')%'
ExtensionName      ::= '%(' IRI ')%'
OMSkeyword         ::= 'ontology'
                     | 'onto'
                     | 'specification'
                     | 'spec'
                     | 'model'
                     | 'oms'
OMSDefinition      ::= OMSkeyword OMSName '='
                       [ConservativityStrength] OMS 'end'
Symbol             ::= IRI
SymbolMapItem      ::= Symbol '|->' Symbol
GeneralSymbolMapItem ::= Symbol | SymbolMapItem
Sentence           ::= <an expression specific to an OMS language>
OMSName            ::= IRI
OMSRef             ::= IRI
LoLaRef            ::= LanguageRef | LogicRef
```

```
OMSLanguageTranslation ::= OMSLanguageTranslationRef | '->' LoLaRef
OMSLanguageTranslationRef ::= 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.

9.6 OMS Mappings

9.6.1 Abstract Syntax

An OMS mapping provides a connection between two OMS. An OMS mapping definition is the definition of either a named interpretation (`InterpretationDefinition`, see Annex M.3 for an example), entailment (`EntailmentDefinition`, see use case 7.11.1 for an example), refinement (`RefinementDefinition`, see use cases 7.10 and 7.11.1 for examples) or equivalence (`EquivalenceDefinition`, see Annex M.3 for an example), a named declaration of the relation between a module of an OMS and the whole OMS (`ConservativeExtension-Definition`, see use case 7.4 for an example), or a named alignment (`AlignmentDefinition`, see use case 7.3 and Annex M.7 for examples).

The DOL metamodel for interpretations and refinements is shown in Figure 9.8. Both interpretations and refinements specify a logical entailment or specialization relation between OMS.
An `InterpretationDefinition` specifies source and target OMS (forming the `InterpretationType`), as well as a `SymbolMap` and/or an `OMSLanguageTranslation`. The `SymbolMap` in an interpretation always must lead to a signature morphism. A proof obligation expressing that the source OMS, when translated along the signature morphism and/or the `OMSLanguageTranslation`, logically follows from the target OMS.

A symbol map in an interpretation is **required** to cover all non-logical symbols of the source OMS; the semantics specification in clause 10 makes this assumption. (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 is a non-injective OMS mapping.)

`Refinements` subsume interpretations (via `SimpleRefinements`), but allow the specification of much more complex relation between OMS (and OMS networks). The style differs from interpretation in that even a single OMS is a refinement (via `RefinementOMS`); this corresponds to the source of an interpretation. Using `SimpleOMSRefinements`, a refinement can be further specialized to a (target) OMS via an `OMSRefinementMap`. The latter involves a symbol map and/or OMS language translation, analogously to interpretations. With this style of notation, simple refinements can be easily chained up (which cannot be done using interpretations). Refinements themselves can also be refined, also by other refinements—this amounts to the possibility of composing refinements. Furthermore, refinements can also be specified between networks (`SimpleNetworkRefinement`, see use case 7.10 for an example). A refinement between OMS networks consists of a list of ordinary refinements (between OMS), one for each node in the source network (the OMS refinement is then required to refine the node in the source network to some node in the target network). The list may also include network refinements, much in the same way as network definitions also may include other networks. All the ordinary refinements occuring as components of the network refinement have to be compatible in a sense exemplified at the end of clause 7.10.

The DOL metamodel for entailments and equivalences is shown in Figure 9.9. An entailment is a variant of an interpretation where all symbols are mapped identically, while an equivalence states that the classes of realizations of two OMS are in bijective correspondence. As for refinements, entailments and equivalences are also possible between networks (`NetworkNetworkEntailment` and `NetworkEquivalence`). An entailment between a network as premise and an OMS as conclusion (`NetworkOMSEntailment`) specifies that all realizations of the network, when restricted to a given node (given by an IRI), are realizations of the OMS.

The DOL metamodel for alignments is shown in Figure 9.10. Signature morphisms used in interpretations and refinements use a functional style of mapping symbols of OMS. In contrast to this style, 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 [13]. 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 'Corre-

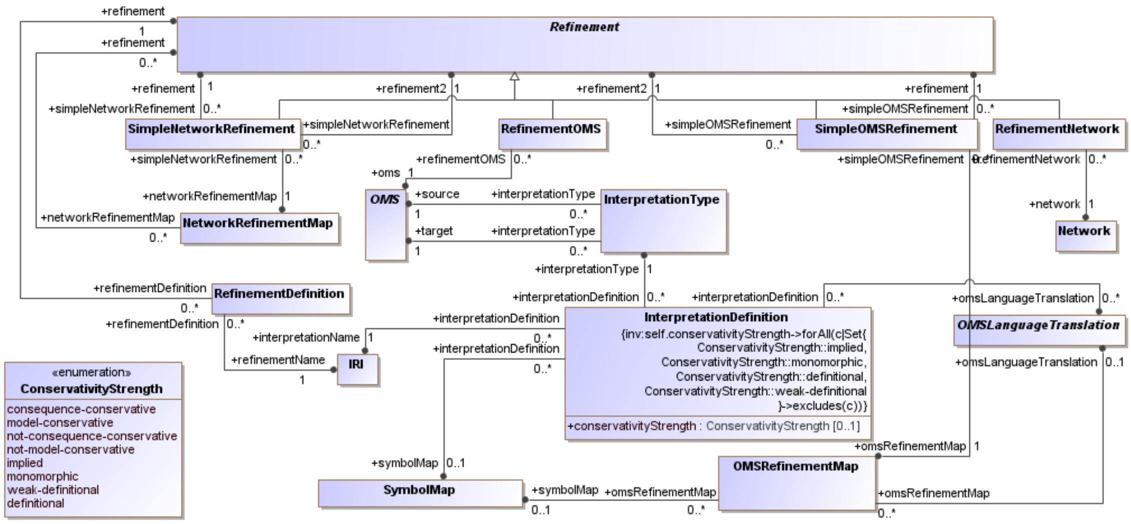


Figure 9.8: DOL metamodel: Interpretations and refinements

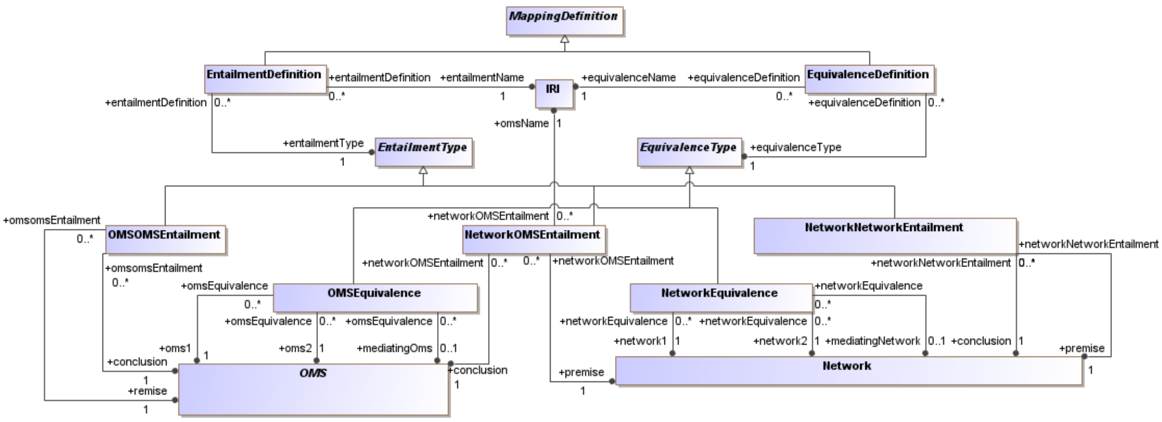


Figure 9.9: DOL metamodel: Entailments and equivalences

spondenceBlock'. Using an `AlignmentCardinality`, 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 `AlignmentSemantics`, 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.

The DOL metamodel for conservative extension definitions is shown in Figure 9.11. A `ConservativeExtension-Definition` declares that a certain ("whole) OMS actually is a conservative extension some other ("module") OMS with respect to the `InterfaceSignature`.

9.6.2 Concrete Syntax

```
MappingDefinition  ::= InterpretationDefinition
                     | EntailmentDefinition
                     | EquivalenceDefinition
                     | ConservativeExtensionDefinition
                     | AlignmentDefinition
```

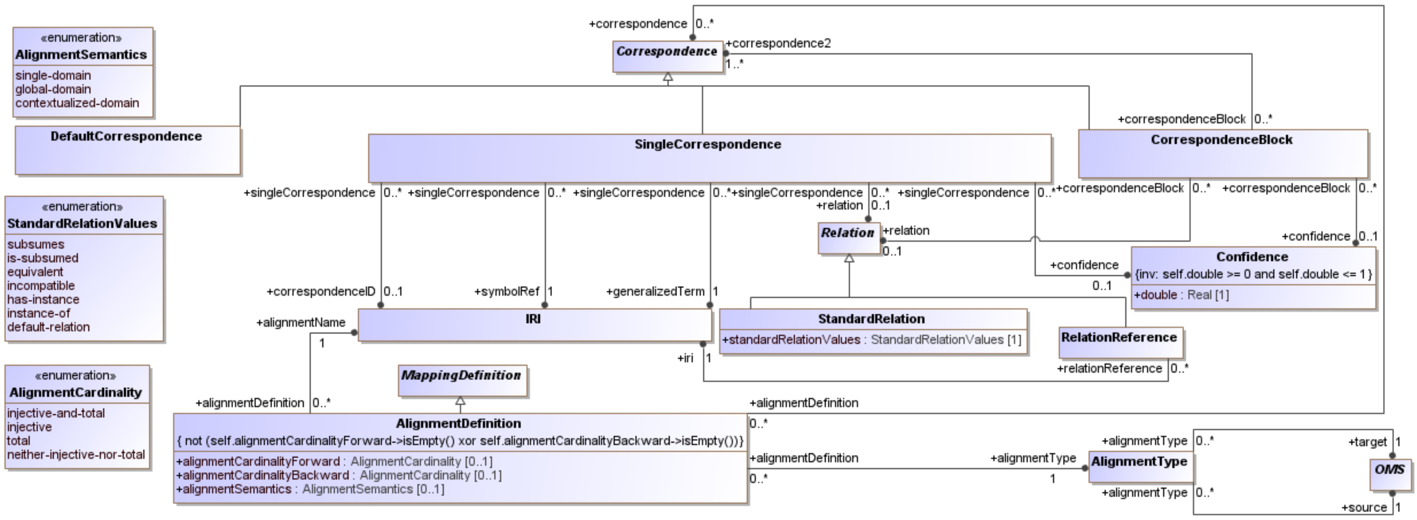


Figure 9.10: DOL metamodel: Alignments

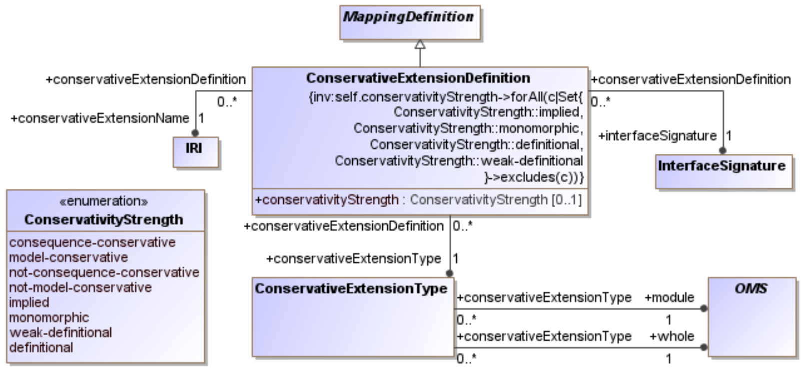


Figure 9.11: DOL metamodel: Conservative extension definitions

```
InterpretationDefinition ::= InlineInterpretation | RefinementDefinition
InlineInterpretation ::= InterpretationKeyword InterpretationName
                         [Conservative] ':' InterpretationType
                         [ '=' LanguageTranslation* [SymbolMap] ]
                         'end'
RefinementDefinition ::= InterpretationKeyword InterpretationName '='
                         Refinement
                         'end'
InterpretationKeyword ::= 'interpretation' | 'view' | 'refinement'
InterpretationName  ::= IRI
InterpretationType  ::= GroupOMS 'to' GroupOMS
Refinement          ::= GroupOMS
                      | NetworkName
                      | Refinement 'refined' [RefMap] 'to' Refinement
                      | Refinement 'interpreted' [RefMap] 'by' Refinement
RefMap              ::= 'via' ( OMSRefinementMap | NetworkRefinementMap )
OMSRefinementMap    ::= LanguageTranslation [SymbolMap]
                      | [LanguageTranslation] SymbolMap
NetworkRefinementMap ::= Refinement ( ',' Refinement )*
EntailmentDefinition ::= 'entailment' EntailmentName '='
                         EntailmentType 'end'
```

```
EntailmentName     ::= IRI
EntailmentType     ::= OMSOMSEntailment
                     | NetworkOMSEntailment
                     | NetworkNetworkEntailment
OMSOMSEntailment         ::= GroupOMS 'entails' GroupOMS
NetworkOMSEntailment     ::= OMSName 'in' Network 'entails' GroupOMS
NetworkNetworkEntailment ::= Network 'entails' Network
EquivalenceDefinition    ::= 'equivalence' EquivalenceName ':'
                             EquivalenceType 'end'
EquivalenceName    ::= IRI
EquivalenceType    ::= OMSEquivalence | NetworkEquivalence
OMSEquivlence      ::= GroupOMS '<->' GroupOMS ['=' OMS]
NetworkEquivalence ::= Network '<->' Network ['=' Network]
ConservativeExtensionDefinition ::= 'cons-ext' ConservativeExtensionName [Conservative] ':'
                        ConservativeExtensionType 'for' InterfaceSignature
ConservativeExtensionName       ::= IRI
ConservativeExtensionType       ::= GroupOMS 'of' GroupOMS
AlignmentDefinition ::= 'alignment' AlignmentName
                        [AlignmentCardinality AlignmentCardinality] ':'
                        AlignmentType
                        ['=' Correspondence ( ',' Correspondence )*]
                        ['assuming' AlignmentSemantics] 'end'
AlignmentName      ::= IRI
AlignmentCardinality ::= '1' | '?' | '+' | '*'
AlignmentType      ::= GroupOMS 'to' GroupOMS
AlignmentSemantics ::= 'SingleDomain'
                     | 'GlobalDomain'
                     | 'ContextualizedDomain'
Correspondence     ::= CorrespondenceBlock | SingleCorrespondence | DefaultCorrespondence
DefaultCorrespondence ::= '*'
CorrespondenceBlock ::= 'relation' [Relation] [Confidence] '{'
                        Correspondence ( ',' Correspondence )* '}'
SingleCorrespondence ::= Symbol [Relation] [Confidence]
                         GeneralizedTerm [CorrespondenceId]
CorrespondenceId   ::= '%(' IRI ')%'
Symbol          ::= IRI
GeneralizedTerm    ::= Symbol
Relation           ::= RelationReference | StandardRelation
RelationReference  ::= IRI
StandardRelation   ::= '>' | '<' | '=' | '%' | 'ni' | 'in'
                       < No keyword corresponding to default-relation as this is just the default if
                         Relation is omitted >
Confidence         ::= Double
Double             ::= < a number ∈ [0,1] >
```

9.7 Identifiers

This section specifies the abstract syntax of identifiers of DOL OMS and their elements. Further, it introduces the concrete syntax that is used in the DOL serialization.

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

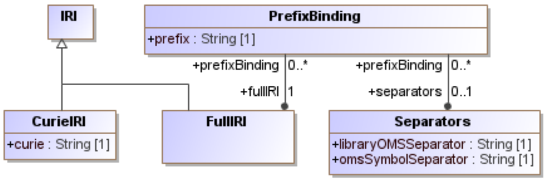


Figure 9.12: DOL metamodel: Prefixes

are documented in [27]. The latter is a specialization of the linked data principles, which apply to any machine-processable data published on the Web [42].) 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 **NR11**. It is **recommended** that DOL libraries use IRIs that translate to URLs when applying the algorithm for mapping IRIs to URIs specified in **NR11**, Section 3.1. DOL descriptions of any element of a DOL 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 **NR11**, DOL adopts the latter into its abstract syntax as well as all of its concrete syntaxes (serializations). The DOL metamodel for IRIs and prefixes is shown in Figure 9.12.

In accordance with semantic web best practices such as the OWL Manchester Syntax [23], 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 [27]:

namespace: an IRI that identifies an OMS, usually ending with `#` or `/`. (See annex O for a specific linked-data compliant URL scheme for DOL.)

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

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. In the following example that uses DOL prefix map mechanism, one the prefix `lang` is bound to `http://purl.net/DOL/languages/`, which means that the CURIE `lang:OWL2` will be expanded to the IRI `http://purl.net/DOL/languages/OWL2`.

```
%prefix( :      <http://www.example.org/mereology#>
         owl:   <http://www.w3.org/2002/07/owl#>
         lang:  <http://purl.net/DOL/languages/>
                %% definitions of conforming languages ...
```

```
        ser:   <http://purl.net/DOL/serializations/>
               %% ... and their serializations
        log:   <http://purl.net/DOL/logics/>
               %% descriptions of logics ...
        trans: <http://purl.net/DOL/translations/> )%
               %% ... and translations

library Mereology

%% OWL Manchester syntax declaration:
language lang:OWL2 logic log:SROIQ serialization ser:OWL2/Manchester
[...]
```

DOL adopts the CURIE specification of RDFa Core 1.1 **NR16**, Section 6 with the following changes:

- 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 DOL library IRI (in the context of a structured OMS where named OMS are referenced) respectively the current OMS IRI (in the context of a basic OMS) to a symbol name. Both the separator between the DOL 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.

CURIES are not required as 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.

CURIEs can occur in any place where IRIs are allowed, as stated in clause 9.7.1.

The CURIE grammar of DOL is summarized in clause 9.7.4.

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 2.3.

Prefix mappings can be defined at the beginning of a DOL library (specified in clause 9.3; these apply to all parts of the DOL 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 takes precedence.

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 **NR2**, Section 5.5 does use IRIs.

- Common Logic **NR7** supports them but does not enforce their use.

— F-logic [33] 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 (`Symbol`). 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 DOL 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 2.3) **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 **NR3** 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 DOL library (`BasicOMS`), not to OMS in external documents (`NativeDocument`); the latter **shall** be self-contained.

While CURIEs used for identifying parts of a DOL 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 DOL library.

9.7.4 Concrete Syntax

```
IRI              ::= '<' FullIRI '>' | CURIE
FullIRI          ::= < an IRI as defined in NR11 >
CURIE            ::= MaybeEmptyCURIE -
MaybeEmptyCURIE ::= [Prefix] RefWithoutComma
RefWithoutComma ::= Reference - StringWithComma
StringWithComma ::= UChar* ',' UChar*
UChar            ::= < any Unicode NR17 character >
Prefix           ::= NCName ':'< see "NCName" in NR15, Section 3 >
Reference        ::= Path [Query] [Fragment]
Path             ::= ipath-absolute | ipath-rootless | ipath-empty < as defined in NR11 >
Query            ::= '?' iquery < as defined in NR11 >
Fragment         ::= '#' ifragment < as defined in NR11 >
```

In a CURIE without a prefix, the `reference` part is **not allowed** to match any of the keywords of the DOL syntax (cf. clause 9.8.1).

9.8 Lexical Symbols

The character set for the DOL text serialization is the UTF-8 encoding of Unicode **NR17**. However, OMS can always be input in the Basic Latin subset, also known as US-ASCII.[19] For enhanced readability of OMS, the DOL text serialization particularly supports the native Unicode glyphs that represent common mathematical symbols (e.g., Greek letters) or operators (e.g., ∂ for partial derivatives).

9.8.1 Keywords and signs

The lexical symbols of the DOL text serialization include various keywords and signs that occur as terminal symbols in the context-free grammar in annex K. Keywords 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.

9.8.1.1 Keywords

Keywords are always written lowercase. The following keywords are reserved, and are not available for use as variables or as CURIEs with no prefix[20], although they can be used as parts of tokens.

[19] In this case, IRIs will have to be mapped to URIs following section 3.1 of **NR11**.

[20] In such a case, one can still rename affected variables, or declare a prefix binding for affected CURIEs, or use absolute IRIs instead. These rewritings do not change the semantics.

```
alignment
along
assuming
and
closed-world
cofree
combine
cons-ext
end
entails
entailment
equivalence
excluding
extract
free
hide
import
in
for
forget
interpretation
keep
language
library
logic
maximize
model
minimize
network
ni
of
oms
onto
ontology
refined
refinement
reject
relation
remove
result
reveal
select
separators
serialization
spec
specification
substitution
then
to
translation
using
vars
via
view
where
with
%cons
%ccons
%complete
%consistent
%def
%implied
%inconsistent
%mcons
%mono
%notccons
%notmcons
%prefix
%wdef
```

9.8.1.2 Key signs

Table 2 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.

Table 2: 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	
$\mapsto$	U+21A6 RIGHTWARDS ARROW FROM BAR	|->
$\rightarrow$	U+2192 RIGHTWARDS ARROW	->

9.9 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 2.3:

XML conformance: 1) If the serialization supports annotation of the root element of the fragment of interest, as specified in XML conformance requirement 3a, an identifier is assigned by way of an annotation whose predicate is http://www.w3.org/2002/07/owl#sameAs from the OWL language **NR5** and whose object is expected to be the desired IRI identifier.

2) If the *dol:id* XML attribute from the http://www.omg.org/spec/DOL/1.0/xml namespace is supported on the element, as specified in XML conformance requirement 3b, its value is expected to be the IRI identifier.

3) If the `dol:id` XML element is supported as the first child of the element, as specified in XML conformance requirement 3b, it is expected to contain exactly one text node whose value is the IRI identifier.

It is a DOL syntax error if 1) an `owl:sameAs` annotation or a `dol:id` attribute or child element is present and its value is not, or cannot be interpreted, as a full IRI, or if 2) more than one of these three alternative fields (annotation, attribute or child element) is present on an element.

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[21)] 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.

Standoff markup conformance: If the given OMS serialization conforms with the *text/plain* media type as per standoff markup conformance requirement 1 but not with XML, **NR12** shall be used as means of non-destructively assigning a URI to pieces of text in the given OMS serialization. If the serialization conforms with XML as per requirement 2, one of **NR12** or XPointer (**NR13**) shall be used. (As an example, consider the identification of imports in the OWL/XML serialization [25], which does not provide a native way for assigning identifiers to imports unless modified as suggested in annex C.3.2. For example, in an OMS file *cars.owx*, the import `<Import>http://example.org/engines</Import>` can be referred to by the IRI `cars.owx#xpointer(/owl:Ontology/owl:Import[text()='http://example.org/engines'])` assuming the right binding for the namespace prefix *owl* in scope. The same import in the text-based OWL Manchester syntax [23] could be referred to as `cars.omn#line=27` according to **NR12** if it is on line 27 of the document.)

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 document may employ RDF annotations that use XPointer or **NR12** as means of non-destructively referencing pieces of XML or text by IRI, as specified above. (The extensibility of the XPointer framework may be utilized by developing additional XPointer schemes, e.g., for pointing to subterms of sentences in the XCL serialization of Common Logic.)

21) The serialization **may** allow whitespace between the keyword and the comment.

10 DOL Semantics

10.1 General

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, satisfiability modulo theories (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 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 clauses 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 a 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 DOL libraries, OMS networks, OMS, and OMS mappings. For each syntactic 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.

10.2 Theoretical Foundations of the DOL Semantics

In the following the theoretical foundations of the semantics of DOL are specified. The notions of *institution* and institution *comorphism* and *morphism* are introduced, which provide formalizations of the terms logic, logic translation and logic reduction, respectively.

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. This specification follows 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 models) can be equipped with one.

An abstract notion of logical system is given by the notion of satisfaction system [6], called 'rooms' in the terminology of [19]. They capture the Tarskian notion of satisfaction of a sentence in a realization 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* ***realizations****, and*

— *a binary relation* $\models \subseteq \mathcal{M} \times Sen$*, called the* ***satisfaction relation****.* $\square$

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, morphisms between realizations are also needed in order to give a semantics to `minimize`, `maximize`, `free` and `cofree`—these constructs use realization morphisms to select certain realizations, e.g., the minimal 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 *Let $\mathbb{S}et$ be the category having all small sets as objects and functions as arrows, and let $\mathbb{C}at$ be the category of categories and functors.*[22] *An **institution** [18] is a quadruple $I = (\mathsf{Sig}, \mathbf{Sen}, \mathbf{Real}, \models)$ consisting of the following:*

- *a category*[23] Sig *of* signatures *and* signature morphisms*,*
- *a functor $\mathbf{Sen}\colon \mathsf{Sig} \longrightarrow \mathbb{S}et$ 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 $\mathbf{Real} : \mathsf{Sig}^{op} \to \mathbb{C}at$ giving, for each signature Σ, the category of* realizations[24] $\mathbf{Real}(\Sigma)$*, and for each signature morphism $\sigma\colon \Sigma \longrightarrow \Sigma'$, the* reduct functor $\mathbf{Real}(\sigma) : \mathbf{Real}(\Sigma') \to \mathbf{Real}(\Sigma)$*, where often $\mathbf{Real}(\sigma)(M')$ 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 $\models_\Sigma \subseteq |\mathbf{Real}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ for each $\Sigma \in |\mathsf{Sig}|$,*

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

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

for each $M' \in |\mathbf{Real}(\Sigma')|$ 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-realization is a function $M : \Sigma \to \{True, False\}$, and the reduct of a Σ_2-realization M_2 along a signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ is the Σ_1-realization 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-realization 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 $UD^* = \{\langle x_1, \ldots, x_n\rangle \mid x_1, \ldots, x_n \in UD\}$ (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 $t_1 \ldots t_n$,

[22] 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.

[23] See [1][47] for an introduction into category theory.

[24] To avoid confusion with models in the sense of model-driven engineering, we use 'realization' instead of the commonly used term 'model' in logic.

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 [30].

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.*[25] *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 the reduct of a realization is generally given by forgetting some of its parts.

For the rest of the section, an arbitrary institution is considered.

Definition 5 (Theory) *A **theory** is a pair (Σ, Δ) where Σ is a signature and Δ is a set of Σ-sentences.*

Given a theory $T = (\Sigma, \Delta)$, the class of T-realization is the class of all Σ-realizations M such that $M \models \delta$, for each sentence $\delta \in \Delta$. A theory (Σ, Δ) is **consistent** if at least one (Σ, Δ)-realization exists. **Semantic entailment** is defined as usual: for a theory (Σ, Δ) and $\varphi \in \mathbf{Sen}(\Sigma)$, Δ entails φ, written $\Delta \models \varphi$, if all realizations satisfying all sentences in Δ also satisfy φ. For a theory (Σ, Δ), we write $\Delta^\bullet$ for the set of all Σ-sentences φ such that $\Delta \models \varphi$.

Definition 6 (Theory morphism) *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 7 (Institution Comorphism) *An **institution comorphism** from an institution $I = (\mathsf{Sig}^I, \mathsf{Real}^I, \mathsf{Sen}^I, \models^I)$ to an institution $J = (\mathsf{Sig}^J, \mathsf{Real}^J, \mathsf{Sen}^J, \models^J)$ consists of a functor $\Phi : \mathsf{Sig}^I \longrightarrow \mathsf{Sig}^J$, and two natural transformations $\beta : \mathsf{Real}^J \circ \Phi^{op} \Longrightarrow \mathsf{Real}^I$ 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 realization $M' \in |\mathsf{Real}^J(\Phi(\Sigma))|$*

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

*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 $\Phi(\Sigma)$-realization M' to a Σ-realization. Naturality of α and β means that for each signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ in I the following squares commute:

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

A comorphism is:

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

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

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

— *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

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

admits (weak) amalgamation, i.e., any for any two realizations $M_2 \in |\mathsf{Real}^I(\Sigma_2)|$ and $M'_1 \in |\mathsf{Real}^J(\Phi(\Sigma_1))|$ with $M_2|_\sigma = \beta_{\Sigma_1}(M'_1)$, there is a unique (not necessarily unique) $M'_2 \in |\mathsf{Real}^J(\Phi(\Sigma_2))|$ 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[26)];

— 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 OMS translation and OMS reduction.

Definition 8 *Given an institution $I = (\mathsf{Sig}^I, \mathsf{Real}^I, \mathsf{Sen}^I, \models^I)$, the institution of its theories, denoted I^{th}, can be defined as follows. The category of signatures of I^{th} is the category of I-theories and I-theory morphisms, denoted Th^I. For each theory (Σ, Δ), its sentences are just Σ-sentences in I, and its realizations are just Σ-realizations in I that satisfy the sentences in Δ, while the (Σ, Δ)-satisfaction is the Σ-satisfaction of sentences in realizations of I.* □

Using this notion, logic translations can be defined that include axiomatization of parts of the syntax of the source logic into the target logic.

Definition 9 *Let $I = (\mathsf{Sig}^I, \mathsf{Real}^I, \mathsf{Sen}^I, \models^I)$ and $J = (\mathsf{Sig}^J, \mathsf{Real}^J, \mathsf{Sen}^J, \models^J)$ be two institutions. A* ***theoroidal institution comorphism*** *from I to J is a institution comorphism from I to J^{th}.* □

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

Definition 10 (Institution Morphism) *An* ***institution morphism*** *from an institution $I = (\mathsf{Sig}^I, \mathsf{Real}^I, \mathsf{Sen}^I, \models^I)$ to an institution $J = (\mathsf{Sig}^J, \mathsf{Real}^J, \mathsf{Sen}^J, \models^J)$ consists of a functor $\Phi : \mathsf{Sig}^I \longrightarrow \mathsf{Sig}^J$, and two natural transformations $\beta : \mathsf{Real}^I \Longrightarrow \mathsf{Real}^J \circ \Phi^{op}$ 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 realization $M \in \mathsf{Real}^I(\Sigma)$*

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

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 [18]. For a formal mathematical definition, see P.1.1.

A major property of colimits of specifications is *amalgamation* (also related to 'exactness' [14]). It can be intuitively explained as stating that realizations of given specifications can be combined to yield a uniquely determined realization of a colimit specification, provided that the original realizations coincide on common components. Amalgamation is a common technical assumption in the study of specification semantics [66].

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

26) An isomorphism if morphisms of realizations are taken into account.

Definition 11 *Given a network $D\colon J \longrightarrow \mathsf{Sig}^I$, a family of realizations $\mathcal{M} = \{M_p\}_{j\in|J|}$ 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* Real. *For realizations, this means that for each D-compatible family of realizations $(M_j)_{j\in|J|}$, there is a Σ-realization M, called an amalgamation of $(M_j)_{j\in|J|}$, with $M|_{\mu_j} = M_j$ $(j \in |J|)$, and similarly for morphisms of realizations. If this realization is unique, the cocone is called* amalgamable. *I (or* Real*) 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.* □

[8] 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, a selection operation is required both for the weakly amalgamable cocone of a network and for the (potentially non-unique) amalgamation of a family of realizations 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 realizations compatible with a network and returning its selected amalgamation.

10.3 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[27)], 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 relation *supports* between OMS languages and institutions, and a binary relation *supports* between OMS languages and serializations. Each language is required to have a default logic and serialization. Moreover, we assume that institutions, institution morphisms and institution comorphisms are uniquely identified by names, and we use the notation $\Gamma(n)$ for the institution, institution morphism and institution comorphism identified by the name n int the heterogeneous logical environment Γ.

We are going to require existence of union and difference operations on the signatures of an institution in the heterogeneous logical environment. These concepts could be captured in a categorical setting using *inclusion systems* [14]. However, inclusion systems are too strong for the purposes of this specification. Therefore, weaker assumptions will be used.

Definition 12 *An* inclusive category *[20] is a category having a broad subcategory[28)] which is a partially ordered class with a least element (denoted $\emptyset$), 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.* □

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

$$\begin{array}{ccc} A & \hookrightarrow & A' \\ \downarrow & & \downarrow \\ B & \hookrightarrow & B' \end{array}$$

for each span where one arrow is an inclusion.

27) 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.

28) That is, with the same objects as the original category.

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

Definition 13 *An institution is* weakly inclusive *if*

— Sig *is inclusive and has pushouts which preserve inclusions,*

— Sen *is inclusive, and*

— *each category of realizations has a broad subcategory of inclusions.* □

Let I be a weakly inclusive institution. *I has differences*, if there is a binary operation $\setminus$ on signatures, such that for each pair of signatures Σ_1, Σ_2, the greatest signature Σ such that

1) $\Sigma \subseteq \Sigma_1$

2) $\Sigma \cap \Sigma_2 = \emptyset$

exists and is equal to $\Sigma_1 \setminus \Sigma_2$.

We will write $\iota_{A \subseteq B}$ for the inclusion of A in B in an inclusive category, when such an inclusion exists. If $\mathcal{I}$ is an inclusive institution and $\Sigma \subseteq \Sigma'$ is an inclusion of signatures, we write $M'|_\Sigma$ for the reduct of a Σ'-realization M' along the inclusion $\iota_{\Sigma \subseteq \Sigma'}$.

To be able to talk about the symbols of a signature in a formal way, it is required that the category of signatures of an institution is an inclusive category with symbols, as defined below:

Definition 14 *An* inclusive category with symbols *is an inclusive category* $\mathbb{C}$ *equipped with a faithful functor* $|\_| : \mathbb{C} \to \mathbb{S}et$[29] *that preserves inclusions.* □

Moreover, if $\sigma : \Sigma \to \Sigma'$ is a signature morphism, it uniquely determines a map $|\sigma| : |\Sigma| \to |\Sigma'|$.

After these preliminaries, we can now list the assumptions made about the institutions in a heterogeneous logical environment. It is required that for each institution in the heterogeneous logical environment there is a trivial signature $\emptyset$ with class of realizations $\mathcal{M}_\emptyset$ and such that there exists a unique signature morphism from $\emptyset$ to any signature of the institution. Moreover, the existence of a partial union operation on institutions is required, denoted $\bigcup$: $L_1 \bigcup L_2 = (L, \rho_1 : L_1 \to L, \rho_2 : L_2 \to L)$, when defined, where L is an institution and ρ_1 and ρ_2 are institution comorphisms, giving the embedding of L_1 and respectively L_2 in L. Finally, some of the comorphisms are marked as default translations and some of the morphisms as default projections, with the condition that between any two institutions at most one comorphism and at most one morphism is marked as default.

For each institution $\mathcal{I}$ in the heterogeneous logical environment, it is further required that there is:

— a function giving the semantics of a basic OMS. It has the format

$$semBasic_{(lang,logic,ser)}(\Sigma, O) = (\Sigma', \Delta')$$

where O is a `BasicOMS`, Σ gives the context of previous declarations, Σ' is the resulting signature and Δ' is the resulting set of sentences. It is required then that $\Sigma \subseteq \Sigma'$.

— a function $makeMorphism_{\mathcal{I}}$ that turns symbol maps into signature morphisms,

— a function $sameName_{\mathcal{I}}$ that takes as arguments two signatures Σ_1 and Σ_2 of $\mathcal{I}$ and returns as result the list of all pairs of symbols (s_i^1, s_i^2) such that $s_i^1 \in |\Sigma_1|$ and $s_i^2 \in |\Sigma_2|$ and the symbols have the same name. The relation represented by $sameName_{\mathcal{I}}(\Sigma_1, \Sigma_2)$ must be an equivalence relation.

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

- a relativization function $relativize_{\mathcal{I}}$ taking as argument a theory and giving as result a theory, and a function $theoryOfCorrespondences$ for translating correspondences of alignments into sentences in the logic according to a given assumption about the semantics of the alignment, both needed in Section 10.3.4.

Further, for each institution, it is required that there exist union and difference operations on signatures.

DOL follows a model-theoretic approach on semantics: the semantics of OMS will be defined as a class of realizations over some signature of an institution. This is called *model-level* semantics. In some cases, but not in all, one 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 realizations given by the model-level semantics is exactly the class of realizations of the theory given by the theory-level semantics.

The following unifying notation is used 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 realizations of O is denoted $\mathsf{Real}(O)$ (which is a class of realizations over $\mathsf{Sig}(O)$),
- the set of axioms of O is denoted $\mathsf{Th}(O)$ (which is a set of sentences over $\mathsf{Sig}(O)$).

Moreover, the semantics of O is the tuple $sem(O) = (I, \Sigma, \mathcal{M}, \Delta)$ where $\mathbf{Inst}(O) = I$, $\mathsf{Sig}(O) = \Sigma$, $\mathsf{Real}(O) = \mathcal{M}$ and $\mathsf{Th}(O) = \Delta$. In the following, we will freely mix these two equivalent descriptions of the semantics. That is, whenever $sem(O)$ is determined in some the context, then also its components $\mathbf{Inst}(O)$, $\mathsf{Sig}(O)$, $\mathsf{Real}(O)$ and $\mathsf{Th}(O)$ are determined. Vice versa, if the four components are determined, then so is $sem(O)$.

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

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
- $\mathsf{Real}(N)$, the class of $\mathsf{Sig}(N)$-realizations 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, $S \to T$)
- theorem links (written using double arrows, $S \Rightarrow T$)

both labeled with heterogeneous signature morphisms between the signatures of the source and target nodes (i.e., an edge from the node S to the node T is labeled with a pair (ρ, σ) where $\rho = (\Phi, \alpha, \beta) : \mathbf{Inst}(S) \to \mathbf{Inst}(T)$ is an institution comorphism and $\sigma : \Phi(\mathsf{Sig}(S)) \to \mathsf{Sig}(T)$ is a signature morphism in $\mathbf{Inst}(T)$). The theory of a node may be undefined, as in the case of OMS, and when it is defined, the class of realizations of that node is the class of realizations of $\mathsf{Th}(N)$. For brevity, the label of a node may be written as a tuple. Further, it is required 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. The intuition is that network provide means of putting together graphs of OMS and OMS mappings and of removing sub-graphs of existing networks.

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, and OMS networks, that is also denoted by Γ, providing access to previous definitions,
- a prefix map, denoted $\Gamma.\mathit{prefix}$, that stores the declared prefixes,
- a triple $\Gamma.current$ that stores the current language, logic and serialization.

If Γ is such a global environment, $\Gamma[\mathtt{IRI} \mapsto \mathcal{S}]$ extends the domain of Γ with `IRI` and the newly added value of `IRI` in Γ 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 $\Gamma_1.\mathit{prefix}$ and $\Gamma_2.\mathit{prefix}$ are disjoint, and then $\Gamma_1 \cup \Gamma_2(\mathtt{IRI}) = \begin{cases} \Gamma_1(\mathtt{IRI}) & \text{if } \mathtt{IRI} \in dom(\Gamma_1) \\ \Gamma_2(\mathtt{IRI}) & \text{if } \mathtt{IRI} \in dom(\Gamma_2) \end{cases}$, $\Gamma_1 \cup \Gamma_2.imports = \Gamma_1.imports \cup \Gamma_2.imports$, $\Gamma_1 \cup \Gamma_2.current = \Gamma_1.current$ and $\Gamma_1 \cup \Gamma_2.\mathit{prefix} = \Gamma_1.\mathit{prefix} \cup \Gamma_2.\mathit{prefix}$. $\Gamma.\{\mathit{prefix} = \mathit{PMap}\}$ represents the global environment that sets the prefix map of Γ to PMap and $\Gamma.\{current = (lang, logic, ser)\}$ is used for updating the current triple of Γ to $(lang, logic, ser)$.

DOL assumes a *language-specific semantics* of native structured OMS, inherited from the OMS language. For a native document D in a language L, logic L' and serialization S, $semNative_{(L,L',S)}(D)$ denotes the language-specific semantics of D.

10.3.1 Semantics of Documents

In this section the semantics of DOL constructs regarding documents and DOL libraries is defined.

$$\boxed{\begin{array}{ll} sem(\mathtt{Document}) & = \Gamma \\ & : LogicalEnvironment \end{array}}$$

A document is either a DOL library, or a native document written in one of the languages supported by the heterogeneous logical environment.

For a `NativeDocument` *nativeDocument*,

$$sem(nativeDocument) = \Gamma''$$

where $\Gamma' = \Gamma_\emptyset.\{current = (lang, logic, ser)\}$, with $lang, logic, ser$ determined from the extension of the file containing the native document,
$postfixLogicIRI(o, l)$ is the string $o?logic = l$,
$l_1 \ldots, l_n$ are the logics supported by $lang$ for some natural number n,
$\Gamma_1 = \Gamma'[postfixLogicIRI(IRI, l_1) \mapsto semNative_{(lang,l_1,ser)}(nativeDocument)]$,
$\Gamma_2 = \Gamma_1[postfixLogicIRI(IRI, l_2) \mapsto semNative_{(lang,l_2,ser)}(nativeDocument)], \ldots$
$\Gamma'' = \Gamma_{n-1}[postfixLogicIRI(IRI, l_n) \mapsto semNative_{(lang,l_n,ser)}(nativeDocument)]$.
Note that if the OMS in the native document does not conform with the logic determined by the extension of the file where the document is stored, $sem(nativeDocument)$ will be undefined.

The rule for `DOLLibrary` is given below.

10.3.1.1 Semantics of libraries

$$\boxed{\begin{array}{ll} \mathit{sem}(\texttt{DOLLibrary}) & = \Gamma \\ & : \mathit{LogicalEnvironment} \end{array}}$$

A DOL library is list of definitions of OMS, OMS mappings and OMS networks, starting with an optional prefix map and a qualification.

For a `DOLLibrary` *dolLibrary*,

$$\mathit{sem}(\mathit{dolLibrary}) = \Gamma'$$

where
$\mathit{sem}(\mathit{dolLibrary}.\texttt{prefixMap}) = PMap$,
$\Gamma_1 = \Gamma_\emptyset.\{\mathit{prefix} = PMap\}$,
$\mathit{sem}(\Gamma_1, \mathit{dolLibrary}.\texttt{qualification}) = \Gamma_2$,
$\mathit{sem}(\Gamma_2, \mathit{dolLibrary}.\texttt{libraryItem}) = \Gamma'$.

Note that *dolLibrary*.`libraryName` is just discarded here. However, this name should be the IRI of the document containing the `Document`. This is known as “linked data compliance”. Tools can issue a warning (not an error), if a `Document` does not follow this practice.

10.3.1.2 Semantics of lists of library items

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, \mathit{Sequence}(\texttt{LibraryItem})) & = \Gamma' \\ & : \mathit{LogicalEnvironment} \end{array}}$$

If $\mathit{libItem}_1, \ldots, \mathit{libItem}_n$ are all `LibraryItems`,

$$\mathit{sem}(\Gamma, \mathit{Sequence}\{\mathit{libItem}_1, \ldots, \mathit{libItem}_n\}) = \Gamma'$$

where
$\mathit{sem}(\Gamma, \mathit{libItem}_1) = \Gamma_1$,
$\mathit{sem}(\Gamma_1, \mathit{libItem}_2) = \Gamma_2, \ldots$
$\mathit{sem}(\Gamma_{n-1}, \mathit{libItem}_n) = \Gamma'$.

10.3.1.3 Semantics of library items

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, \texttt{LibraryItem}) & = \Gamma' \\ & : \mathit{LogicalEnvironment} \end{array}}$$

For a `LibraryImport` *libImport*,

$$\mathit{sem}(\Gamma, \mathit{libImport}) = \Gamma \cup \Gamma'$$

where $\mathit{sem}(\Gamma, \mathit{libImport}.\texttt{libraryName}) = anIRI$ and $\mathit{sem}(anIRI) = \Gamma'$.

A `LibraryItem` can also be an `OMSDefinition`, `NetworkDefinition` or `MappingDefinition`, and equations for these are given in the next sections. (Annex L also introduces `QueryRelatedDefinition`.)

10.3.1.4 Semantics of a list of qualifications

$$\boxed{\begin{array}{ll} sem(\Gamma, Sequence(\texttt{Qualification})) & = \Gamma' \\ & : LogicalEnvironment \end{array}}$$

If $q_1, \ldots, q_n$ are all `Qualifications`,

$$sem(\Gamma, Sequence(q_1, \ldots, q_n)) = \Gamma'$$

where $sem(\Gamma, q_1) = \Gamma_1$, $sem(\Gamma_1, q_2) = \Gamma_2$, ..., $sem(\Gamma_{n-1}, q_n) = \Gamma'$.

10.3.1.5 Semantics of qualifications

$$\boxed{\begin{array}{ll} sem(\Gamma, \texttt{Qualification}) & = \Gamma' \\ & : LogicalEnvironment \end{array}}$$

For a `LanguageQualification` q,

$$sem(\Gamma, q) = \Gamma'$$

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

For a `LogicQualification` q,

$$sem(\Gamma, q) = \Gamma'$$

where $\Gamma' = \Gamma.\{current = (lang', q.\texttt{logicRef}, ser')\}$
$lang = lang(\Gamma.current)$, $ser = ser(\Gamma.current)$
$lang' = \begin{cases} lang, & \text{if } lang \text{ supports } q.\texttt{logicRef} \\ \text{the unique language supporting } q.\texttt{logicRef}, & \text{otherwise} \end{cases}$
$ser' = \begin{cases} ser, & \text{if } lang' \text{ supports } ser \\ \text{the default serialization for } lang', & \text{otherwise} \end{cases}$
Note that "the unique language supporting q.`logicRef`" may be undefined; in this case, the semantics of q construct is undefined.

For a `SyntaxQualification` q,

$$sem(\Gamma, q) = \Gamma'$$

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

10.3.2 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. By convention, 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.

The following auxiliary functions are used:

— $insert(G, \Gamma, iri, id)$, where G is a graph, Γ is a global environment, iri is an `IRI` and id is an `Id`, defined as follows:

  — if iri denotes an OMS in Γ, then a new node named iri and labeled with $\Gamma(iri)$ 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, the result is the union of G with the graph of $\Gamma(iri)$.

— $removeElement(\Gamma, G, anIRI)$, where G is a graph, Γ is a global environment and $anIRI$ is an `IRI`, defined as follows:

  — if $anIRI$ denotes an OMS in Γ, then the node labeled with $anIRI$ and all its incoming and outgoing edges are removed from G,

  — if $anIRI$ denotes an OMS mapping in Γ, then $\Gamma(anIRI)$ 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 $anIRI$ is a network in Γ, then all the nodes of its graph and all their incoming and outgoing edges are removed from G.

— $removePaths(\Gamma, G, iri_1, iri_2)$, where G is a graph, Γ is a global environment and iri_1, iri_2 are `IRI`s, whose result is that all paths of imports in G between the nodes labeled with iri_1 and iri_2 are removed from G.

Finally, the operation $addImports(\Gamma, G, [iri_1, \ldots, iri_n])$ adds to G all import edges in $\Gamma.imports$ between nodes which appear in the subgraph determined by $\Gamma(iri_1), \ldots, \Gamma(iri_n)$.

10.3.2.1 Semantics of network definitions

$$\boxed{\begin{array}{ll} sem(\Gamma, \texttt{NetworkDefinition}) & = \Gamma' \\ & : LogicalEnvironment \end{array}}$$

If n is a `NetworkDefinition`,

$$sem(\Gamma, n) = \Gamma'$$

where $\Gamma' = \Gamma[n.\texttt{networkName} \mapsto sem(\Gamma, n.\texttt{network})]$.

If $n.$`ConservativityStrength` is `model-conservative`, the semantics is only defined if the class of families of realizations compatible with the graph $sem(\Gamma, n.\texttt{network})$ is not empty.

If $n.$`ConservativityStrength` 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 $n.$`ConservativityStrength` is `monomorphic`, the semantics is only defined if the class of families of realizations compatible with the graph $sem(\Gamma, n.\texttt{network})$ consist of exactly one isomorphism class of families of realizations.

If n.ConservativityStrength is weak-definitional, the semantics is only defined if the class of families of realizations compatible with the graph $sem(\Gamma, n.\texttt{network})$ is at most a singleton.

If n.ConservativityStrength is definitional, the semantics is only defined if the class of families of realizations compatible with the graph $sem(\Gamma, n.\texttt{network})$ is a singleton.

If n.ConservativityStrength is not-model-conservative, the semantics is only defined if the class of families of realizations compatible with the graph $sem(\Gamma, n.\texttt{network})$ is the empty set.

If n.ConservativityStrength 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.

10.3.2.2 Semantics of networks

$$\boxed{\begin{aligned} sem(\Gamma, \texttt{Network}) \quad &= G \\ &: OMSGraph \end{aligned}}$$

If n is a network,

$$sem(\Gamma, n) = G'$$

where $sem(\Gamma, n.\texttt{networkElement}) = G$ and $sem(\Gamma, G, n.\texttt{excludedElement}) = G'$.

10.3.2.3 Semantics of sets of network elements

$$\boxed{\begin{aligned} sem(\Gamma, Set(\texttt{NetworkElement})) \quad &= G \\ &: OMSGraph \end{aligned}}$$

If $elem_1, \ldots, elem_n$ are all NetworkElements,

$$sem(\Gamma, Set\{elem_1, \ldots, elem_n\}) = G$$

where
$G_1 = sem(\Gamma, G_\emptyset, elem_1)$, where $G_\emptyset$ is the empty graph,
$G_2 = sem(\Gamma, G_1, elem_2)$
...
$G_n = sem(\Gamma, G_{n-1}, elem_n)$,
$G = addImports(\Gamma, G_n, [elem_1, \ldots, elem_n])$.

10.3.2.4 Semantics of network elements

$$\boxed{\begin{aligned} sem(\Gamma, G, \texttt{NetworkElement}) \quad &= G' \\ &: OMSGraph \end{aligned}}$$

If $networkElement$ is a NetworkElement,

$$sem(\Gamma, G, networkElement) = insert(G, \Gamma, networkElement.\texttt{elementRef.iri}, networkElement.\texttt{id})$$

10.3.2.5 Semantics of sets of excluded elements

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, G, \mathit{Set}(\texttt{ExcludedElement})) & = G' \\ & : \mathit{OMSGraph} \end{array}}$$

If $\mathit{elem}_1, \ldots, \mathit{elem}_n$ are all `ExcludedElements`,

$$\mathit{sem}(\Gamma, G, \mathit{Set}\{\mathit{elem}_1, \ldots, \mathit{elem}_n\}) = G'$$

where
$G_1 = \mathit{sem}(\Gamma, G, \mathit{elem}_1)$
$G_2 = \mathit{sem}(\Gamma, G_1, \mathit{elem}_2)$
...
$G' = \mathit{sem}(\Gamma, G_{n-1}, \mathit{elem}_n)$

10.3.2.6 Semantics of excluded elements

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, G, \texttt{ExcludedElement}) & = G' \\ & : \mathit{OMSGraph} \end{array}}$$

If $\mathit{excludedElem}$ is a `ElementRef`,

$$\mathit{sem}(\Gamma, G, \mathit{excludedElem}) = \mathit{removeElement}(\Gamma, G, \mathit{excludedElem}.\texttt{iri})$$

.

If $\mathit{excludedElem}$ is a `PathReference`,

$$\mathit{sem}(\Gamma, G, \mathit{excludedElem}) = \mathit{removePaths}(\Gamma, G, \mathit{iri}_1, \mathit{iri}_2)$$

where $\mathit{iri}_1 = \mathit{excludedElem}.\texttt{elementRef.iri}$ and $\mathit{iri}_2 = \mathit{excludedElem}.\texttt{elementRef2.iri}$).

10.3.3 Semantics of OMS

In the rest of this section, given a global environment Γ and an `OMS` O, the notation $\mathit{Env}(\Gamma, O)$ is used for the global environment Γ' such that $\mathit{sem}(\Gamma, O) = (\Gamma', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta))$.

10.3.3.1 Semantics of basic OMS

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, \texttt{BasicOMS}) & = (\Gamma', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)) \\ & : (\mathit{LogicalEnvironment}, (\mathit{Institution}, \mathit{Signature}, \mathit{RealizationClass}, \mathit{Sentences})) \end{array}}$$

For a `BasicOMS` O in a global environment Γ, the semantics is defined as follows:

$$\mathit{sem}(\Gamma, O) = (\Gamma', (\Gamma.\mathit{logic}, \Sigma', \mathcal{M}', \Delta'))$$

where

— $(\Sigma', \Delta') = semBasic_{(\Gamma.lang, \Gamma.logic, \Gamma.ser)}(O)$,

— $\mathcal{M}' = \{M \in \mathsf{Real}(\Sigma') \mid M \models \Delta'\}$

— Γ' is obtained from Γ by adding to $\Gamma.imports$ a new node labeled with the name of O, $\Gamma.logic$,Σ', $\mathcal{M}'$ and Δ'.

10.3.3.2 Semantics of basic OMS in a local environment

$$\begin{array}{|ll|}\hline sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), \texttt{BasicOMS}) & = (\Gamma', (\mathcal{I}', \Sigma', \mathcal{M}', \Delta')) \\ & : (LogicalEnvironment, (Institution, Signature, RealizationClass, Sentences)) \\ \hline\end{array}$$

For a `BasicOMS` O in a global environment Γ and local environment $(\mathcal{I}, \Sigma, \mathcal{M}, \Delta)$, its semantics is defined only if $\Gamma.logic = \mathcal{I}$ as follows:

$$sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), O) = (\Gamma', (\Gamma.logic, \Sigma', \mathcal{M}', \Delta'))$$

where

— $(\Sigma', \Delta') = semBasic_{(\Gamma.lang, \Gamma.logic, \Gamma.ser)}(\Sigma, O)$

— $\mathcal{M}' = \{M \in \mathcal{M} \mid M \models \Delta'\}$

— Γ' is obtained from Γ by adding to $\Gamma.imports$ a new node labeled with the name of O, $\Gamma.logic$,Σ', $\mathcal{M}'$ and Δ'.

10.3.3.3 Semantics of closable OMS

$$\begin{array}{|ll|}\hline sem(\Gamma, \texttt{ClosableOMS}) & = (\Gamma', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)) \\ & : (LogicalEnvironment, (Institution, Signature, RealizationClass, Sentences)) \\ \hline\end{array}$$

The semantics of a `BasicOMS` has been defined above.

The semantics of an `OMSReference` O is given by

$$sem(\Gamma, O) = (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)$$

where $locId = postfixLogicIRI(O.\texttt{omsRef}, name(\Gamma.logic))$ and

— $postfixLogicIRI(o, l)$ is the string $o?logic = l$

— $\mathcal{I} = \mathbf{Inst}(\Gamma(locId))$,

— $\Sigma = \mathsf{Sig}(\Gamma(locId))$

— $\mathcal{M} = \mathsf{Real}(\Gamma(locId))$

— $\Delta = \mathsf{Th}(\Gamma(locId))$

— $Env(\Gamma, O)$ extends the graph of imports $\Gamma.imports$ with a new node for O whose name is either $O.importName$, or, if $O.importName$ is missing, $locId$, and whose other components of the label are as defined in the items above, and with a new edge from the node labeled with $locId$ to O, named $O.\texttt{importName}$ and labeled with the identity on $\mathsf{Sig}(\Gamma(locId))$.

10.3.3.4 Semantics of closable OMS in a local environment

$$\boxed{\begin{array}{l} \mathit{sem}(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), \ \texttt{ClosableOMS}) = (\Gamma', (\mathcal{I}', \Sigma', \mathcal{M}', \Delta')) \\ \qquad : (\mathit{LogicalEnvironment}, (\mathit{Institution}, \mathit{Signature}, \mathit{RealizationClass}, \mathit{Sentences})) \end{array}}$$

The semantics of a `BasicOMS` has been defined above.

The semantics of an `OMSReference` O is defined only if $\mathbf{Inst}(\Gamma(\mathit{locId})) = \mathcal{I}$,
where $\mathit{locId} = \mathit{postfixLogicIRI}(O.\texttt{omsRef}, \mathit{name}(\Gamma.\mathit{logic}))$, as follows:

$$\mathit{sem}(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), O) = (\Gamma', (\mathcal{I}', \Sigma', \mathcal{M}', \Delta'))$$

where

— $\mathcal{I}' = \mathbf{Inst}(\Gamma(\mathit{locId}))$

— $\Sigma' = \mathsf{Sig}(\Gamma(\mathit{locId})) \cup \Sigma$

— $\mathcal{M}' = \{M \in \mathsf{Real}(\Sigma') \mid M|_{\Sigma} \in \mathcal{M} \text{ and } M|_{\mathsf{Sig}(\Gamma(\mathit{locId}))} \in \mathsf{Real}(\Gamma(\mathit{locId}))\}$

— $\Delta' = \iota_{\mathsf{Sig}(\Gamma(\mathit{locId})) \subseteq \Sigma'}(\mathsf{Th}(\Gamma(\mathit{locId}))) \cup \iota_{\Sigma \subseteq \Sigma'}(\Delta)$

— $\mathit{Env}(\Gamma, O)$ extends the graph of imports $\Gamma.\mathit{imports}$ with a new node for O labeled as defined in the items above and with a new edge from the node labeled with locId to O, named $O.\texttt{importName}$ and labeled with the inclusion of Σ in Σ'.

10.3.3.5 Semantics of ExtendingOMS

$$\boxed{\begin{array}{l} \mathit{sem}(\Gamma, \ \texttt{ExtendingOMS}) = (\Gamma', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)) \\ \qquad : (\mathit{LogicalEnvironment}, (\mathit{Institution}, \mathit{Signature}, \mathit{RealizationClass}, \mathit{Sentences})) \end{array}}$$

The semantics for `ClosableOMS` has been defined above.

If O is a `RelativeClosureOMS`, $O.\texttt{closureType} = \texttt{minimize}$ and $O' = O.\texttt{closableOMS}$, then

$$\mathit{sem}(\Gamma, O) = (\Gamma', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta))$$

where

— $\mathcal{I} = \mathbf{Inst}(O')$

— $\Sigma = \mathsf{Sig}(O')$

— $\mathcal{M} = \{M \in \mathsf{Real}(O') \mid M \text{ is minimal in } \mathsf{Real}(O')\}$ and "minimal" is interpreted in the pre-order defined by $M_1 \leq M_2$ if there is a homomorphism of realizations $\mathcal{M}_1 \to \mathcal{M}_2$.

— $\Delta = \perp$

— Γ' is obtained from $\Gamma'' = \mathit{Env}(\Gamma, O')$ by adding to $\Gamma''.\mathit{imports}$ a new node labeled with $(\mathit{Name}(O), \mathbf{Inst}(O),$ $\mathsf{Sig}(O), \mathsf{Real}(O), \mathsf{Th}(O))$ 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 O is defined similarly for the other three alternatives of $O.\texttt{closureType}$, only the class of realizations differs:

— if $O.\texttt{closureType} = \texttt{maximize}$, $\mathcal{M} = \{M \in \mathsf{Real}(O') \mid M$ is maximal in $\mathsf{Real}(O')\}$

— if $O.\texttt{closureType} = \texttt{free}$, $\mathcal{M} = \{M \in \mathsf{Real}(O') \mid M$ is initial in $\mathsf{Real}(O')\}$

— if $O.\texttt{closureType} = \texttt{cofree}$, $\mathcal{M} = \{M \in \mathsf{Real}(O') \mid M$ is terminal in $\mathsf{Real}(O')\}$

Here, initial and terminal realizations are defined as in category theory, see P.1.1.

10.3.3.6 Semantics of ExtendingOMS in a local environment

$$\boxed{\begin{array}{ll} sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), & \texttt{ExtendingOMS})) = (\Gamma', (\mathcal{I}', \Sigma', \mathcal{M}', \Delta')) \\ & : (LogicalEnvironment, (Institution, Signature, RealizationClass, Sentences)) \end{array}}$$

The semantics for `ClosableOMS` has been defined above.

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

Formally, if O is a `RelativeClosureOMS`, $O.\texttt{closureType} = \texttt{minimize}$ and $O' = O.\texttt{closableOMS}$, and $sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), O') = (\Gamma', (\mathcal{I}', \Sigma', \mathcal{M}', \Delta'))$ then

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

where

— $\mathcal{I}'' = \mathcal{I}'$

— $\Sigma'' = \Sigma'$

— $\mathcal{M}'' = \{M \in \mathcal{M} \mid M$ is minimal in $\{M' \in \mathcal{M} \mid M'|_\Sigma = M|_\Sigma\}\}$ and "minimal" is interpreted in the pre-order defined by $M_1 \leq M_2$ if there is a homomorphism of realizations $\mathcal{M}_1 \to \mathcal{M}_2$

— $\Delta'' = \bot$

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

The theory-level semantics for O cannot be defined.

The semantics of O is defined similarly for the other three alternatives of $O.\texttt{closureType}$, only the class of realizations differs:

— if $O.\texttt{closureType} = \texttt{maximize}$, $\mathcal{M}'' = \{M \in \mathcal{M} \mid M$ is maximal in $\{M' \in \mathcal{M} \mid M'|_\Sigma = M|_\Sigma\}\}$

— if $O.\texttt{closureType} = \texttt{free}$, $\mathcal{M}'' = \{M \in \mathcal{M} \mid M$ is initial in $\{M' \in \mathcal{M} \mid M'|_\Sigma = M|_\Sigma\}\}$

— if $O.\texttt{closureType} = \texttt{cofree}$, $\mathcal{M}'' = \{M \in \mathcal{M} \mid M$ is terminal in $\{M' \in \mathcal{M} \mid M'|_\Sigma = M|_\Sigma\}\}$

10.3.3.7 Semantics of OMS

$$\boxed{\begin{array}{ll} sem(\Gamma, \texttt{OMS}) & = (\Gamma', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)) \\ & : (LogicalEnvironment, (Institution, Signature, RealizationClass, Sentences)) \end{array}}$$

The semantics for a `ClosableOMS` has been defined above.

The semantics for an `ExtendingOMS` has been defined above.

If o is a `ClosureOMS`,

$$sem(\Gamma, O) = (\Gamma'', (I, \Sigma, \mathcal{M}', \bot))$$

where

$$(\Gamma', (I, \Sigma, \mathcal{M}, \Delta)) = sem(\Gamma, O.\mathtt{oms}), \qquad \Sigma_{closure} = sem(\Gamma', \Sigma, O.\mathtt{closure.circClosure}),$$
$$\Sigma_{var} = sem(\Gamma', \Sigma, O.\mathtt{closure.circVars}), \qquad \Sigma_{fixed} = \Sigma \setminus (\Sigma_{closure} \cup \Sigma_{var})$$

and

— if $O.\mathtt{closure.closureType} = \mathtt{minimize}$, then
$\mathcal{M}' = \{M \in \mathcal{M} \mid M|_{\Sigma_{closure} \cup \Sigma_{fixed}}$ is minimal in $\{M' \in \mathcal{M}|_{\Sigma_{closure} \cup \Sigma_{fixed}} \mid M'|_{\Sigma_{fixed}} = M|_{\Sigma_{fixed}}\}\}$

— if $O.\mathtt{closure.closureType} = \mathtt{maximize}$, then
$\mathcal{M}' = \{M \in \mathcal{M} \mid M|_{\Sigma_{closure} \cup \Sigma_{fixed}}$ is maximal in $\{M' \in \mathcal{M}|_{\Sigma_{closure} \cup \Sigma_{fixed}} \mid M'|_{\Sigma_{fixed}} = M|_{\Sigma_{fixed}}\}\}$

— if $O.\mathtt{closure.closureType} = \mathtt{free}$, then
$\mathcal{M}' = \{M \in \mathcal{M} \mid M|_{\Sigma_{closure} \cup \Sigma_{fixed}}$ is initial in $\{M' \in \mathcal{M}|_{\Sigma_{closure} \cup \Sigma_{fixed}} \mid M'|_{\Sigma_{fixed}} = M|_{\Sigma_{fixed}}\}\}$

— if $O.\mathtt{closure.closureType} = \mathtt{cofree}$, then
$\mathcal{M}' = \{M \in \mathcal{M} \mid M|_{\Sigma_{closure} \cup \Sigma_{fixed}}$ is terminal in $\{M' \in \mathcal{M}|_{\Sigma_{closure} \cup \Sigma_{fixed}} \mid M'|_{\Sigma_{fixed}} = M|_{\Sigma_{fixed}}\}\}$

Γ'' is obtained from $\Gamma' = Env(\Gamma, O.\mathtt{oms})$ 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 $O.\mathtt{oms}$ to the node of O labeled with the identity morphism on Σ.

The semantics of a `TranslationOMS` O is given by

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

where

— $\mathcal{I} = J$,

— $\Sigma = \Sigma'$, when $sem(\Gamma, \mathsf{Sig}(O.\mathtt{oms}), O.\mathtt{omsTranslation}) = ((\Phi, \alpha, \beta) : \mathbf{Inst}(O.\mathtt{oms}) \to J, \sigma : \Phi(\mathsf{Sig}(O.\mathtt{oms})) \to \Sigma')$,

— $\mathcal{M} = \{M \in \mathsf{Real}(\Sigma') \mid \beta_{\mathsf{Sig}(O.\mathtt{oms})}(M|_\sigma) \in \mathsf{Real}(O.\mathtt{oms})\}$

— $\Delta = \{Sen^J(\sigma)(\alpha_{\mathsf{Sig}(O.\mathtt{oms})}(\delta)) \mid \delta \in \mathsf{Th}(O.\mathtt{oms})\}$. It is defined only if $O.\mathtt{oms}$ is flattenable.

— Γ'' is obtained from $\Gamma' = Env(\Gamma, O.\mathtt{oms})$ 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 $O.\mathtt{oms}$ to the node of O labeled with $((\Phi, \alpha, \beta), \sigma)$.

The semantics of a `ReductionOMS` O is

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

where

— $\mathcal{I} = J$,

— $\Sigma = \Sigma'$, when $sem(\Gamma, \mathsf{Sig}(O.\mathtt{oms}), O.\mathtt{reduction}) = ((\Phi, \alpha, \beta) : \mathbf{Inst}(O.\mathtt{oms}) \to J, \sigma : \Sigma' \to \Phi(\mathsf{Sig}(O.\mathtt{oms})))$,

— $\mathcal{M} = \{\beta_\Sigma(M)|_\sigma \mid M \in \mathsf{Real}(O.\mathtt{oms})\}$

— $\Delta = \perp$

— Γ'' is obtained from $\Gamma' = \mathit{Env}(\Gamma, O.\mathtt{oms})$ by extending $\Gamma'.\mathit{imports}$ 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 $O.\mathtt{oms}$ labeled with $((\Phi, \alpha, \beta), \sigma)$.

The semantics of an `ExtractionOMS` O is

$$\mathit{sem}(\Gamma, O) = (\Gamma'', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta))$$

where

— $\mathcal{I} = \mathbf{Inst}(O.\mathtt{oms})$

— $\Sigma = \Sigma'$,

— $\Delta = \Delta'$

— $\mathcal{M}$ is the class of Δ-realizations

— Γ'' is obtained from $\Gamma' = \mathit{Env}(\Gamma, O.\mathtt{oms})$ by extending $\Gamma'.\mathit{imports}$ 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 $O.\mathtt{oms}$ labeled with the inclusion of Σ' in $\mathsf{Sig}(O.\mathtt{oms})$

where $\mathit{sem}(\Gamma, (\mathsf{Sig}(O.\mathtt{oms}), \mathsf{Th}(O.\mathtt{oms})), O.\mathtt{extraction}) = (\Sigma', \Delta')$.

The semantics of an `ApproximationOMS` O is

$$\mathit{sem}(\Gamma, O) = (\Gamma'', (\mathcal{I}'', \Sigma', \mathcal{M}, \Delta))$$

where

— $\mathcal{I}'' = \mathcal{I}'$,

— $\Sigma' = \Phi(\Sigma)$

— $\Delta = \alpha^{-1}_{\mathsf{Sig}(O.\mathtt{oms})}(\mathsf{Th}(O.\mathtt{oms})^\bullet) \cap \mathsf{Sen}^{\mathcal{I}'}(\mathsf{Sig}(O.\mathtt{oms}))$. , i.e., that part of $\mathsf{Th}(O.\mathtt{oms})$ that can be expressed in the smaller signature and logic . In practice, one looks for a finite subset that still is logically equivalent to this set.

— $\mathcal{M}$ is the class of Δ-realizations

— Γ'' is obtained from $\Gamma' = \mathit{Env}(\Gamma, O.\mathtt{oms})$ by extending $\Gamma'.\mathit{imports}$ with a new node for O labeled as in the items above and with a new edge from the node of $O.\mathtt{oms}$ to the node of O labeled with $(\rho, \iota : \Phi(\Sigma) \to \mathsf{Sig}(O.\mathtt{oms}))$

where $(\rho = (\Phi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}', \Sigma) = \mathit{sem}(\Gamma, (\mathbf{Inst}(O.\mathtt{oms}), \mathsf{Sig}(O.\mathtt{oms})), O.\mathtt{approximation})$.

The semantics of a `FilteringOMS` O is defined by case distinction. Let $(\Gamma', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)) = \mathit{sem}(\Gamma, O.\mathtt{oms})$ and $(c, \mathcal{I}', \Sigma', \Delta') = \mathit{sem}(\Gamma', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), O.\mathtt{filtering})$.

If $c = \mathit{keep}$, the semantics of O is given by

$$\mathit{sem}(\Gamma, O) = (\Gamma'', (\mathcal{I}'', \Sigma'', \mathcal{M}'', \Delta''))$$

where

— $\mathcal{I}'' = \mathcal{I}'$

— Σ'' is the smallest signature with $\Sigma' \subseteq \Sigma''$ and $\Delta' \subseteq \mathsf{Sen}(\Sigma'')$. (If this smallest signature does not exist, the semantics is undefined.)

— $\Delta'' = (\Delta \cap \mathsf{Sen}(\Sigma'')) \cup \Delta'$

— $\mathsf{Real}(O)$ is the class of all Δ-realizations

— Γ'' is obtained from Γ' 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 $O.\texttt{oms}$ labeled with the inclusion of Σ'' in Σ.

If $c = remove$, the semantics of O is

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

where

— $\mathcal{I}'' = \mathcal{I}'$

— $\Sigma'' = \Sigma \setminus \Sigma'$

— $\Delta'' = \Delta \cap \mathsf{Sen}(\Sigma'') \setminus \Delta'$

— $\mathcal{M}''$ is the class of all $\mathsf{Th}(O)$-realizations

— Γ'' is obtained from Γ' 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 $O.\texttt{oms}$ labeled with the inclusion of Σ'' in Σ.

The semantics of an `UnionOMS` O is

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

where

— $\mathcal{I}' = \mathcal{I}$ where $\mathbf{Inst}(O_1) \bigcup \mathbf{Inst}(O_2) = (\mathcal{I}, (\Phi_1, \alpha_1, \beta_1) : \mathbf{Inst}(O_1) \to \mathcal{I}, (\Phi_2, \alpha_2, \beta_2) : \mathbf{Inst}(O_2) \to \mathcal{I})$

— $\Sigma' = \Phi_1(\mathsf{Sig}(O_1)) \cup \Phi_2(\mathsf{Sig}(O_2))$

— $\mathcal{M}' = \{M \in \mathsf{Real}^{\mathcal{I}}(\Sigma') \mid \beta_{\Sigma_i}(M|_{\Phi_i(\mathsf{Sig}(O_i))}) \in \mathsf{Real}(O_i), \text{ for } i = 1, 2\}$

— $\Delta' = \alpha_1(\mathsf{Th}(O_1)) \cup \alpha_2(\mathsf{Th}(O_2))$.

— Γ'' is obtained from $\Gamma' = Env(Env(\Gamma, O_1), O_2)$ by extending $\Gamma'.imports$ with a new node for O labeled as in the items above and with edges from the nodes of O_1 and O_2, respectively, to the node of O, labeled for each $i = 1, 2$ with $(\Phi_i, \alpha_i, \beta_i, \iota_i : \Phi_i(O_i) \to \Sigma')$.

where $O_1 = O.\texttt{oms}$ and $O_2 = O.\texttt{oms2}$.

If $O.\texttt{conservativityStrength}$ is present, then O must be a conservative extension of the appropriate strength of O_1.

The semantics of an `ExtensionOMS` O is

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

where

— $\mathcal{I}' = \mathbf{Inst}(O.\texttt{oms}) = \mathbf{Inst}(O.\texttt{extension})$ (which means that the institutions of $O.\texttt{oms}$ and $O.\texttt{extension}$ must be the same)

— $\Sigma' = \mathsf{Sig}(O.\texttt{oms}) \cup \mathsf{Sig}(sem(\Gamma, (\mathbf{Inst}(O.\texttt{oms}), \mathsf{Sig}(O.\texttt{oms}), \mathsf{Real}(O.\texttt{oms}), \mathsf{Th}(O.\texttt{oms})), O.\texttt{extension})$

— $\mathcal{M}' = \{M \in \mathsf{Real}(\Sigma') \mid M|_{\mathsf{Sig}(O.\texttt{oms})} \in \mathsf{Real}(O.\texttt{oms}) \text{ and } M|_{\mathsf{Sig}(O.\texttt{extension})} \in \mathsf{Real}(O.\texttt{extension})\}$

— $\Delta' = \mathsf{Th}(O.\texttt{oms}) \cup \mathsf{Th}(O.\texttt{extension})$

— Γ'' is $Env(Env(\Gamma, O.\texttt{oms}), O.\texttt{extension})$.

The semantics of a `QualifiedOMS` O in the context Γ is the same as the semantics of $O.\texttt{oms}$ in the context Γ' given by the semantics of $O.\texttt{qualification}$ in the context Γ. The change of context is local to $O.\texttt{oms}$, 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'. Formally,

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

where $(\Gamma'', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)) = sem(sem(\Gamma, O.\texttt{qualification}), O.\texttt{oms})$.

The semantics of a `CombinationOMS` O is

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

where

— $\mathcal{I}' = \mathcal{I}$,

— $\Sigma' = \Sigma$, where $(\mathcal{I}, \Sigma, \{\mu_i\}_{i \in |G|})$ is the colimit of the graph $G = sem(\Gamma, O.\texttt{network})$,

— $\Delta' = \cup_{i \in |G|} \mu_i(\mathsf{Th}(O_i))$, where O_i is the OMS label of the node i in G

— $\mathcal{M}' = \{M \in \mathsf{Real}(\Sigma) \mid M|_{\mu_i} \in \mathsf{Real}(O_i), i \in |G|\}$, where O_i is the OMS label of the node i in G.

— Γ'' 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 $i \in |G|$.

10.3.3.8 Semantics of `CircClosure`

$$\boxed{\begin{aligned} sem(\Gamma, \Sigma, \texttt{CircClosure}) &= \Sigma' \\ &: Signature \end{aligned}}$$

If c is a `CircClosure`,

$$sem(\Gamma, \Sigma, c) = sem(\Gamma, \Sigma, c.symbol)$$

10.3.3.9 Semantics of `CircVar`

$$\boxed{\begin{aligned} sem(\Gamma, \Sigma, \texttt{CircVar}) &= \Sigma' \\ &: Signature \end{aligned}}$$

If c is a `CircVar`,

$$sem(\Gamma, \Sigma, c) = sem(\Gamma, \Sigma, c.symbol)$$

10.3.3.10 Semantics of OMS translations

$$\boxed{\begin{array}{ll} sem(\Gamma, \Sigma, \texttt{OMSTranslation}) & = (\rho, \sigma) \\ & : (Comorphism, SignatureMorphism) \end{array}}$$

The semantics of a `OMSTranslation` $O =$ is given by

— $\rho = sem(O.\texttt{omsLanguageTranslation}) : \Gamma.logic \to logic'$

— $\sigma = sem(\Gamma.\{current = (lang', logic', ser')\}, \Phi(\Sigma), O.\texttt{symbolMap})$

where $lang'$ and ser' are the default language and serialization for logic $logic'$. If $O.\texttt{omsLanguageTranslation}$ is missing, it defaults to the identity comorphism of the current logic.

10.3.3.11 Semantics of OMS language translations

$$\boxed{\begin{array}{ll} sem(\Gamma, \texttt{OMSLanguageTranslation}) & = \rho \\ & : Translation \end{array}}$$

If t is a `NamedLanguageTranslation`,

$$sem(\Gamma, t) = \Gamma(O.\texttt{omsLanguageTranslationRef})$$

where $\Gamma(O.\texttt{omsLanguageTranslationRef})$ is an institution comorphism. This is defined only if the domain of ρ is the current logic of Γ.

If t is a `DefaultTranslation`,

$$sem(\Gamma, t) = \rho$$

where ρ is the unique default institution comorphism of the heterogeneous logical environment running from $\Gamma.logic$ to $t.\texttt{languageRef}$ (if this is a logic) or to some logic supported by $t.\texttt{languageRef}$ (if this is a language). If there is no or no unique such comorphism, the semantics is undefined.

$$\boxed{\begin{array}{ll} sem(\Gamma, Sequence(\texttt{OMSLanguageTranslation})) & = \rho \\ & : Translation \end{array}}$$

If $t_1, \ldots, t_n$ are all `OMSLanguageTranslations`, $sem(\Gamma, Sequence\{t_1, \ldots, t_n\}) = \rho$, where $sem(\Gamma, t_i) = \rho_i$ for $i = 1, \ldots, n$ and $(\Phi, \alpha, \beta) = \rho_1; \rho_2; \ldots; \rho_n$.

10.3.3.12 Semantics of reductions

$$\boxed{\begin{array}{ll} sem(\Gamma, \Sigma, \texttt{Reduction}) & = (\rho, \sigma) \\ & : (Morphism, SignatureMorphism) \end{array}}$$

The semantics of a `Reduction` $O =$ with $O.\texttt{reduction.removalKind} = remove$ is given by

— $\rho = sem(O.\texttt{reduction.omsLanguageTranslation}) : \Gamma.logic \to logic'$

— $\sigma = \iota : \Sigma' \to \Phi(\Sigma)$, where $\Sigma' = sem(\Gamma.\{current = (lang', logic', ser')\}, \Phi(\Sigma), O.\texttt{reduction.symbolList})$, $lang'$ and ser' are the default language and serialization for logic $logic'$ and ι is the inclusion morphism.

If $O.\texttt{reduction.omsLanguageTranslation}$ is missing, it defaults to the identity morphism of the current logic of Γ.

The semantics of a reduction $O =$ with $O.\texttt{reduction.removalKind} = \mathit{keep}$ is

— ρ is the identity morphism on the current logic of Γ

— σ is the inclusion of $\mathit{sem}(\Gamma, \Sigma, O.\texttt{reduction.symbolList})$ in Σ.

10.3.3.13 Semantics of sets of symbols

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, \Sigma, \mathit{Set}(\texttt{Symbol})) & = \Sigma' \\ & : \mathit{Signature} \end{array}}$$

If $s_1, \ldots, s_n$ are all $\texttt{Symbol}$s,

$$\mathit{sem}(\Gamma, \Sigma, \mathit{Set}\{s_1, \ldots, s_n\}) = \Sigma'$$

where Σ' is the smallest sub-signature of Σ containing $\mathit{sem}(\Gamma, \Sigma, s_1), \ldots, \mathit{sem}(\Gamma, \Sigma, s_n)$, if such a sub-signature exists and is otherwise undefined.

10.3.3.14 Semantics of symbol maps

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, \Sigma, \texttt{SymbolMap}) & = \sigma : \Sigma \to \Sigma' \\ & : \mathit{SignatureMorphism} \end{array}}$$

If r is a $\texttt{SymbolMap}$ such that for every $\texttt{SymbolMapItem}$ gItem in $r.\texttt{generalSymbolMapItem}$ we have that $\mathit{gItem}.\texttt{source}$ is a symbol in $|\Sigma|$ and for every $\texttt{Symbol}$ s in $r.\texttt{generalSymbolMapItem}$ we have that s is a symbol in $|\Sigma|$,

$$\mathit{sem}(\Gamma, \Sigma, r) = \sigma : \Sigma \to \Sigma'$$

where σ must be the unique signature morphism with the properties

1) σ matches r: for each $\texttt{SymbolMapItem}$ $r.\texttt{generalSymbolMapItem}$ gItem in $r.\texttt{generalSymbolMapItem}$, we have that $|\sigma|(\mathit{gItem}.\texttt{source}) = \mathit{gItem}.\texttt{target}$ and for each $\texttt{Symbol}$ s in $r.\texttt{generalSymbolMapItem}$, $|\sigma|(s) = s$,

2) σ is the identity outside the domain of r: for each symbol $s \in |\mathit{Sigma}|$ such that there is no $\texttt{SymbolMapItem}$ gItem in $r.\texttt{generalSymbolMapItem}$ with $\mathit{gItem.source} = s$, $|\sigma|(s) = s$,

3) σ is surjective on $|\Sigma'|$: for each $y \in |\Sigma'|$, there is $x \in |\Sigma|$ such that $|\sigma|(x) = y$,

4) σ is final: for each signature Σ'' and each map of symbols $r' : |\Sigma'| \times |\Sigma''|$, r' determines a signature morphism $\sigma'' : \Sigma' \to \Sigma''$ whenever $|\sigma|; r'$ determines a signature morphism $\sigma_{r'} : \Sigma \to \Sigma''$.

If σ does not exist or is not uniquely determined, the semantics of r is undefined.

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, \Sigma, \Sigma', \texttt{SymbolMap}) & = \sigma : \Sigma \to \Sigma' \\ & : \mathit{SignatureMorphism} \end{array}}$$

If r is a $\texttt{SymbolMap}$ such that for each $\texttt{SymbolMapItem}$ gItem in $r.\texttt{generalSymbolMapItem}$ we have that $\mathit{gItem}.\texttt{source} \in |\Sigma|$ and $\mathit{gItem}.\texttt{target} \in |\Sigma'|$
and for each $\texttt{Symbol}$ s in $r.\texttt{generalSymbolMapItem}$ we have that s is an element both of $|\Sigma|$ and $|\Sigma'|$,

$$\mathit{sem}(\Gamma, \Sigma, \Sigma', m) = \sigma : \Sigma \to \Sigma'$$

where σ must be the unique element of the set $\{\varphi : \Sigma \to \Sigma' \mid$ there is a set $S \subseteq |\Sigma|$ such that φ matches r and is the identity on S and S is maximal with the property that such morphism exists$\}$. If the set fails to be a singleton, the semantics of r is undefined.

10.3.3.15 Semantics of extractions

$$\boxed{\begin{array}{ll} sem(\Gamma, (\Sigma, \Delta), \texttt{Extraction}) & = (\Sigma', \Delta') \\ & : (Signature, Sentences) \end{array}}$$

If e is an `Extraction`,

$$sem(\Gamma, (\Sigma, \Delta), e) = (\Sigma', \Delta')$$

where $sem(\Gamma, \Sigma, e.\texttt{removalKind}, e.\texttt{interfaceSignature}) = \Sigma''$, $\langle \Sigma', \Delta' \rangle$ is the smallest depleting Σ''-module, i.e., the smallest sub-theory $\langle \Sigma', \Delta' \rangle$ of (Σ, Δ) such that the following model-theoretic inseparability holds

$$\Delta \setminus \Delta' \equiv_{\Sigma' \cup \Sigma''} \emptyset.$$

This means intuitively that $\Delta \setminus \Delta'$ cannot be distinguished from $\emptyset$ (as far as $\Sigma' \cup \Sigma''$ is concerned) and formally that

$$\begin{array}{ll} & \{M|_{\Sigma' \cup \Sigma''} \mid M \in \mathsf{Real}(\Sigma), M \models \Delta \setminus \Delta'\} \\ = & \{M|_{\Sigma' \cup \Sigma''} \mid M \in \mathsf{Real}(\Sigma)\}. \end{array}$$

[36] defines the concept of smallest depleting Σ-module in a description logic context and shows that the smallest depleting Σ''-module exists in description logics. [29] generalizes both the definition of smallest depleting Σ''-module and the mentioned result to arbitrary institutions.

10.3.3.16 Semantics of approximations

$$\boxed{\begin{array}{ll} sem(\Gamma, (\mathcal{I}, \Sigma), \texttt{Approximation}) & = (\rho : \mathcal{I} \to \mathcal{I}', \Sigma') \\ & : (Morphism, Signature) \end{array}}$$

If a is an `Approximation`,

$$sem(\Gamma, (\mathcal{I}, \Sigma), a) = (\rho, \Sigma')$$

where $\Sigma' = sem(\Gamma, \Sigma, a.\texttt{removalKind}, a.\texttt{interfaceSignature})$, and ρ is the default projection (institution morphism) from $\mathcal{I}$ to $sem(\Gamma, a.\texttt{logicRef}) = \mathcal{I}'$, when $a.\texttt{logicRef}$ is present and the identity institution morphism on $\mathcal{I}$, when $a.\texttt{logicRef}$ is missing.

10.3.3.17 Semantics of filtering

$$\boxed{\begin{array}{ll} sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), \texttt{Filtering}) & = (c, \mathcal{I}', \Sigma', \Delta') \\ & : ('keep' | 'remove', Institution, Signature, Sentences) \end{array}}$$

If f is a `Filtering` such that $f.\texttt{removalKind} = keep$,

$$sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), f) = (keep, \mathcal{I}', \Sigma', \Delta')$$

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

If f is a $\texttt{Filtering}$ such that $f.\texttt{removalKind} = \mathit{remove}$,

$$sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), f) = (\mathit{remove}, \mathcal{I}', \Sigma', \Delta')$$

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

$$\boxed{\begin{array}{ll} sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), \texttt{BasicOMSOrSymbolList}) & = (\mathcal{I}', \Sigma', \mathcal{M}', \Delta') \\ & : (\mathit{Institution}, \mathit{Signature}, \mathit{Sentences}) \end{array}}$$

If O is a $\texttt{BasicOMS}$, we have defined $sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), O) = (\mathit{Gamma}, (\mathcal{I}', \Sigma', \mathcal{M}', \Delta'))$ and the semantics of O as a $\texttt{BasicOMSOrSymbolList}$ is $(\mathcal{I}', \Sigma', \mathcal{M}', \Delta')$.

If s is a set of symbols, $sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), s) = (\mathcal{I}, sem(\Gamma, \Sigma, s), \emptyset, \emptyset)$.

10.3.3.18 Semantics of extension

$$\boxed{\begin{array}{ll} sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), & \texttt{Extension}) = (\Gamma', (\mathcal{I}, \Sigma', \mathcal{M}', \Delta')) \\ & : (\mathit{LogicalEnvironment}, (\mathit{Institution}, \mathit{Signature}, \mathit{RealizationClass}, \mathit{Sentences})) \end{array}}$$

If e is an $\texttt{Extension}$,

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

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

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

If $e.\texttt{extensionName}$ is present, the inclusion link is labeled with this name.

10.3.3.19 Semantics of interface signatures

$$\boxed{\begin{array}{ll} sem(\Gamma, \Sigma, \texttt{RemovalKind}, \texttt{InterfaceSignature}) & = \Sigma' \\ & : \mathit{Signature} \end{array}}$$

If r is a $\texttt{RemovalKind}$ and s is an $\texttt{InterfaceSignature}$,

$$sem(\Gamma, \Sigma, r, s) = \Sigma'$$

where

$$\Sigma' = \begin{cases} \Sigma \cap sem(\Gamma, \Sigma, s.\texttt{symbolList}) & \text{if} \quad r = \mathit{keep} \\ \Sigma \setminus sem(\Gamma, \Sigma, s.\texttt{symbolList}) & \text{if} \quad r = \mathit{remove} \end{cases}$$

10.3.3.20 Semantics of OMS definitions

$$\boxed{\begin{array}{ll} sem(\Gamma, \texttt{OMSDefinition}) & = \Gamma' \\ & : \mathit{LogicalEnvironment} \end{array}}$$

An `OMSDefinition` O extends the global environment.

$$sem(\Gamma, O) = \Gamma''$$

where for each of the institutions $\mathcal{I}_1, \ldots, \mathcal{I}_n$ supported by $\Gamma.lang$, we have
$\Gamma_1 = \Gamma'_1[postfixLogicIRI(O.\texttt{omsName}, \mathcal{I}_1) \mapsto (\mathcal{I}_1, \Sigma_1, \mathcal{M}_1, \Delta_1)]$
where $sem(\Gamma.logic = \mathcal{I}_1, O.\texttt{oms}) = (\Gamma'_1, (\mathcal{I}_1, \Sigma_1, \mathcal{M}_1, \Delta_1))$,
$\Gamma_2 = \Gamma'_2[postfixLogicIRI(O.\texttt{omsName}, \mathcal{I}_2) \mapsto (\mathcal{I}_2, \Sigma_2, \mathcal{M}_2, \Delta_2)]$
where $sem(\Gamma'_1.logic = \mathcal{I}_2, O.\texttt{oms}) = (\Gamma'_2, (\mathcal{I}_2, \Sigma_2, \mathcal{M}_2, \Delta_2))$, ...,
$\Gamma'' = \Gamma'_n[postfixLogicIRI(O.\texttt{omsName}, \mathcal{I}_n) \mapsto (\mathcal{I}_n, \Sigma_n, \mathcal{M}_n, \Delta_n)]$
where $sem(\Gamma'_{n-1}.logic = \mathcal{I}_n, O.\texttt{oms}) = (\Gamma'_n, (\mathcal{I}_n, \Sigma_n, \mathcal{M}_n, \Delta_n))$

The conservativity strength annotations refer to the semantics of $O.\texttt{oms}$ in the current logic of Γ. Therefore, let $(\Gamma_0, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)) = sem(\Gamma, O.\texttt{oms})$.

If $O.$`conservativityStrength` is `model-conservative`, the semantics is only defined if $\mathcal{M} \neq \emptyset$.
If $O.$`conservativityStrength` is `consequence-conservative`, the semantics is only defined if Δ has only tautologies[30)] as signature-free[31)] logical consequences.
If $O.$`conservativityStrength` is `monomorphic`, the semantics is only defined if $\mathcal{M}$ consist of exactly one isomorphism class of realizations.
If $O.$`conservativityStrength` is `weak-definitional`, the semantics is only defined if $\mathcal{M}$ is empty or a singleton.
If $O.$`conservativityStrength` is `definitional`, the semantics is only defined if $\mathcal{M}$ is a singleton.

10.3.3.21 Semantics of OMS references

$$\boxed{\begin{aligned} sem(\Gamma, \texttt{OMSReference}) &= (\Gamma', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)) \\ &: (LogicalEnvironment, (Institution, Signature, RealizationClass, Sentences)) \end{aligned}}$$

The rule for `OMSReferences` has been given above, as `OMSReferences` are a particular case of `ClosableOMS`.

10.3.3.22 Semantics of symbols

$$\boxed{\begin{aligned} sem(\Gamma, \Sigma, \texttt{Symbol}) &= s \\ &: LogicalSymbol \end{aligned}}$$

If sym is a `Symbol`

$$sem(\Gamma, \Sigma, sym) = s$$

where s is a logic-specific symbol with the name $sym.\texttt{iri}$ from $|\Sigma|$. If such symbol does not exist, the semantics is undefined.

10.3.3.23 Semantics of symbol map items

$$\boxed{\begin{aligned} sem(\Gamma, \Sigma_1, \Sigma_2, \texttt{SymbolMapItem}) &= (s_1, s_2) \\ &: (LogicalSymbol, LogicalSymbol) \end{aligned}}$$

30) A tautology is a sentence holding in every realization.

31) A signature-free sentence is one over the empty signature.

If smi is a `SymbolMapItem`,

$$sem(\Gamma, \Sigma_1, \Sigma_2, smi) = (s_1, s_2)$$

where $sem(\Gamma, \Sigma_1, smi.\mathtt{source}) = s_1$ and $sem(\Gamma, \Sigma_2, smi.\mathtt{target}) = s_2$.

10.3.3.24 Semantics of general symbol map items

$$\boxed{\begin{aligned} sem(\Gamma, \Sigma_1, \Sigma_2, \mathtt{GeneralSymbolMapItem}) &= (s, t) \\ &: (LogicalSymbol, LogicalSymbol) \end{aligned}}$$

If $gsmi$ is a `SymbolMapItem`, then its semantics has been given in the previous rule.

If $gsmi$ is a `Symbol`, $sem(\Gamma, \Sigma_1, \Sigma_2, gsmi) = (s, s)$ where $sem(\Gamma, \Sigma_1, gsmi) = s$

10.3.3.25 Semantics of references

$$\boxed{\begin{aligned} sem(\mathtt{LolaRef}) &= L \\ &: Language|Institution \end{aligned}}$$

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

$$\boxed{\begin{aligned} sem(\mathtt{LanguageRef}) &= L \\ &: Language \end{aligned}}$$

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

$$\boxed{\begin{aligned} sem(\mathtt{SyntaxRef}) &= S \\ &: Serialization \end{aligned}}$$

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

$$\boxed{\begin{aligned} sem(\mathtt{LogicRef}) &= L \\ &: Institution \end{aligned}}$$

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

10.3.4 Semantics of OMS Mappings

10.3.4.1 Semantics of mapping definitions

$$\boxed{\begin{aligned} sem(\Gamma, \mathtt{MappingDefinition}) &= \Gamma' \\ &: LogicalEnvironment \end{aligned}}$$

See equations for `InterpretationDefinition`, `EntailmentDefinition`, `EquivalenceDefinition`, `ConservativeExtensionDefinition` and `AlignmentDefinition`.

10.3.4.2 Semantics of interpretation definitions

$$\boxed{\begin{array}{ll} sem(\Gamma, \texttt{InterpretationDefinition}) & = \Gamma' \\ & : LogicalEnvironment \end{array}}$$

If d is an `InterpretationDefinition`,

$$sem(\Gamma, d) = \Gamma'$$

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

— $(L_1, L_2) = sem(\Gamma, d.\texttt{interpretationType})$

— $\rho = (\Phi, \alpha, \beta) : \mathbf{Inst}(L_1) \to \mathbf{Inst}(L_2)$ is the comorphism given by $sem(\Gamma, d.\texttt{omsLanguageTranslation})$.
If $d.\texttt{OMSLanguageTranslation}$ is missing, the default translations between the logics is selected.

— $\sigma = sem(\Gamma.\{current = (lang, logic', ser)\}, \Phi(\mathsf{Sig}(L_1)), \mathsf{Sig}(L_2), d.\texttt{symbolMap})$, where $\Gamma.current = (lang, logic, ser)$ and $logic'$ is the target logic of ρ, or, if $d.\texttt{symbolMap}$ is missing, σ is the identity signature morphism on $\Phi(\mathsf{Sig}(L_1))$ which must be equal with $\mathsf{Sig}(L_2)$.

The semantics is only defined if $\beta_{\mathsf{Sig}(L_1)}(M_2|_\sigma) \in \mathsf{Real}(L_1)$ for each $M_2 \in \mathsf{Real}(L_2)$.
If the optional argument $d.\texttt{conservativityStrength}$ is

— `model-conservative`, for each realization $M_1 \in |\mathsf{Real}(L_1)|$ there must exist a realization $M_2 \in |\mathsf{Real}(L_2)|$ such that $\beta_{\mathsf{Sig}(L_1)}(M_2|_\sigma) = M_1$.

— `consequence-conservative`, for each $\mathsf{Sig}(L_1)$-sentence φ, if $\mathcal{M}_2 \models \sigma(\alpha_{\mathsf{Sig}(L_1)}(\varphi))$ then $\mathcal{M}_1 \models \varphi$.

— `not-model-conservative`, there must exist a realization $M_1 \in |\mathsf{Real}(L_1)|$ such that there is no realization $M_2 \in |\mathsf{Real}(L_2)|$ such that $\beta_{\mathsf{Sig}(L_1)}(M_2|_\sigma) = M_1$.

— `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 $\mathcal{M}_1 \not\models \varphi$.

10.3.4.3 Semantics of refinement definitions

$$\boxed{\begin{array}{ll} sem(\Gamma, \texttt{RefinementDefinition}) & = \Gamma' \\ & : LogicalEnvironment \end{array}}$$

If d is a `RefinementDefinition`,

$$sem(\Gamma, d) = \Gamma'$$

where $\Gamma' = \Gamma[d.\texttt{interpretationName} \mapsto sem(\Gamma, d.\texttt{refinement})]$.

10.3.4.4 Semantics of interpretation types

$$\boxed{\begin{array}{ll} sem(\Gamma, \texttt{InterpretationType}) & = ((N_1, \mathcal{I}_1, \Sigma_1, \mathcal{M}_1, \Delta_1), (N_2, \mathcal{I}_2, \Sigma_2, \mathcal{M}_2, \Delta_2)) \\ & : (NodeLabel, NodeLabel) \end{array}}$$

If t is an `InterpretationType`,

$$sem(\Gamma, t) = (L_1, L_2)$$

where

— $\mathbf{Name}(L_1) = \mathbf{Name}(t.\texttt{source})$ and $\mathbf{Name}(L_2) = \mathbf{Name}(t.\texttt{target})$,

— $(\mathbf{Inst}(L_1), \mathsf{Sig}(L_1), \mathsf{Real}(L_1), \mathsf{Th}(L_1)) = \mathit{sem}(\Gamma, t.\texttt{source})$,

— $(\mathbf{Inst}(L_2), \mathsf{Sig}(L_2), \mathsf{Real}(L_2), \mathsf{Th}(L_2)) = \mathit{sem}(\Gamma, t.\texttt{target})$,

10.3.4.5 Semantics of refinements

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, \texttt{Refinement}) & = (((G_1, G_2), \sigma, \mathcal{M}) \\ & : (OMSGraph, OMSGraph, GraphMorphism, RealizationClass) \end{array}}$$

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 the mapping is stored along which the refinement is done. Given two networks G_1 and G_2, a *network morphism* $\sigma : G_1 \to G_2$ is

1) a graph homomorphism $\sigma^G : \mathit{Shape}(G_1) \to \mathit{Shape}(G_2)$, where given a network G, its shape $\mathit{Shape}(G)$ is a graph with same nodes and edges as G but with no labels of nodes,

2) a natural transformation $\sigma^M : G_1 \to \sigma^G; G_2$

such that

1) 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, there is a signature morphism $(\rho^M_{N_1}, \sigma^M_{N_1}) : (\mathcal{I}_1, \Sigma_1) \to (\mathcal{I}_2, \Sigma_2)$, where

2) $\rho^M_{N_1} = (\Phi, \alpha, \beta) : \mathcal{I}_1 \to \mathcal{I}_2$ 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 realization is a class $\mathcal{M}$ of pairs of families of realizations compatible with the two networks. Given a network morphism $\sigma : G_1 \to G_2$ and a G_2-realization F, $F|_\sigma$ is defined as the family of realizations $\{M_i\}_{i \in Nodes(G_1)}$ such that $M_i = F_{\sigma^G(i)}|_{\sigma^M_i}$ for each $i \in Nodes(G_1)$.

Thus, the semantics of a `Refinement` consists of

— a refinement signature (G_1, G_2),

— a network morphism σ and

— a refinement realization $\mathcal{M}$.

If r is `RefinementOMS`,

$$\mathit{sem}(\Gamma, \mathtt{r}) = ((G, G), \sigma, \mathcal{M})$$

where

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

— σ is the identity morphism on $\mathsf{Sig}(r.\texttt{oms})$,

— $\mathcal{M} = \{((M), (M)) \mid M \in \mathsf{Real}(r.\texttt{oms})\}$, where (M) is the singleton family consisting of M.

If r is `RefinementNetwork`,

$$\mathit{sem}(\Gamma, r) = ((G, G), \sigma, \mathcal{M})$$

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

If r is `SimpleOMSRefinement`,

$$sem(\Gamma, r) = ((G_1, G_2'), \sigma', \mathcal{M})$$

where $sem(\Gamma, r.\texttt{refinement}) = ((G_1, G_1'), \sigma_1, \mathcal{M}_1)$, $sem(\Gamma, r.\texttt{refinement2}) = ((G_2, G_2'), \sigma_2, \mathcal{M}_2)$ G_1' and G_2 are both graphs with one isolated node, labeled $(name1, \mathcal{I}_1, \Sigma_1, \mathcal{M}_1, \Delta_1)$ and, respectively, $(name2, \mathcal{I}_2, \Sigma_2, \mathcal{M}_2, \Delta_2)$, $sem(\Gamma, \Sigma_1, \Sigma_2, r.\texttt{omsRefinementMap}) = (\rho = (\Phi, \alpha, \beta) : \mathcal{I}_1 \to \mathcal{I}_2, \sigma : \Phi(\Sigma_1) \to \Sigma_2)$,
σ' maps the node n of G_1 to the node of G_3 and $(\sigma')^M_n = (\sigma_1)^M_n; (\rho; \sigma); (\sigma_2)^M_{\sigma_1^G(n)}$, and $\mathcal{M} = \{(M_1, M_3) \mid \exists M_2$ such that $(M_2, M_3) \in \mathcal{M}_2$ and $(M_1, M_2|_{(\rho,\sigma)}) \in \mathcal{M}_1\}$ The refinement is correct only if for each $(M, N) \in \mathcal{M}_2$, there exists $(M_1, M_2|_{(\rho,\sigma)}) \in \mathcal{M}_1$.

If r is `SimpleNetworkRefinement`,

$$sem(\Gamma, r) = ((G_1, G_2'), \sigma', \mathcal{M}')$$

where
$sem(\Gamma, r.\texttt{refinement}) = ((G_1, G_1'), \sigma_1, \mathcal{M}_1)$,
$sem(\Gamma, r.\texttt{refinement2}) = ((G_2, G_2'), \sigma_2, \mathcal{M}_2)$,
$sem(\Gamma, G_1', G_2, r.\texttt{networkRefinementMap}) = \sigma : G_1' \to G_2$,
$\sigma' = \sigma_1; \sigma; \sigma_2$ and $\mathcal{M} = \{(F_1, F_3) \mid \exists F_2$ such that $(F_1, F_2|_\sigma) \in \mathcal{M}_1$ and $(F_2, F_3) \in \mathcal{M}_2\}$.
The refinement is correct only if for each $(F_2, F_3) \in \mathcal{M}_2$, there is a $(F_1, F_2|_\sigma) \in \mathcal{M}_1$.

10.3.4.6 Semantics of a set of refinements

$$\boxed{\begin{aligned} sem(\Gamma, G_1, G_2, Set(\texttt{Refinement})) &= \sigma \\ &: GraphMorphism \end{aligned}}$$

If $r_1, \ldots, r_n$ are all `Refinements`,

$$sem(\Gamma, G_1, G_2, Set(r_1, \ldots, r_n)) = \sigma$$

where $sem(\Gamma, r_1) = ((G_1^1, G_2^1), \sigma_1, \mathcal{M}_1)$, $\ldots$, $sem(\Gamma, r_n) = ((G_1^n, G_2^n), \sigma_n, \mathcal{M}_n)$ such that $G_1 = \bigcup_{i=1,\ldots,n} G_1^i$ and no node of G_1 appears in two graphs G_1^i and G_1^j for some $i \neq j \in 1, \ldots, n$, $G_2 = \bigcup_{i=1,\ldots,n} G_2^i$ and no node of G_2 appears in two graphs G_2^i and G_2^j for some $i \neq j \in 1, \ldots, n$, and $\sigma : G_1 \to G_2$ is defined by $\sigma^G(n) = \sigma_i^G(n)$ if the node n comes from G_1^i and similarly for such a node n of G coming from G_1^i, we have that $\sigma_n^M = \sigma_i^M(n)$. Moreover, σ must be total on the nodes of G_1.

10.3.4.7 Semantics of refinement maps

$$\boxed{\begin{aligned} sem(\Gamma, G_1, G_2, \texttt{RefinementMap}) &= \sigma \\ &: GraphMorphism \end{aligned}}$$

If m is an `OMSRefinementMap`,

$$sem(\Gamma, G_1, G_2, m) = (name_1, name_2, \rho, \sigma)$$

where
G_1 must be a graph with just one isolated node labeled $(name_1, \mathcal{I}_1, \Sigma_1, \mathcal{M}_1, \Delta_1)$
G_2 must be a graph with just one isolated node labeled $(name_2, \mathcal{I}_2, \Sigma_2, \mathcal{M}_2, \Delta_2)$,
$sem(\Gamma, m.\texttt{translation}) = \rho = (\Phi, \alpha, \beta) : \mathcal{I}_1 \to \mathcal{I}_2$, or if $m.\texttt{translation}$ is missing, the default comorphism between $\mathcal{I}_1$ and $\mathcal{I}_2$,
$sem(\Gamma', \Phi(\Sigma_1), \Sigma_2, m.\texttt{symbolMap}) = \sigma : \Phi(\Sigma_1) \to \Sigma_2$ where $\Gamma.current = (lang, logic, ser)$, $logic'$ is the target logic of (Φ, α, β), $lang'$ and ser' are the default language and serialization for $logic'$ and $\Gamma' = \Gamma.current =$

$(lang', logic', ser')$, or, when m.symbolMap is missing, $\Phi(\Sigma_1)$ and Σ_2 must be the same and σ is the identity signature morphism on Σ_2.

If m is a NetworkRefinementMap,

$$sem(\Gamma, G_1, G_2, m) = \sigma$$

where $sem(\Gamma, G_1, G_2, m.refinements) = \sigma$.

10.3.4.8 Semantics of entailment definitions

$$\boxed{\begin{array}{ll} sem(\Gamma, \texttt{EntailmentDefinition}) & = \Gamma' \\ & : LogicalEnvironment \end{array}}$$

If e is an EntailmentDefinition,

$$sem(\Gamma, e) = \Gamma'$$

where $\Gamma' = \Gamma[e.\texttt{entailmentName} \mapsto sem(\Gamma, e.\texttt{entailmentType})]$.

10.3.4.9 Semantics of entailment types

$$\boxed{\begin{array}{ll} sem(\Gamma, \texttt{EntailmentType}) & = G \\ & : OMSGraph \end{array}}$$

If t is an OMSOMSEntailment,

$$sem(\Gamma, t) = L_2 \stackrel{\mathrm{id}}{\to} L_1$$

where $\mathbf{Name}(L_1) = \mathbf{Name}(t.\texttt{premise})$, $\mathbf{Name}(L_2) = \mathbf{Name}(t.\texttt{conclusion})$,
$(\mathbf{Inst}(L_1), \mathsf{Sig}(L_1), \mathsf{Real}(L_1), \mathsf{Th}(L_1)) = sem(\Gamma, t.\texttt{premise})$,
$(\mathbf{Inst}(L_2), \mathsf{Sig}(L_2), \mathsf{Real}(L_2), \mathsf{Th}(L_2)) = sem(\Gamma, t.\texttt{conclusion})$ such that
$\mathsf{Sig}(L_1) = \mathsf{Sig}(L_2)$ and $\mathsf{Real}(L_1) \subseteq \mathsf{Real}(L_2)$ and id is the identity morphism on $\mathsf{Sig}(L_1)$.

If t is a NetworkOMSEntailment, $sem(\Gamma, t) = G$

where $sem(\Gamma, t.\texttt{network}) = G'$ such that G' contains a node n labeled with $\mathbf{Name}(t.\texttt{premise})$,
$sem(\Gamma, t.\mathtt{oms}) = (\mathcal{I}, \Sigma, \mathcal{M}_2, \Delta_2)$ and
$\{\mathcal{M}_n \mid \mathcal{M} \text{ is compatible with } G'\} \subseteq \mathcal{M}_2$. Then G extends G' with a new node whose label has the name $\mathbf{Name}(t.\mathtt{oms})$ and the other components given by $sem(\Gamma, t.\mathtt{oms})$ and with a new theorem link from this new node to the node $\mathbf{Name}(t.\mathtt{omsName})$, labeled with the identity morphism on Σ.

If t is a NetworkNetworkEntailment,

$$sem(\Gamma, t) = G$$

where $sem(\Gamma, t.\texttt{premise}) = G_1$, $sem(\Gamma, t.\texttt{conclusion}) = G_2$, such that $Shape(G_1) = Shape(G_2)$ and, for each node $i \in |Shape(G_1)|$, its names in the networks G_1 and G_2 are the same, its signatures are the same and the class of realizations obtained by projecting each family of realizations compatible with G_1 to the component i is included in the class of realizations obtained by projecting each family of realizations 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 occurrences 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 $\mathsf{Sig}(i_1)$.

10.3.4.10 Semantics of equivalence definitions

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, \texttt{EquivalenceDefinition}) & = \Gamma' \\ & : \mathit{LogicalEnvironment} \end{array}}$$

If d is an `EquivalenceDefinition`,

$$\mathit{sem}(\Gamma, d) = \Gamma'$$

where $\Gamma' = \Gamma[d.\texttt{equivalenceName} \mapsto \mathit{sem}(\Gamma, d.\texttt{equivalenceType})]$.

10.3.4.11 Semantics of OMS equivalences

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, \texttt{OMSEquivalence}) & = (G, N_1, N_2) \\ & : (\mathit{OMSGraph}, \mathit{Node}, \mathit{Node}) \end{array}}$$

If t is an `OMSEquivalence`,

$$\mathit{sem}(\Gamma, t) = (G, N_1, N_2)$$

where $O_1 = t.\texttt{oms}$, $O_2 = t.\texttt{oms2}$, $O_3 = t.\texttt{mediatingOMS}$,
$\mathit{sem}(\Gamma, (\mathcal{I}, \mathsf{Sig}(O_1) \cup \mathsf{Sig}(O_2), \mathsf{Real}^{\mathcal{I}}(\mathsf{Sig}(O_1) \cup \mathsf{Sig}(O_2)), \emptyset), O_3) = (\Gamma', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta))$
G is the graph $N_1 \xrightarrow{\iota_1} N_3 \xleftarrow{\iota_2} N_3$ where

1) N_1 is labeled with $(\mathbf{Name}(O_1), \mathbf{Inst}(O_1), \mathsf{Sig}(O_1), \mathsf{Real}(O_1), \mathsf{Th}(O_1))$,

2) N_2 is labeled with $(\mathbf{Name}(O_2), \mathbf{Inst}(O_2), \mathsf{Sig}(O_2), \mathsf{Real}(O_2), \mathsf{Th}(O_2))$ and

3) N_3 is labeled with $(\mathbf{Name}(O_3), \mathcal{I}, \Sigma, \mathcal{M}, \Delta)$

such that

1) $\iota_i : \mathsf{Sig}(O_i) \to \Sigma$ are signature inclusions,

2) $\mathcal{I} = \mathbf{Inst}(O_1) = \mathbf{Inst}(O_2)$ and

3) for each $i = 1, 2$ and each realization $M_i \in \mathsf{Real}(O_i)$ there exists a unique realization $M \in \mathcal{M}$ such that $M|_{\mathsf{Sig}(O_i)} = M_i$.

10.3.4.12 Semantics of network equivalences

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, \texttt{NetworkEquivalence}) & = (G_1, G_2, G_3) \\ & : (\mathit{OMSGraph}, \mathit{OMSGraph}, \mathit{OMSGraph}) \end{array}}$$

If t is a `NetworkEquivalence`,

$$\mathit{sem}(\Gamma, t) = (G_1, G_2, G_3)$$

where $n_1 = t.\texttt{network}$, $n_2 = t.\texttt{network2}$, $n_3 = t.\texttt{mediatingNetwork}$, $\mathit{sem}(\Gamma, n_1) = G_1$, $\mathit{sem}(\Gamma, n_2) = G_2$, $\mathit{sem}(\Gamma, n_3) = G_3$ such that G_1 and G_2 are subgraphs of G_3 and for each $i = 1, 2$ and each family of realizations $\mathcal{M}_i$ compatible with G_i there is a unique family of realizations $\mathcal{M}$ compatible with G_3 such that the projection of $\mathcal{M}$ to the nodes in G_i is $\mathcal{M}_i$.

10.3.4.13 Semantics of conservative extension definitions

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, \texttt{ConservativeExtensionDefinition}) & = \Gamma' \\ & : \mathit{LogicalEnvironment} \end{array}}$$

If d is a `ConservativeExtensionDefinition`,

$$\mathit{sem}(\Gamma, d) = \Gamma'$$

where $O_1 = d.\texttt{moduleType.module}$, $O_2 = d.\texttt{moduleType.whole}$, $c = d.\texttt{conservativityType}$, $\Sigma = \mathit{sem}(\Gamma, d.\texttt{interfaceSignature})$, $\Gamma' = \Gamma[d.\texttt{moduleName} \mapsto (G, \iota, N_2, N_1)]$ and G is the graph $N_1 \xrightarrow{\iota} N_2$ where N_1 is labeled with $(O_1, \mathbf{Inst}(O_1), \mathsf{Sig}(O_1), \mathsf{Real}(O_1), \mathsf{Th}(O_1))$, N_2 with $(O_2, \mathbf{Inst}(O_2), \mathsf{Sig}(O_2), \mathsf{Real}(O_2), \mathsf{Th}(O_2))$, and ι is an inclusion, when $\Sigma \subseteq \mathsf{Sig}(O_2) \subseteq \mathsf{Sig}(O_1)$ and if $c = \texttt{model-conservative}$ and for each $M \in \mathsf{Real}(O_2)$ there is a realization $M' \in \mathsf{Real}(O_1)$ such that $M'|_\Sigma = M|_\Sigma$, or if $c = \texttt{consequence-conservative}$ and for each $\varphi \in \mathsf{Sen}(\Sigma)$, $O_1 \models \varphi$ implies $O_2 \models \varphi$.

10.3.4.14 Semantics of alignment definitions

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, \texttt{AlignmentDefinition}) & = \Gamma' \\ & : \mathit{LogicalEnvironment} \end{array}}$$

If d is an `AlignmentDefinition`,

$$\mathit{sem}(\Gamma, d) = \Gamma'$$

where $\mathit{sem}(\Gamma, d.\texttt{alignmentType}) = (\Gamma_0, L_1, L_2)$ and $\Gamma' = \Gamma_0[d.\texttt{AlignmentName} \mapsto (G, L_1', L_2')]$,
where $(L_1', L_2') = \mathit{sem}(\Gamma, L_1, L_2, d.\texttt{alignmentSemantics})$,
$\mathit{card} = d.\texttt{alignmentCardinality}$ or, when this is missing, $\mathit{card} = ('1', '1')$,
$\mathit{aSem} = d.\texttt{alignmentSemantics}$ or, when this is missing, $\mathit{aSem} = \texttt{single-domain}$,
and $G = \mathit{sem}(\Gamma_0, L_1', L_2', \mathit{card}, \mathit{aSem}, d.\texttt{correspondence})$.

10.3.4.15 Semantics of alignment types

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, \texttt{AlignmentType}) & = (\Gamma', L_1, L_2) \\ & : (\mathit{LogicalEnvironment}, \mathit{NodeLabel}, \mathit{NodeLabel}) \end{array}}$$

If t is an `AlignmentType`

$$\mathit{sem}(\Gamma, t) = (\Gamma'', L_1, L_2)$$

where $\mathit{sem}(\Gamma, t.\texttt{source}) = (\Gamma', (\mathcal{I}_1, \Sigma_1, \mathcal{M}_1, \Delta_1))$, $\mathit{sem}(\Gamma', t.\texttt{target}) = (\Gamma'', (\mathcal{I}_2, \Sigma_2, \mathcal{M}_2, \Delta_2))$, L_1 and L_2 are the labels of the nodes of $t.\texttt{source}$ and $t.\texttt{target}$ in $\Gamma'.\mathit{imports}$.

10.3.4.16 Semantics of alignments

$$\boxed{\begin{array}{ll} \mathit{sem}(\Gamma, L_1, L_2, (\texttt{AlignmentCardinality}, \texttt{AlignmentCardinality}), \texttt{AlignmentSemantics}, & \mathit{Set}(\texttt{Correspondence})) = G \\ & : \mathit{OMSGraph} \end{array}}$$

If $\mathit{card}_1, \mathit{card}_2$ are `AlignmentCardinality`, aSem is an `AlignmentSemantics` and $C = \mathit{Set}\{c_1, \ldots, c_n\}$ a set of `Correspondences`,

$$\mathit{sem}(\Gamma, L_1, L_2, (\mathit{card}_1, \mathit{card}_2), \mathit{aSem}, C) = G$$

where $sem(\Gamma, \mathsf{Sig}(L_1), \mathsf{Sig}(L_2), aSem, C) = (\Sigma_s, \Sigma_t, (\Sigma, \Delta), \phi_s : \Sigma_s \to \Sigma, \phi_t : \Sigma_t \to \Sigma, smap, cvalues)$,
where the semantics of the alignment is not defined in the following cases:

— if $cvalues = True$ and then at least one of the correspondences in C has a confidence value different than 1 or

— if the alignment does not have the specified cardinality, i.e.,

  — if $card_1 =' ?'$, then $smap$ must be injective,

  — if $card_1 =' +'$, then $smap$ must be total on the symbols of $\mathsf{Sig}(L_1)$,

  — if $card_1 =' 1'$, then $smap$ must be injective and total,

  — if $card_1 =' *'$, then no cardinality restriction on $smap$ is made,

  — if $card_2 =' ?'$, then $smap^{-1}$ must be injective,

  — if $card_2 =' +'$, then $smap^{-1}$ must be total on the symbols of $\mathsf{Sig}(L_2)$,

  — if $card_2 =' 1'$, then $smap^{-1}$ must be injective and total,

  — if $card_2 =' *'$, then no cardinality restriction on $smap^{-1}$ is made,

and when the above conditions are met, G is a W-shaped graph as below

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

where $L_s = (alignName + +\text{"\_source"}, \mathbf{Inst}(L_1), \Sigma_s, \mathsf{Real}(\Sigma_s), \emptyset)$,
$L_t = (alignName + +\text{"\_target"}, \mathbf{Inst}(L_2), \Sigma_t, \mathsf{Real}(\Sigma_t), \emptyset)$ and
$L_B = (alignName + +\text{"\_bridge"}, \mathcal{I}, \Sigma, \mathsf{Real}((\Sigma, \Delta)), \Delta)$.

10.3.4.17 Semantics of sets of correspondences

$$\boxed{\begin{aligned} sem(\Gamma, \Sigma_1, \Sigma_2, \texttt{AlignmentSemantics}, Set(\texttt{Correspondence})) &= (\Sigma_s, \Sigma_t, (\mathcal{I}, \Sigma, \Delta), \phi_s : \Sigma_s \to \Sigma, \phi_t : \Sigma_t \to \Sigma, smap, cvalues) \\ &: (Signature, Signature, (Institution, Signature, Sentences), \\ &\quad SignatureMorphism, SignatureMorphism, MapOfSymbols, Bool) \end{aligned}}$$

If $c_1, \ldots, c_n$ are all `Correspondence`s and $aSem$ is an `AlignmentSemantics`,

$$sem(\Gamma, \Sigma_1, \Sigma_2, aSem, Set(c_1, \ldots, c_n)) = (\Sigma_s, \Sigma_t, (\Sigma, \Delta), \phi_s, \phi_t, smap, cvalues)$$

where $sem(\Gamma, \Sigma_1, \Sigma_2, (1, \texttt{equivalent}), c_i) = (clist_i, cvalues_i)$ for $i = 1, \ldots, n$,
$cvalues = \vee_{i=1,\ldots,n} cvalues_i$,
$smap = \{s_i^1 \mapsto s_i^2\}$,
$(\mathcal{I}, \Sigma, \Delta, \phi_s : \Sigma_s \to \Sigma, \phi_t : \Sigma_t \to \Sigma) = theoryOfCorrespondences_{\Gamma.logic}(aSem, \Sigma_1, \Sigma_2, clist_1 + + \ldots + + clist_n)$.

10.3.4.18 Semantics of correspondences

$$\boxed{\begin{aligned} sem(\Gamma, \Sigma_1, \Sigma_2, (defaultConf, defaultRel), \texttt{Correspondence}) &= (clist, cvalues) \\ &: (Sequence((Relation, Symbol, Symbol)), Bool) \end{aligned}}$$

If c is a `DefaultCorrespondence`,

$$sem(\Gamma, \Sigma_1, \Sigma_2, (\mathit{defaultConf}, \mathit{defaultRel}), c) = (\mathit{clist}, \mathit{cvalues})$$

where $\mathit{cvalues} = True$ if $\mathit{defaultConf}$ is different than 1 and $False$ otherwise,
$Sequence((sym_1^1, sym_1^2), \ldots, (sym_k^1, sym_k^2)) = sameName_{\Gamma.logic}(\Sigma_1, \Sigma_2)$,
$clist = Sequence((defaultRel, sym_i^1, sym_i^2)\}_{i=1,\ldots,k})$.

If c is a `SingleCorrespondence`,

$$sem(\Gamma, \Sigma_1, \Sigma_2, (\mathit{defaultConf}, \mathit{defaultRel}), c) = (\mathit{clist}, \mathit{cvalues})$$

where $conf = \begin{cases} \mathit{defaultConf} & c.\texttt{confidence} \text{ is missing,} \\ c.\texttt{confidence} & \text{otherwise} \end{cases}$
$rel = \begin{cases} \mathit{defaultRel} & c.\texttt{relation} \text{ is missing,} \\ c.\texttt{relation} & \text{otherwise} \end{cases}$
$cvalues = \begin{cases} False & conf = 1 \\ True & \text{otherwise} \end{cases}$
$sem(\Gamma, \Sigma_1, c.generalizedTerm) = sym1$, $sem(\Gamma, \Sigma_2, c.symbolRef) = sym2$, $clist = Sequence((rel, sym1, sym2))$,

If c is a `CorrespondenceBlock`,

$$sem(\Gamma, \Sigma_1, \Sigma_2, (\mathit{defaultConf}, \mathit{defaultRel}), c) = (\mathit{clist}, \mathit{cvalues})$$

where if $c.\texttt{relation}$ is missing, $rel = \mathit{defaultRel}$, else $rel = c.\texttt{relation}$
if $c.\texttt{confidence}$ is missing, $conf = \mathit{defaultConf}$, else $conf = c.\texttt{confidence}$
for all correspondences $c_1, \ldots, c_n$ in $c.\texttt{correspondence}$, $(clist_i, cvalues_i) = sem(\Gamma, \Sigma_1, \Sigma_2, (conf, rel), c_i)$,
$clist = clist_1 + + \ldots + +clist_n$ and $cvalues = \vee_{i=1,\ldots,n} cvalues_i$.

$$\boxed{\begin{array}{ll} sem(\Gamma, L_1, L_2, \texttt{AlignmentSemantics}) & = ((Name_1, \mathcal{I}_1', \Sigma_1', \mathcal{M}_1', \Delta_1'), (Name_2, cI_2', \Sigma_2', \mathcal{M}_2', \Delta_2')) \\ & : (NodeLabel, NodeLabel) \end{array}}$$

If s is an `AlignmentSemantics`, $L_1 = (aName, \mathcal{I}_1, \Sigma_1, \mathcal{M}_1, \Delta_1)$ and $L_2 = (aName', \mathcal{I}_2, \Sigma_2, \mathcal{M}_2, \Delta_2)$

$$sem(\Gamma, L_1, L_2, s) = (rel(L_1), rel(L_2))$$

where

$$rel(L) = \begin{cases} (L_1, L_2) & \text{if } s = \texttt{single-domain} \\ (name_1, \mathcal{I}_1, \Sigma_1', \mathcal{M}_1', \Delta_1'), (name_2, \mathcal{I}_2, \Sigma_2', \mathcal{M}_2', \Delta_2') & \text{otherwise} \end{cases}$$

$relativize_{\mathcal{I}_1}(\Sigma_1, \Delta_1) = (\Sigma_1', \Delta_1')$, $\mathcal{M}_1'$ is the class of Δ_1'-realizations, $name_1 =' relativized' + +aName$,
$relativize_{\mathcal{I}_2}(\Sigma_2, \Delta_2) = (\Sigma_2', \Delta_2')$, $\mathcal{M}_2'$ is the class of Δ_2'-realizations and $name_2 =' relativized' + +aName'$.

Annex A
(normative)
DOL Registry

OMG hosts a registry for DOL-conforming languages and translations. This registry will enable the use of other DOL-conforming languages than the ones that are discussed in this OMG Specification. 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.

OMG maintains the registry as an informative resource governed by the standard. The registry contents itself is informative.

This page intentionally left blank.

Annex B
(informative)
DOL Ontology

B.1 General

This annex describes the DOL ontology, which implements the terms and definitions from clause 4. While the ontology itself is informative, it is required for the forthcoming Application Programming Interfaces (APIs) for Knowledge Platforms (API4KP) specification, and so every effort has been made to provide a good foundation for that purpose.

B.2 Namespace Definitions

The namespaces and prefixes corresponding to external elements required for use by the DOL ontology are provided below. Table B.1 lists the prefixes and namespaces on which DOL depends.

Table B.1: Prefix and Namespaces for referenced/external vocabularies

Prefix	Namespace
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
owl	http://www.w3.org/2002/07/owl#
xsd	http://www.w3.org/2001/XMLSchema#
dct	http://purl.org/dc/terms/
skos	http://www.w3.org/2004/02/skos/core#
sm	http://www.omg.org/techprocess/ab/SpecificationMetadata/

The namespace approach taken for DOL is based on OMG guidelines and is constructed as follows:

— A standard prefix http://www.omg.org/spec/

— The abbreviation for the specification: in this case DOL

— The ontology name

Note that the URI/IRI strategy for the ontology takes a “slash” rather than “hash” approach, in order to accommodate server-side applications. Table B.2 provides the namespace definition for the DOL ontology. While the prefixes given in Tables B.1 and B.2 are informative, their use is required in any extension, including API4KP.

Table B.2: Prefix and Namespaces for the DOL ontology

Prefix	Namespace
dol	http://www.omg.org/spec/DOL/DOL-terms/

The ontology itself is not documented here, as it is informative, and the bulk of the ontology is documented in clause 4, Terms and Definitions, as stated above. Later versions of this specification may provide complete documentation, including machine-readable, ODM-compliant UML XMI, ODM XMI, and additional content such as "about" files, if usage of the ontology is more widespread than anticipated and thus such documentation is warranted.

Annex C
(informative)
Conformance of OWL 2 DL With DOL

C.1 General

The semantic conformance of OWL 2 DL (as specified in W3C/TR REC-owl2-syntax:2012) with DOL is established in [56].

C.2 Abstract Syntax Conformance of OWL 2 With DOL

The metaclass `OWLOntology` (**NR24**, 11.2) is a subclass (in the sense of SMOF multiple classification) of `NativeDocument`. The metaclass `OWLUniverse` (**NR24**, 11.7) is a subclass (in the sense of SMOF multiple classification) of `BasicOMS`.

C.3 Conformance of the OWL Serializations With DOL

C.3.1 Text Conformance of the OWL 2 Manchester Syntax With DOL

The OWL 2 Manchester syntax satisfies the criteria for text conformance established in clause 2.3 in a straightforward way thanks to its line-based comment syntax (comments starting with #) and its flexible handling of line breaks.

C.3.2 Conformance of the XML and RDF Serializations of OWL With DOL

C.3.2.1 General Issues

With minor modifications detailed below, the OWL/XML serialization [25] satisfies the criteria for XML conformance and the serialization of OWL in RDF (**NR20**) satisfies RDF the criteria for RDF conformance. Both modifications define a super-language of the respective OWL serialization. Any OWL ontology serialization S' in one of these two super-languages can be translated into an OWL ontology serialization S that fully conforms to the original specification OWL/XML or "OWL serialized in RDF" and is semantically equivalent to the extended serialization S' with regard to the semantics of OWL. Without these modifications, neither OWL/XML nor "OWL serialized in RDF" satisfies the XML or RDF conformance requirements, respectively. The reason is that with imports there is a structural element supported by OWL that cannot have identifiers nor carry annotations, and that these two OWL serializations do not permit the use of XML or RDF constructs that would enable assigning identifiers to imports.

C.3.2.2 XML Conformance of a Modified OWL/XML With DOL

In the OWL/XML serialization, the *Import* element does not have annotations and is only allowed to carry the attributes *xml:base*, *xml:lang* and *xml:space*, but no further attributes or child elements from foreign namespaces (requirement (3b)), and therefore in particularly not a *dol:id* attribute or child elements, as would be required for adding identifiers (cf. clause 9.9).

An extended specification of OWL/XML that does allow the *dol:id* attribute on *Import* satisfies the XML conformance criteria. From an ontology serialized in this super-language of OWL/XML, one can obtain a semantically equivalent ontology (with regard to the semantics of OWL) by stripping all *dol:id* attributes.

C.3.2.3 RDF Conformance of a Modified Serialization of OWL in RDF With DOL

The serialization of OWL in RDF (regardless of the concrete *RDF* serialization employed to serialize the RDF graph that represents the OWL ontology) does not satisfy requirement (2) for RDF conformance because 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 an RDF resource. However, only resources can have identifiers in RDF. RDF reification would allow for turning the statement O_1 `owl:imports` O_2 into a resource and thus giving it an identifier. However, the RDF triples required for expressing 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[32]. 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[33].

After extending the specification of the serialization of OWL in RDF in the following way, it satisfies the RDF conformance criteria: if the input RDF graph G considered in section 3 of **NR20** contains the pattern

```
i rdf:subject s .
i rdf:predicate owl:imports .
i rdf:object o .
```

and thus introduces a resource i to represent that the ontology s imports the ontology o, these three triples are removed from G. From an ontology serialized in this super-language of the serialization of OWL in RDF, one can obtain semantically equivalent ontologies (with regard to the semantics of OWL) by stripping all triples whose predicate is *rdf:subject*, *rdf:predicate* or *rdf:object*, or by adding triples that declare these three properties to be *annotation properties*.

C.4 Semantic Conformance of OWL 2 With DOL

The logic $\mathcal{SROIQ}$ underlying OWL can be formalized as an institution as follows:

Definition 15 ***OWL 2 DL.*** OWL *2 DL is the description logic (DL) based fragment of the web ontology language* OWL. *First, the simple description logic $\mathcal{ALC}$ is discussed, afterward the approach is generalized 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. Realizations 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*

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

[32] **NR20**, section 3

[33] See the last sentence of section 3.2.5 of **NR20**

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 $\mathcal{SROIQ}$ [28], which is the logical core of the Web Ontology Language OWL *2 DL*[34]*, 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 [28].*

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) .*

The logic $\mathcal{EL}$ is underlying the EL *profile. (To be exact,* EL *adds various 'harmless' expressive means and syntactic sugar to $\mathcal{EL}$ resulting in the DL $\mathcal{EL}$ ++.) $\mathcal{EL}$ is a syntactic restriction of $\mathcal{ALC}$ 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. □

OWL itself is more complicated than $\mathcal{SROIQ}$ due to the presence of datatypes. Following the direct model-theoretic semantics of OWL [61]:

Definition 16 *A datatype map, formalizing datatype maps from the OWL 2 Specification [62], is a 6-tuple*

$$D = (N_{DT}, N_{LS}, N_{FS}, \cdot^{DT}, \cdot^{LS}, \cdot^{FS})$$

with the following components:

— *N_{DT} is a set of datatypes (more precisely, names of datatypes) that does not contain the datatype rdfs:Literal.*

— *N_{LS} is a function that assigns to each datatype $DT \in N_{DT}$ a set $N_{LS}(DT)$ of strings called lexical forms. The set $N_{LS}(DT)$ is called the lexical space of DT.*

— *N_{FS} is a function that assigns to each datatype $DT \in N_{DT}$ a set $N_{FS}(DT)$ of pairs (F, v), where F is a constraining facet and v is an arbitrary data value called the constraining value. The set $N_{FS}(DT)$ is called the facet space of DT.*

— *For each datatype $DT \in N_{DT}$, the interpretation function $\cdot^{DT}$ assigns to DT a set $(DT)^{DT}$ called the value space of DT.*

— *For each datatype $DT \in N_{DT}$ and each lexical form $LV \in N_{LS}(DT)$, the interpretation function $\cdot^{LS}$ assigns to the pair (LV, DT) a data value $(LV, DT)^{LS} \in (DT)^{DT}$.*

— *For each datatype $DT \in N_{DT}$ and each pair $(F, v) \in N_{FS}(DT)$, the interpretation function $\cdot^{FS}$ assigns to (F, v) the set $(F, v)^{FS} \subseteq (DT)^{DT}$.*

The set of datatypes N_{DT} of a datatype map D is not required to contain all datatypes from the OWL 2 datatype map; this allows one to talk about subsets of the OWL 2 datatype map, which may be necessary for the various profiles of OWL 2. If, however, D contains a datatype DT from the OWL 2 datatype map, then $N_{LS}(DT)$, $N_{FS}(DT)$, $(DT)^{DT}$, $(LV, DT)^{LS}$ for each $LV \in N_{LS}(DT)$, and $(F, v)^{FS}$ for each $(F, v) \in N_{FS}(DT)$ are required to coincide with the definitions for DT in the OWL 2 datatype map. □

[34] See also http://www.w3.org/TR/owl2-overview/

Given two datatype maps $D = (N_{DT}, N_{LS}, N_{FS}, \cdot^{DT}, \cdot^{LS}, \cdot^{FS})$ and $D' = (N'_{DT}, N'_{LS}, N'_{FS}, \cdot^{DT'}, \cdot^{LS'}, \cdot^{FS'})$, we write $D \subseteq D'$ if $N_{DT} \subseteq N'_{DT}$, and the other components of D are restrictions (as functions) of those of D'.

Definition 17 *A vocabulary $V = (V_C, V_{OP}, V_{DP}, V_I, V_{DT}, V_{LT}, V_{FA})$ over a datatype map D is a 7-tuple consisting of the following elements:*

- *V_C is a set of classes as defined in the OWL 2 Specification [62], containing at least the classes owl:Thing and owl:Nothing.*
- *V_{OP} is a set of object properties as defined in the OWL 2 Specification [62], containing at least the object properties owl:topObjectProperty and owl:bottomObjectProperty.*
- *V_{DP} is a set of data properties as defined in the OWL 2 Specification [62], containing at least the data properties owl:topDataProperty and owl:bottomDataProperty.*
- *V_I is a set of individuals (named and anonymous) as defined in the OWL 2 Specification [62].*
- *V_{DT} is a set containing all datatypes of D, the datatype rdfs:Literal, and possibly other datatypes; that is, $N_{DT} \cup \{\textit{rdfs:Literal}\} \subseteq V_{DT}$.*
- *V_{LT} is a set of literals LV^{DT} for each datatype $DT \in N_{DT}$ and each lexical form $LV \in N_{LS}(DT)$.*
- *V_{FA} is the set of pairs (F, lt) for each constraining facet F, datatype $DT \in N_{DT}$, and literal $lt \in V_{LT}$ such that $(F, (LV, DT_1)^{LS}) \in N_{FS}(DT)$, where LV is the lexical form of lt and DT_1 is the datatype of lt.*

□

Definition 18 *Given a datatype map D and a vocabulary V over D, an interpretation*

$$I = (\Delta_I, \Delta_D, \cdot^C, \cdot^{OP}, \cdot^{DP}, \cdot^I, \cdot^{DT}, \cdot^{LT}, \cdot^{FA}, NAMED)$$

for D and V is a 10-tuple with the following structure:

- *Δ_I is a nonempty set called the object domain.*
- *Δ_D is a nonempty set disjoint with Δ_I called the data domain such that $(DT)^{DT} \subseteq \Delta_D$ for each datatype $DT \in V_{DT}$.*
- *$\cdot^C$ is the class interpretation function that assigns to each class $C \in V_C$ a subset $(C)^C \subseteq \Delta_I$ such that*
  - *$(\textit{owl:Thing})^C = \Delta_I$ and*
  - *$(\textit{owl:Nothing})^C = \emptyset$.*
- *$\cdot^{OP}$ is the object property interpretation function that assigns to each object property $OP \in V_{OP}$ a subset $(OP)^{OP} \subseteq \Delta_I \times \Delta_I$ such that*
  - *$(\textit{owl:topObjectProperty})^{OP} = \Delta_I \times \Delta_I$ and*
  - *$(\textit{owl:bottomObjectProperty})^{OP} = \emptyset$.*
- *$\cdot^{DP}$ is the data property interpretation function that assigns to each data property $DP \in V_{DP}$ a subset $(DP)^{DP} \subseteq \Delta_I \times \Delta_D$ such that*
  - *$(\textit{owl:topDataProperty})^{DP} = \Delta_I \times \Delta_D$ and*
  - *$(\textit{owl:bottomDataProperty})^{DP} = \emptyset$.*

— $\cdot^I$ *is the individual interpretation function that assigns to each individual* $a \in V_I$ *an element* $(a)^I \in \Delta_I$.

— $\cdot^{DT}$ *is the datatype interpretation function that assigns to each datatype* $DT \in V_{DT}$ *a subset* $(DT)^{DT} \subseteq \Delta_D$ *such that*

  — $\cdot^{DT}$ *is the same as in* D *for each datatype* $DT \in N_{DT}$, *and*

  — $(\textit{rdfs:Literal})^{DT} = \Delta_D$.

— $\cdot^{LT}$ *is the literal interpretation function that is defined as* $(lt)^{LT} = (LV, DT)^{LS}$ *for each* $lt \in V_{LT}$, *where* LV *is the lexical form of* lt *and* DT *is the datatype of* lt.

— $\cdot^{FA}$ *is the facet interpretation function that is defined as* $(F, lt)^{FA} = (F, (lt)^{LT})^{FS}$ *for each* $(F, lt) \in V_{FA}$.

— $NAMED$ *is a subset of* Δ_I *such that* $(a)^I \in NAMED$ *for each named individual* $a \in V_I$.

□

The institution $\mathcal{SROIQ}(D)$ underlying OWL is now defined as follows:

Definition 19 — *An* $\mathcal{SROIQ}(D)$ *signature is a pair* (D, V), *where* D *is a datatype map and* V *a vocabulary over* D.

— *Given* $\mathcal{SROIQ}(D)$ *signatures* (D, V) *and* (D', V'), *a* $\mathcal{SROIQ}(D)$ *signature morphism* $\sigma\colon (D, V) \to (D', V')$ *only exists if* $D \subseteq D'$. *In this case, such a signature morphism consists of*

  — *a map* $\sigma_C\colon V_C \to V'_C$,

  — *a map* $\sigma_{OP}\colon V_{OP} \to V'_{OP}$,

  — *a map* $\sigma_{DP}\colon V_{DP} \to V'_{DP}$,

  — *a map* $\sigma_I\colon V_I \to V'_I$,

  — *a map* $\sigma_{DT}\colon V_{DT} \to V'_{DT}$ *that is the identity on* $N_{DT} \cup \{\textit{rdfs:Literal}\}$,

  — *a map* $\sigma_{LT}\colon V_{LT} \to V'_{LT}$

— *The sentences for a signature are definded as in the direct model-theoretic semantics of OWL [61]. Sentence translation is substitution of symbols.*

— (D, V)*-realizations are interpretations for* D *and* V. *Morphisms of* (D, V)*-realizations are maps between the domains* Δ_I *preserving membership in classes and properties, where* Δ_D *is mapped identically. Reducts of realizations are built by first translating along the signature morphism and then looking up the interpretation in the realization to be reduced.*

— *The satisfaction relation is defined as in direct model-theoretic semantics of OWL [61].*

□

Remark: strictly speaking, the institution defined above is *OWL 2 DL without restrictions* in the sense of [67]. The reason is that in an institution, the sentences can be used for arbitrary 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.4.1 Relativization in OWL

Definition 20 *Given an* OWL *theory* $T = ((C, R, I), \Delta)$*, the* relativization *of* T*, denoted* $\tilde{T}$*, is the theory* $((C', R, I), \Delta')$ *where*

— $C' = C \cup \{\top_T\}$

— Δ' *contains axioms stating that:*

— *each concept in* C *is subsumed by* $\top_T$*,*

— *each individual in* I *is an instance of* $\top_T$*,*

— *each role* r *has its domain and range intersected with* $\top_T$*, if they are present in* Δ*, otherwise they are* $\top_T$*,*

and, for each sentence $e \in \Delta$*, the sentence* $\alpha(e)$*, obtained by replacing the concepts in* e *as follows: are made:*

— *each occurence of* $\top$ *is replaced with* $\top_T$*,*

— *each occurence of* $\neg C$ *is replaced with* $\top_T \sqcap \neg C$*,*

— *each occurence of* $\forall\ r \bullet C$ *is replaced with* $\top_T \sqcap \forall\ r \bullet C$*.*

Definition 21 *Given an* OWL *theory* $T = ((C, R, I), \Delta)$*, we define* $\beta : Mod^{\mathsf{OWL}}(\tilde{T}) \to Mod^{\mathsf{OWL}}(T)$ *as follows: if* $M' \in Mod^{\mathsf{OWL}}(\tilde{T})$*, then* $M = \beta(M')$ *has as universe* Δ^M *the set* $(\top_T)^{M'}$ *and each concept, role and individual are interpreted in* M *in the same way as in* M'*. Since* M' *is a* Δ'*-realization, we get that* M *is indeed a* (C, R, I)*-realization and moreover* $M \models \Delta$*.*

NOTE If $T = ((C, R, I), \Delta)$ is an OWL theory, M' is a $\tilde{T}$-realization and e is a (C, R, I)-sentence, we have that $M' \models \alpha(e)$ if and only if $\beta(M') \models e$.

C.4.2 Translating correspondences to a bridge theory in OWL

We define the function $theoryOfCorrespondences_{\mathsf{OWL}}$ that takes as arguments the assumption made on the semantics of the alignment where the correspondences come from, the signatures of the two ontologies being aligned and a list of processed correspondences, in the sense that the default correspondence and any correspondence block, if present, are replaced with the lists of single correspondences they induce, represented as triples of the form $(relation, sourceSymbol, targetSymbol)$. The result of the function is a co-span of theories: $(\Sigma_s, \emptyset) \xrightarrow{\varphi_s} (\Sigma, \Delta) \xleftarrow{\varphi_t} (\Sigma_t, \emptyset)$. Intuitively, Σ_s and Σ_t gather the symbols of the aligned ontologies that appear in the list of correspondences passed as an argument, while (Σ, Δ) contains an OWL sentence representing each correspondence.

We distinguish three cases.

1) Single domain:

— no other symbols occur in the signatures Σ_s and Σ_t than the ones that appear in correspondences: $\Sigma_s = (C_s, R_s, I_s)$ and $\Sigma_t = (C_t, R_t, I_t)$, where C_s, R_s and I_s are the sets of all concept names, roles and individuals that appear in the list of correspondences as source symbols and C_t, R_t and I_t are the sets of all concept names, roles and individuals that appear in the list of correspondences as target symbols.

— $\Sigma = \Sigma_s \uplus \Sigma_t$, where we prefix the symbols of Σ coming from Σ_s with $1:$ and those coming from Σ_t with $2:$

— φ_s maps each symbol s in Σ_s to $1:s$ and φ_t maps each symbol s in Σ_t to $2:s$.

— Δ contains the translation of correspondences to Σ-sentences using the following rules:

$('equivalent', c1, c2)$	`Class: 1:c1 EquivalentTo: 2:c2`
$('equivalent', r1, r2)$	`ObjectProperty: 1:r1 EquivalentTo: 2:r2`
$('equivalent', i1, i2)$	`Individual: 1:i1 SameAs: 2:i2`
$('incompatible', c1, c2)$	`Class: 1:c1 DisjointWith: 2:c2`
$('incompatible', r1, r2)$	`ObjectProperty: 1:r1 DisjointWith: 2:r2`
$('incompatible', i1, i2)$	`Individual: 1:i1 DifferentFrom: 2:i2`
$('subsumes', c1, c2)$	`Class: 2:c2 SubClassOf: 1:c1`
$('subsumes', r1, r2)$	`ObjectProperty: 2:r2 SubPropertyOf: 1:r1`
$('is\text{-}subsumed', c1, c2)$	`Class: 1:c1 SubClassOf: 2:c2`
$('is\text{-}subsumed', r1, r2)$	`ObjectProperty: 1:r1 SubPropertyOf: 2:r2`
$('has\text{-}instance', c1, i2)$	`Individual: 2:i2 Types: 1:c1`
$('instance\text{-}of', i1, c2)$	`Individual: 1:i1 Types: 2:c2`

2) Global domain:

— $\Sigma_s = (C_s \cup \top_s, R_s, I_s)$ and $\Sigma_t = (C_t \cup \top_t, R_t, I_t)$, where C_s, R_s and I_s are the sets of all concept names, roles and individuals that appear in the list of correspondences as source symbols and C_t, R_t and I_t are the sets of all concept names, roles and individuals that appear in the list of correspondences as target symbols.

— $\Sigma = \Sigma_s \uplus \Sigma_t$, where we prefix the symbols of Σ coming from Σ_s with $1:$ and those coming from Σ_t with $2:$

— φ_s maps each symbol s in Σ_s to $1:s$ and φ_t maps each symbol s in Σ_t to $2:s$.

— Δ is constructed in the same way as in the previous case, except that if `Thing` appears in a correspondence, it is replaced by $\top_s$ or $\top_t$.[35)]

3) Contextualized domain:

— Σ_s and Σ_t are constructed as before, but now they also include the relativized top concepts $\top_S$ and $\top_T$ respectively.

— Σ extends the disjoint union $\Sigma_s \uplus \Sigma_t$ with new roles r_{st} and r_{ts}.

— Δ contains the following axioms:

`ObjectProperty:` r_{st} `Domain:`$\top_S$ `Range:`$\top_T$

and

`ObjectProperty:` r_{ts} `Domain:`$\top_T$ `Range:`$\top_S$

together with the property that r_{st} is the converse of r_{ts}:

`ObjectProperty:` r_{st} `InverseOf:` r_{ts}

35) An extension of the language where complex concepts are allowed in alignments would make the construction of Δ in this case substantially different to the one in the previous case.

and translation of correspondences to Σ-sentences using the following rules:

$('equivalent', c1, c2)$	`Class: 1:c1 EquivalentTo:` r_{st} `some 2:c2`
$('equivalent', r1, r2)$	`ObjectProperty: 1:r1 EquivalentTo:` r_{st} `o 2:r2 o` r_{ts}
$('equivalent', i1, i2)$	`Individual: 1:i1 Facts:` r_{st} `2:i2`
$('incompatible', c1, c2)$	`EquivalentClasses:` $1:c1$ `and` r_{st} `some 2:c2, Nothing`
$('incompatible', r1, r2)$	`ObjectProperty: 1:r1 DisjointWith:` r_{st} `o 2:r2 o` r_{ts}
$('incompatible', i1, i2)$	`Individual: 1:i1 DifferentFrom: 2:i2`
$('subsumes', c1, c2)$	`Class: 2:c2 SubClassOf:` r_{sz} `some 1:c1`
$('subsumes', r1, r2)$	`ObjectProperty: 2:r2 SubPropertyTo:` r_{ts} `o 1:r1 o` r_{st}
$('is\text{-}subsumed', c1, c2)$	`Class: 1:c1 SubClassOf:` r_{ts} `some 2:c2`
$('is\text{-}subsumed', r1, r2)$	`ObjectProperty: 1:r1 SubPropertyTo:` r_{st} `o 2:r2 o` r_{ts}
$('has\text{-}instance', c1, i2)$	`Individual: 2:i2 Types:` r_{ts} `some 1:c1`
$('instance\text{-}of', i1, c2)$	`Individual: 1:i1 Types:` r_{st} `some 2:c2`

Note that we must express equivalences where role compositions are involved. This is only possible in OWL 2 Full. Thus the diagram returned by the function *theoryOfCorrespondences* becomes heterogeneous:

$$(\mathsf{OWL}, \Sigma_s, \emptyset) \xrightarrow{(OWL2Full, \varphi_s)} (\mathsf{OWL}_{FULL}), \Sigma, \Delta) \xleftarrow{(OWL2Full, \varphi_t)} (\mathsf{OWL}, \Sigma_t, \emptyset)$$

where $OWL2Full$ is the inclusion comorphism of OWL in OWL 2 Full, φ_s maps each symbol s to $1:s$ and $\top_s$ to itself, and φ_t maps each symbol t to 2_t and $\top_t$ to itself.

Annex D
(informative)
Conformance of Common Logic with DOL

D.1 Abstract Syntax Conformance of Common Logic With DOL

The metaclass `Text` (**NR24**, 12.2) is a subclass (in the sense of SMOF (**NR26**) multiple classification) of `NativeDocument`. The metaclass `Sentence` (**NR24**, 12.2) is a subclass (in the sense of SMOF (**NR26**) multiple classification) of `BasicOMS`.

D.2 Serialization Conformance of Common Logic With DOL

The semantic conformance of Common Logic (as specified in **NR7**) with DOL is established in [56].

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.

D.3 Semantic Conformance of Common Logic With DOL

Common Logic can be defined as an institution as follows:

Definition 22 ***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-realization $I = (UR, UD, rel, fun, int, seq)$ 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 $UD^* = \{\langle x_1, \ldots, x_n\rangle \mid x_1, \ldots, x_n \in UD\}$ (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.

Reducts of realizations are defined in the following way: Given a signature morphism $\sigma : \Sigma_1 \rightarrow \Sigma_2$ and a Σ_2-realization $I_2 = (UR, UD, rel, fun, int, seq)$, $I|_\sigma = (UR, UD, rel, fun, int \circ \sigma, seq \circ \sigma)$.

Given two CL *realizations* $I_1 = (UR_1, UD_1, rel_1, fun_1, int_1, seq_1)$ *and* $I_2 = (UR_2, UD_2, rel_2, fun_2, int_2, seq_2)$*, a homomorphism of realizations* $h : I_1 \to I_2$ *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* $k^*(s) \in rel_2(h(x))$[36]*,*

— *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*,* $seq_2(n) = k^*(seq_1(n))$*.*

CL$^-$ *is the restriction of* CL *to sentence without sequence markers.*□

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.

[36] k^* is the extension of h to sequences.

Annex E
(informative)
Conformance of RDF and RDF Schema with DOL

E.1 Abstract Syntax Conformance of RDF and RDF Schema With DOL

The metaclass `rdfDocument` (**NR24**, 14.2.2) is a subclass (in the sense of SMOF (**NR26**) multiple classification) of `NativeDocument`. The metaclass `graph` (**NR24**, 14.2.3) is a subclass (in the sense of SMOF (**NR26**) multiple classification) of `BasicOMS`.

E.2 Serialization Conformance of RDF and RDF Schema With DOL

The way of representing RDF Schema ontologies as RDF graphs satisfies the criteria for RDF conformance.

E.3 Semantic Conformance of RDF and RDF Schema With DOL

The semantic conformance of RDF Schema (as specified in **NR18**) with DOL is established in [56].

Definition 23 (RDF **and** RDF **Schema)** *The institutions for the Resource Description Framework (*RDF*) and* RDF *Schema (also known as* RDFS*), respectively, are defined in the following [44]. Both* RDF *and* RDFS *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* $\mathbf{R_s}$*-realization* $M = \langle R_m, P_m, S_m, EXT_m \rangle$ *consists of a set* R_m *of resources, a set* $P_m \subseteq R_m$ *of predicates, a mapping function* $S_m : \mathbf{R_s} \to R_m$*, and an extension function* $EXT_m : P_m \to \mathcal{P}(R_m \times R_m)$ *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 realizations.*

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.,*

$$(\mathtt{rdf:type}, \mathtt{rdf:type}, \mathtt{rdf:Property}) \text{ , and}$$
$$(\mathtt{rdf:subject}, \mathtt{rdf:type}, \mathtt{rdf:Property}) \text{ .}$$

In the realizations, the standard vocabulary is interpreted with a fixed realization. Moreover, for each RDF*-realization* $M = \langle R_m, P_m, S_m, EXT_m \rangle$*, if* $p \in P_m$*, then it must hold* $(p, S_m(\mathtt{rdf:Property})) \in EXT_m(\mathtt{rdf:type})$*. 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 *[24].* □

This page intentionally left blank.

Annex F
(informative)
Conformance of UML class and object models with DOL

F.1 General

This informative annex demonstrates conformance of a subset of UML class and object models with DOL by defining an institution for both. The subset is restricted to the static aspects of class models; that is, change of state is ignored. This means that all operations are query operations.

F.2 Abstract Syntax Conformance of UML With DOL

The metaclass `Package` (**NR8**) is a subclass (in the sense of SMOF (**NR26**) multiple classification) of `NativeDocument`. The metaclass `PackageableElement` (**NR8**) is a subclass (in the sense of SMOF (**NR26**) multiple classification) of `BasicOMS`.

F.3 Serialization Conformance of UML With DOL

The XMI (**NR27**) serialization, derived from the MOF metamodel, is widely used for UML. Hence, UML is serialization conformant with DOL.

F.4 Semantic Conformance of UML With DOL

The institution of UML class and object models is defined using a translation of UML class models to Common Logic, following the fUML specification and [68].

F.4.1 Preliminaries

The axioms for primitive types are imported from the fUML specification, section 10.3.1: 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`.

The following infrastructure, consisting off a number of predicates axiomatized in Common Logic, provides a foundation for an institution for UML class models described in the later sections of this Annex.

```
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
```

```
  // 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 (p)
                       (and (form:in-sequence s p)
                            (form:in-position p 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)
          (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))))
  (forall (x s1 s2)
          (iff (= (form:ordered-set-insert x s1) s2)
               (and // No element can be inserted twice
                    (if (from:ordered-set-member x s1)
                        (form:same-ordered-set s1 s2))
                    // Inserting a new element
                    (if (not (from:ordered-set-member x s1))
                        (and // 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.
  (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
```

F.4.2 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 *generalization relation* on C, where 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 $c \in \{\mathsf{Boolean}, \mathsf{UnlimitedNatural}, \mathsf{Integer}, \mathsf{Real}, \mathsf{String}\}$.

The *collection type constructors* OrderedSet, Set, Sequence, and Bag are used for representing the meta-attributes "ordered" and "unique" of MultiplicityElement according to the following table:[37)]

	ordered	not ordered
unique	OrderedSet	Set
not unique	Sequence	Bag

The default is "not ordered" and "unique".[38)]

For a class/data type $c \in C$ of a class/data type-hierarchy $(C, \leq_C)$ and a collection type constructor

$$\tau \in \{\mathsf{OrderedSet}, \mathsf{Set}, \mathsf{Sequence}, \mathsf{Bag}\},$$

37) [63, p. 34].

38) [63, p. 33].

the expression $\tau[c]$ denotes the induced *collection type*.

Let $(C, \leq_C)$ be a class/data type hierarchy.

– An *attribute declaration*[39] 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*. Additionally, an attribute may be *composite* and we write $c{\blacklozenge}p : \tau[c']$ if this fact plays a rôle. (Attributes and association member ends are distinguished due to their different uses. In UML, both are of class Property. Hence, attribute declarations are a kind of *property declarations*. Another kind of property declaration will be introduced through member end declarations below.)

– 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*.[40] 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 $r = 2$.[41] For a binary association with $\tau_1 = \mathsf{Set}$, the second member end may be *composite*, and we write $a(p_1 : \mathsf{Set}[c_1], {\blacklozenge}p_2 : \tau_2[c_2])$ if this fact plays a rôle.[42]

Class/data type nets (Signatures). A *class/data type net* $\Sigma = ((C, \leq_C), P, O, A)$ comprises a class/data type hierarchy $(C, \leq_C)$ and a set P of attribute declarations, a set O of operation declarations, and a set A of association declarations over $(C, \leq_C)$, such that the following properties are satisfied:

– attribute names are unique along the generalization relation: if $c_1.p_1 : \tau_1[c'_1]$ and $c_2.p_2 : \tau_2[c'_2]$ are different property declarations in P and $c_1 \leq_C c_2$, then $p_1 \neq p_2$;

– association names are unique: if d_1 and d_2 are the names of two different association declarations in 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, then $p_i \neq p_j$ for $1 \leq i \neq j \leq r$;[43]

– the type of a member end[44] 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 $c_0.p_i : \tau_i[c_i] \in P$, where for the second end $\mathbf{a}.p_2 : \tau_2[c_2]$ of an association declaration $\mathbf{a} = a(p_1 : \mathsf{Set}[c_1], {\blacklozenge}p_2 : \tau_2[c_2])$ the property has to be composite, i.e., $c_0{\blacklozenge}p_2 : \tau_2[c_2]$. (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* $\sigma = (\gamma, \varphi, \alpha) : \Sigma = ((C, \leq_C), P, A) \to \mathrm{T} = ((D, \leq_D), Q, B)$ 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 $\tau = \tau'$; furthermore, each composite attribute has to be mapped to a composite attribute.

[39] We separate attributes from association member ends due to their different uses. In UML, both are of class Property ([63, p. 109]).

[40] The member ends are ordered [63, p. 197] hence they are represented in a tuple-like notation.

[41] Only binary association may show member ends that are properties not owned by the association [63, p. 218]. The property declarations induced by a more than binary association result in a query operation.

[42] Composite properties, i.e., properties with aggregation kind composite can only be member ends of binary associations [63, p. 218] and their multiplicity must not exceed one [63, p. 150].

[43] 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, this constraint is added. 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 $1, \dots, r$.

[44] All member ends are instances of Property [63, p. 206].

– 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 $1 \le i \le r$, and member ends owned by the association 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 Figure F.1 the class/data type net is

$$
\begin{aligned}
\text{Classes/data types: } & \mathsf{Net}, \mathsf{Station}, \mathsf{Line}, \mathsf{Connector}, \mathsf{Unit}, \mathsf{Track}, \mathsf{Point}, \mathsf{Linear}, \\
& \mathsf{Boolean}, \mathsf{UnlimitedNatural}, \mathsf{Integer}, \mathsf{Real}, \mathsf{String} \\
\text{Generalizations: } & \mathsf{Point} \le \mathsf{Unit}, \mathsf{Linear} \le \mathsf{Unit} \\
\text{Properties: } & \mathsf{Line.linear} : \mathsf{Set}[\mathsf{Boolean}], \mathsf{Track.linear} : \mathsf{Set}[\mathsf{Boolean}], \\
& \mathsf{Net}\blacklozenge\mathsf{station} : \mathsf{Set}[\mathsf{Station}], \mathsf{Net}\blacklozenge\mathsf{line} : \mathsf{Set}[\mathsf{Line}], \\
& \mathsf{Station.net} : \mathsf{Set}[\mathsf{Net}], \mathsf{Station}\blacklozenge\mathsf{unit} : \mathsf{Set}[\mathsf{Unit}], \mathsf{Station}\blacklozenge\mathsf{track} : \mathsf{Set}[\mathsf{Track}], \\
& \mathsf{Line.net} : \mathsf{Set}[\mathsf{Net}], \mathsf{Line.linear} : \mathsf{Set}[\mathsf{Linear}], \\
& \mathsf{Connector.unit} : \mathsf{Set}[\mathsf{Unit}], \\
& \mathsf{Unit.station} : \mathsf{Set}[\mathsf{Station}], \mathsf{Unit.connector} : \mathsf{Set}[\mathsf{Connector}], \\
& \mathsf{Track.station} : \mathsf{Set}[\mathsf{Station}], \mathsf{Track.linear} : \mathsf{Set}[\mathsf{Linear}], \\
& \mathsf{Linear.track} : \mathsf{Set}[\mathsf{Track}], \mathsf{Linear.line} : \mathsf{Set}[\mathsf{Line}] \\
\text{Associations: } & \mathsf{L2L}(\mathsf{line} : \mathsf{Set}[\mathsf{Line}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]), \\
& \mathsf{L2T}(\mathsf{linear} : \mathsf{Set}[\mathsf{Linear}], \mathsf{track} : \mathsf{Set}[\mathsf{Track}]), \\
& \mathsf{C2U}(\mathsf{connector} : \mathsf{Set}[\mathsf{Connector}], \mathsf{unit} : \mathsf{Set}[\mathsf{Unit}]) \\
& \mathsf{N2S}(\mathsf{net} : \mathsf{Set}[\mathsf{Net}], \blacklozenge\mathsf{station} : \mathsf{Set}[\mathsf{Station}]), \\
& \mathsf{N2L}(\mathsf{net} : \mathsf{Set}[\mathsf{Net}], \blacklozenge\mathsf{line} : \mathsf{Set}[\mathsf{Line}]), \\
& \mathsf{S2U}(\mathsf{station} : \mathsf{Set}[\mathsf{Station}], \blacklozenge\mathsf{unit} : \mathsf{Set}[\mathsf{Unit}]), \\
& \mathsf{S2T}(\mathsf{station} : \mathsf{Set}[\mathsf{Station}], \blacklozenge\mathsf{track} : \mathsf{Set}[\mathsf{Track}])
\end{aligned}
$$

Here all member ends are owned by class/data types.

F.4.3 Realizations

As stated above, realizations of UML class models are obtained via a translation to Common Logic.

For a classifier net $\Sigma = ((C, \le_C), P, O, A)$, a Common Logic theory $\mathsf{CL}(\Sigma)$ is defined consisting of:

– for $c \in C$, a predicate[45] $\mathsf{CL}(c)$, 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`,

[45] Strictly speaking, this is just a name.

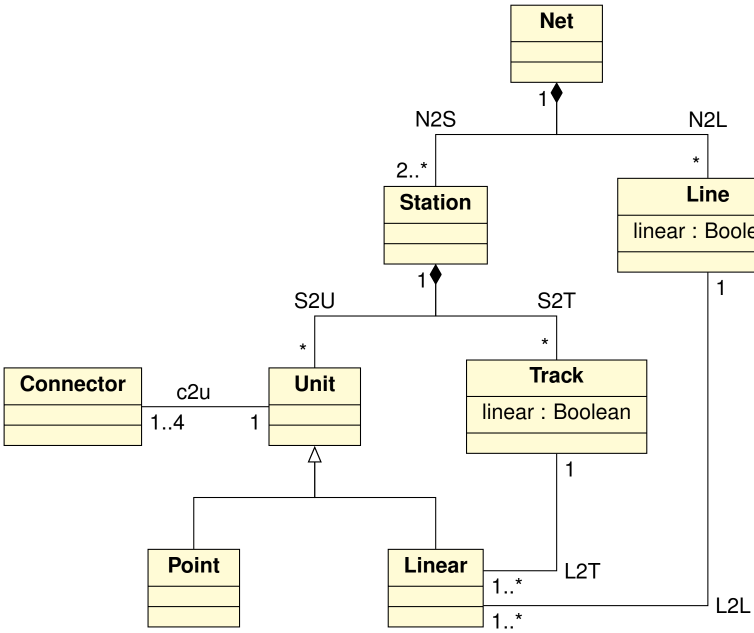


Figure F.1: Sample UML class model

- $\mathsf{CL}(c) = c$, 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 k_j)) for $i \neq j$
  (**forall** (x) (**if** ($\mathsf{CL}(c)$ x) (**or** (= x k_1) $\cdots$ (= x k_n))))

- $\mathsf{CL}(\mathsf{List}[c]) = \mathsf{CL}(\mathsf{List}) =$ `form:Sequence`,

- $\mathsf{CL}(\mathsf{Set}[c]) = \mathsf{CL}(\mathsf{Set}) =$ `form:Set`,

- $\mathsf{CL}(\mathsf{OrderedSet}[c]) = \mathsf{CL}(\mathsf{OrderedSet}) =$ `form:OrderedSet`,

- $\mathsf{CL}(\mathsf{Bag}[c]) = \mathsf{CL}(\mathsf{Bag}) =$ `form:Bag`,

- $\mathsf{CL}(c) = c$, 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** ($\mathsf{CL}(c_1)$ x) ($\mathsf{CL}(c_2)$ x)))

- 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** ($\mathsf{CL}(c.p)$ x y) ($\mathsf{CL}(c)$ x)))
  (**forall** (x y) (**if** ($\mathsf{CL}(c.p)$ x y) ($\tau[c']$ y))) [46)]
  (**forall** (x) (**if** ($\mathsf{CL}(c)$ x) (**exists** (y) ($\mathsf{CL}(c.p)$ x y))))
  (**forall** (x y z) (**if** (**and** ($\mathsf{CL}(c.p)$ x y) ($\mathsf{CL}(c.p)$ 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:

  (**forall** (x x_1 x_2 $\cdots$ x_n y) (**if** ($\mathsf{CL}(c.q)$ x x_1 x_2 $\cdots$ x_n y) ($\mathsf{CL}(c)$ x)))
  (**forall** (x x_1 x_2 $\cdots$ x_n y) (**if** ($\mathsf{CL}(c.q)$ x x_1 x_2 $\cdots$ x_n y) ($\tau_i[c_i]$ x_i))) for each $i = 1, \ldots, n$ [47)]

[46)] ($\tau[c']$ y) is an abbreviation of either (**and** ($\mathsf{CL}(\tau)$ y) (**forall** (m) (**if** (from:$\mathsf{CL}(\tau)$-member m y) ($\mathsf{CL}(c')$ m)))) (if τ is present) or just (c' y) (if τ is omitted).

```
(forall (x x₁ x₂ ··· xₙ y) (if (CL(c.q) x x₁ x₂ ··· xₙ y) (τ[c'] y)))
(forall (x x₁ x₂ ··· xₙ y z)
        (if (and (CL(c.q) x x₁ x₂ ··· xₙ y) (CL(c.q) x x₁ x₂ ··· xₙ z))
            (= y z)))
```

Query operations are modeled as partial functions: they may be undefined for certain arguments due to violation of multiplicity constraints.

– 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)[48]:

```
(from:Sequence CL(a))
(forall (t)
        (if (form:sequence-member t CL(a))
            (exists (x₁ ··· xᵣ)
                    (and (CL(c₁) x₁) ··· (CL(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 r-ary predicate:

```
(forall (x₁ x₂ ··· xᵣ)
        (if (CL(a) x₁ x₂ ··· xᵣ) (and (CL(c₁) x₁) ··· (CL(cᵣ) xᵣ))))
```

– 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 (CL(c.p) o s)
            (= s (form:seq2CL(τᵢ) (form:selecti o CL(a))))))
```

If $\mathbf{a}$ is represented in simplified form, then instead the following is used

```
(forall (o s)
        (if (CL(c.p) o s)
            (forall (x) (iff (member x s) (CL(a) o x)))))
```

– For the compositions, let $c^1 \blacklozenge p^1 : \tau[c'^1], \dots, c^k \blacklozenge p^k : \tau[c'^k]$ be all the composite attributes in P and $\mathbf{a}^1 = a^1(p_1^1 : \mathsf{Set}[c_1^1], \blacklozenge p_2^1 : \tau_2^{(1)}[c_2^1]), \dots, \mathbf{a}^l = a^{(l)}(p_1^l : \mathsf{Set}[c_1^l], \blacklozenge p_2^l : \tau_2^{(1)}[c_2^l])$ all the composite binary associations in A. Abbreviate

```
(or (CL(c¹.p¹) o x) ··· (CL(cᵏ.pᵏ) o x)
    (form:sequence-member (form:pair o x) CL(a¹)) ···
      (form:sequence-member (form:pair o x) CL(aˡ))))
```

by `(owner o x)`, where, for each binary association a^j represented in the simplified way, `(CL(aʲ) o i)` replaces `(form:sequence-member (form:pair o x) CL(aʲ))`. Then

[47] Note that the ··· here is meta notation, not a sequence marker.

[48] ignoring the annotations τ_i in the interpretation of an association is intentional [63, p. 197]: “a link is a tuple with one value for each memberEnd of the association, where each value is an instance whose type conforms to or implements the type of the end. [...] 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.” The additional information required for links is covered by using sequences of tuples.

```
(forall (o1 o2 x)
        (if (and (owner o1 x) (owner o2 x))
            (= o1 o2)))
```

It is straightforward to extend CL from signatures to signature morphisms.

Realizations. A Σ-realization of the UML class model institution is just a $\mathsf{CL}(\Sigma)$-realization in Common Logic. That is, the UML class model institution inherits realizations from Common Logic. Moreover, reducts of realizations are inherited as well, using the action of CL on signature morphisms.

F.4.4 Sentences

The set of *multiplicity formulae* *Frm* is given by the following grammar:

$$
\begin{array}{rcl}
\mathit{Frm} & ::= & \mathit{NumLiteral} \leq \mathit{FunExpr} \\
 & | & \mathit{FunExpr} \leq \mathit{NumLiteral} \\
\mathit{FunExpr} & ::= & \#\ \mathit{Attribute} \\
 & | & \#\ \mathit{Association}\ .\ \mathit{End} \\
 & | & \#\ \mathit{Operation}\ .\ \mathit{Param} \\
\mathit{Attribute} & ::= & \mathit{Classifier}\ .\ \mathit{End}\ :\ \mathit{Type} \\
 & | & \mathit{Classifier} \blacklozenge \mathit{End}\ :\ \mathit{Type} \\
\mathit{Association} & ::= & \mathit{Name}\ (\ \mathit{End} : \mathit{Type}(\ ,\ \mathit{End}\ :\ \mathit{Type})^*\ ) \\
 & | & \mathit{Name}\ (\ \mathit{End} : \mathsf{Set}\ [\ \mathit{Classifier}\ ], \blacklozenge\mathit{End}\ :\ \mathit{Type}\ ) \\
\mathit{Operation} & ::= & \mathit{Name}\ (\ (\ \mathit{NumLiteral} \leq \mathit{Param} \leq \mathit{NumLiteral}\!:\ \mathit{Type},\ )^*\ ) : \mathit{Type} \\
\mathit{Type} & ::= & \mathit{Annot}\ [\ \mathit{Classifier}\ ] \\
\mathit{Classifier} & ::= & \mathit{Name} \\
\mathit{End} & ::= & \mathit{Name} \\
\mathit{Param} & ::= & \mathit{Name} \\
\mathit{Annot} & ::= & \mathsf{OrderedSet} \mid \mathsf{Set} \mid \mathsf{Sequence} \mid \mathsf{Bag} \\
\mathit{NumLiteral} & ::= & 0 \mid 1 \mid \cdots
\end{array}
$$

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* 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, 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 Figure F.1 there are the following multiplicity formulas:

$$
\begin{gathered}
2 \leq \#\mathsf{N2S}(\mathsf{net} : \mathsf{Set}[\mathsf{Net}], \blacklozenge\mathsf{station} : \mathsf{Set}[\mathsf{Station}]).\mathsf{station} \\
\#\mathsf{N2S}(\mathsf{net} : \mathsf{Set}[\mathsf{Net}], \blacklozenge\mathsf{station} : \mathsf{Set}[\mathsf{Station}]).\mathsf{net} = 1 \\
\#\mathsf{N2L}(\mathsf{net} : \mathsf{Set}[\mathsf{Net}], \blacklozenge\mathsf{line} : \mathsf{Set}[\mathsf{Line}]).\mathsf{net} = 1 \\
\#\mathsf{S2U}(\mathsf{station} : \mathsf{Set}[\mathsf{Station}], \blacklozenge\mathsf{unit} : \mathsf{Set}[\mathsf{Unit}]).\mathsf{station} = 1 \\
\#\mathsf{S2T}(\mathsf{station} : \mathsf{Set}[\mathsf{Station}], \blacklozenge\mathsf{track} : \mathsf{Set}[\mathsf{Track}]).\mathsf{station} = 1 \\
1 \leq \#\mathsf{C2U}(\mathsf{connector} : \mathsf{Set}[\mathsf{Connector}], \mathsf{unit} : \mathsf{Set}[\mathsf{Unit}]).\mathsf{unit} \leq 4 \\
\#\mathsf{C2U}(\mathsf{connector} : \mathsf{Set}[\mathsf{Connector}], \mathsf{unit} : \mathsf{Set}[\mathsf{Unit}]).\mathsf{connector} = 1
\end{gathered}
$$

$$1 \leq \#\mathsf{L2T}(\mathsf{track} : \mathsf{Set}[\mathsf{Track}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]).\mathsf{track}$$
$$\#\mathsf{L2T}(\mathsf{track} : \mathsf{Set}[\mathsf{Track}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]).\mathsf{linear} = 1$$
$$1 \leq \#\mathsf{L2T}(\mathsf{line} : \mathsf{Set}[\mathsf{Line}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]).\mathsf{line}$$
$$\#\mathsf{L2L}(\mathsf{line} : \mathsf{Set}[\mathsf{Line}], \mathsf{linear} : \mathsf{Set}[\mathsf{Linear}]).\mathsf{linear} = 1$$

"$x = y$" is an abbreviation for the two inequations "$x \leq y$" and "$y \leq x$". "$x \leq y \leq z$" is an abbreviation for the two inequations "$x \leq y$" and "$y \leq z$".

F.4.5 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 model Σ, a multiplicity formula φ and a Σ-realization M (the latter amounts to a $\mathsf{CL}(\Sigma)$-realization M in Common Logic):

$$M \models_\Sigma \varphi \iff M \models_{\mathsf{CL}(\Sigma)} \mathsf{CL}(\varphi)$$

The translation of multiplicity formulas to Common Logic is as follows:

$\mathsf{CL}(\ell \leq \#c.p : \tau[c']) = \mathsf{CL}(\ell \leq \#c\blacklozenge p : \tau[c']) =$

```
  (forall (x y n)
          (if (and (CL(c.p) x y) (form:CL(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 (x1 ··· x(i-1) x(i+1) ··· xr n)
          (if (and (CL(c1) x1) ··· (CL(c(i-1)) x(i-1)) (CL(c(i+1)) x(i+1)) ··· (CL(cr) xr)
                   (form:sequence-size (form:n-select a i [x1 ··· x(i-1) x(i+1) ··· xr]) n))
              (buml:leq ⟦ℓ⟧ n)))
```

If a is represented in simplified form, the following is used instead:

$\mathsf{CL}(\ell \leq \#a(p_1 : \tau_1[c_1], \ldots, p_r : \tau_r[c_r]).p_i =$

```
  (forall (x1 ··· x(i-1) x(i+1) ··· xr)
          (if (and (CL(c1) x1) ··· (CL(c(i-1)) x(i-1)) (CL(c(i+1)) x(i+1)) ··· (CL(cr) xr))
              (exists (y1 ··· y⟦ℓ⟧)
                      (and (not (= (y1 y2))) ··· (not (= (y(⟦ℓ⟧-1) y⟦ℓ⟧)))
                           (CL(a) x1 ··· x(i-1) y1 x(i+1) ··· xr) ···
                           (CL(a) x1 ··· x(i-1) y⟦ℓ⟧ x(i+1) ··· xr)))))
```

$\mathsf{CL}(\ell \leq \#a(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2]).p_i) =$

```
  (forall (x n)
          (if (and (CL(c(3-i)) x) (form:CL(τ)-size (form:selecti x CL(a)) n))
              (buml:leq ⟦ℓ⟧ n)))
```

If a is represented in simplified form, the following is used instead:

$\mathsf{CL}(\ell \leq \#a(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2]).p_1) =$

```
  (forall (x)
          (if (CL(c2) x)
              (exists (y1 ··· y⟦ℓ⟧)
                      (and (not (= (y1 y2))) ··· (not (= (y(⟦ℓ⟧-1) y⟦ℓ⟧)))
```

$$
\begin{array}{l}
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{(}\mathsf{CL}(a)\ \texttt{y}_1\ \texttt{x)}\ \cdots\ \texttt{(}\mathsf{CL}(m)\ \texttt{y}_{[\![\ell]\!]}\ \texttt{x)))))} \\
\mathsf{CL}(\ell \leq \#a(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2]).p_2) = \\
\quad \texttt{(}\textbf{forall}\ \texttt{(x)} \\
\qquad\qquad \texttt{(}\textbf{if}\ \texttt{(}\mathsf{CL}(c_1)\ \texttt{x)} \\
\qquad\qquad\qquad \texttt{(}\textbf{exists}\ \texttt{(y}_1\ \cdots\ \texttt{y}_{[\![\ell]\!]}\texttt{)} \\
\qquad\qquad\qquad\qquad\qquad \texttt{(}\textbf{and}\ \texttt{(}\textbf{not}\ \texttt{(= (y}_1\ \texttt{y}_2\texttt{)))}\ \cdots\ \texttt{(}\textbf{not}\ \texttt{(= (y}_{[\![\ell]\!]-1}\ \texttt{y}_{[\![\ell]\!]}\texttt{)))} \\
\qquad\qquad\qquad\qquad\qquad\qquad \texttt{(}\mathsf{CL}(a)\ \texttt{x}\ \texttt{y}_1\texttt{)}\ \cdots\ \texttt{(}\mathsf{CL}(a)\ \texttt{x}\ \texttt{y}_{[\![\ell]\!]}\texttt{)))))}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{CL}(\ell \leq \#c.q(\ell_1 \leq f_1 \leq \ell'_1 : \tau_1[c_1], \ldots, \ell_k \leq f_k \leq \ell'_k : \tau_k[c_k]) : \tau[c']) = \\
\quad \texttt{(}\textbf{forall}\ \texttt{(x x}_1\ \texttt{x}_2\ \cdots\ \texttt{x}_n\ \texttt{n}_1\ \cdots\ \texttt{n}_k\ \texttt{n)} \\
\qquad\qquad \texttt{(}\textbf{if}\ \texttt{(}\textbf{and}\ \texttt{(}\mathsf{CL}(c.q)\ \texttt{x x}_1\ \texttt{x}_2\ \cdots\ \texttt{x}_n\ \texttt{y)} \\
\qquad\qquad\qquad\qquad \texttt{(form:}\mathsf{CL}(\tau)\texttt{-size x}_1\ \texttt{n}_1\texttt{)}\ \cdots\ \texttt{(form:}\mathsf{CL}(\tau)\texttt{-size}\ x_k\ n_k\texttt{)} \\
\qquad\qquad\qquad\qquad \texttt{(form:}\mathsf{CL}(\tau)\texttt{-size}\ y\ \texttt{n)} \\
\qquad\qquad\qquad\qquad \texttt{(buml:leq}\ [\![\ell_1]\!]\ \texttt{n}_1\texttt{)}\ \texttt{(buml:leq n}_1\ [\![\ell'_1]\!]\texttt{)}\ \cdots \\
\qquad\qquad\qquad\qquad \texttt{(buml:leq}\ [\![\ell_k]\!]\ \texttt{n}_k\texttt{)}\ \texttt{(buml:leq n}_k\ [\![\ell'_k]\!]\texttt{))} \\
\qquad\qquad\qquad\qquad \texttt{(buml:leq}\ [\![\ell]\!]\ \texttt{n)))}
\end{array}
$$

where $[\![-]\!] : NumLit \to \mathbb{Z}$ maps a numerical literal to an integer, and $[x_1 \cdots x_n]$ abbreviates `(form:sequence-insert` x_1 `···` `(form:sequence-insert` x_n `form:empty-sequence)···)`. The translation for *FunExpr* $\leq$ *NumLiteral* is analogous. In case of simplified representation, the existence of $[\![\ell]\!]$ distinct individuals would be replaced with a statement expressing that if $[\![\ell]\!] + 1$ individuals have the specified property, at least two of them must be equal.

Annex G
(informative)
Conformance of TPTP with DOL

G.1 General

TPTP [72, 74, 73] 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.

G.2 Abstract Syntax Conformance of TPTP With DOL

The BNF nonterminal `TPTP_file` of the TPTP concrete syntax [71] is construed as a metaclass, and as such it is a subclass (in the sense of SMOF (**NR26**) multiple classification) of `NativeDocument`. The BNF nonterminal `annotated_formula` of the TPTP concrete syntax [71] is construed as a metaclass, and as such is a subclass (in the sense of SMOF (**NR26**) multiple classification) of `BasicOMS`.

G.3 Serialization Conformance of TPTP With DOL

The TPTP text syntax is text conformant with DOL.

G.4 Semantic Conformance of TPTP With DOL

In [18], many-sorted first-order logic 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).

This page intentionally left blank.

Annex H
(informative)
Conformance of CASL with DOL

H.1 General

CASL [10] extends many-sorted first-order logic with partial functions and subsorting. It also provides induction sentences, expressing the (free) generation of datatypes.

H.2 Abstract Syntax Conformance of CASL With DOL

The EBNF nonterminal `LIBRARY` for the CASL abstract syntax [10] is construed as a metaclass, and as such it is a subclass (in the sense of SMOF (**NR26**) multiple classification) of `NativeDocument`. The EBNF nonterminal `BASIC_SPEC` for the CASL abstract syntax [10] is construed as a metaclass, and as such it is a subclass (in the sense of SMOF (**NR26**) multiple classification) of `BasicOMS`.

H.3 Serialization Conformance of CASL With DOL

The CASL text syntax is text conformant with DOL.

H.4 Semantic Conformance of CASL With DOL

CASL has been presented as an institution in [52, 10]. This section presents a sketch of 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.

Realizations 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 realization M if the carriers of the reduct of M along σ' of the sorts in S' are generated by function symbols in F'.

This page intentionally left blank.

Annex I
(informative)
A Core Logic Graph

I.1 General

This annex provides a core heterogeneous environment that could be used as a basis for semantics of DOL as defined in Sec. 10.

I.2 Languages

The selected OMS languages are those whose conformance with DOL is established in the preceding annexes (OWL 2 DL in annex C, Common Logic in annex D, RDFS in annex E, CASL in annex H, UML class models in annex F and TPTP in annex G). The logic graph is shown in Figure I.2; the language graph and supports relation in Figure I.1. Its nodes refer to the following OMS languages and profiles:

— RDF **NR14**

— RDF Schema **NR18**

— EL, QL, RL (all being profiles of OWL) **NR6**

— OWL **NR2**

— CL (Common Logic) **NR7**

— UML class models **NR8**, version 2.5

— CASL [10] and its sublanguage classical first-order logic (FOL)

— TPTP

The list of language translations, given below, comprises standard translations from the literature [56, 53], as well as further translations that are considered useful for logical interoperability:

— EL → OWL

— QL → OWL

— RL → OWL

— RDF → RDFS

— RDFS → OWL

— OWL → CASL.**FOL**

— CASL.**FOL** → *TPTP*

— *TPTP* → CASL.**FOL**

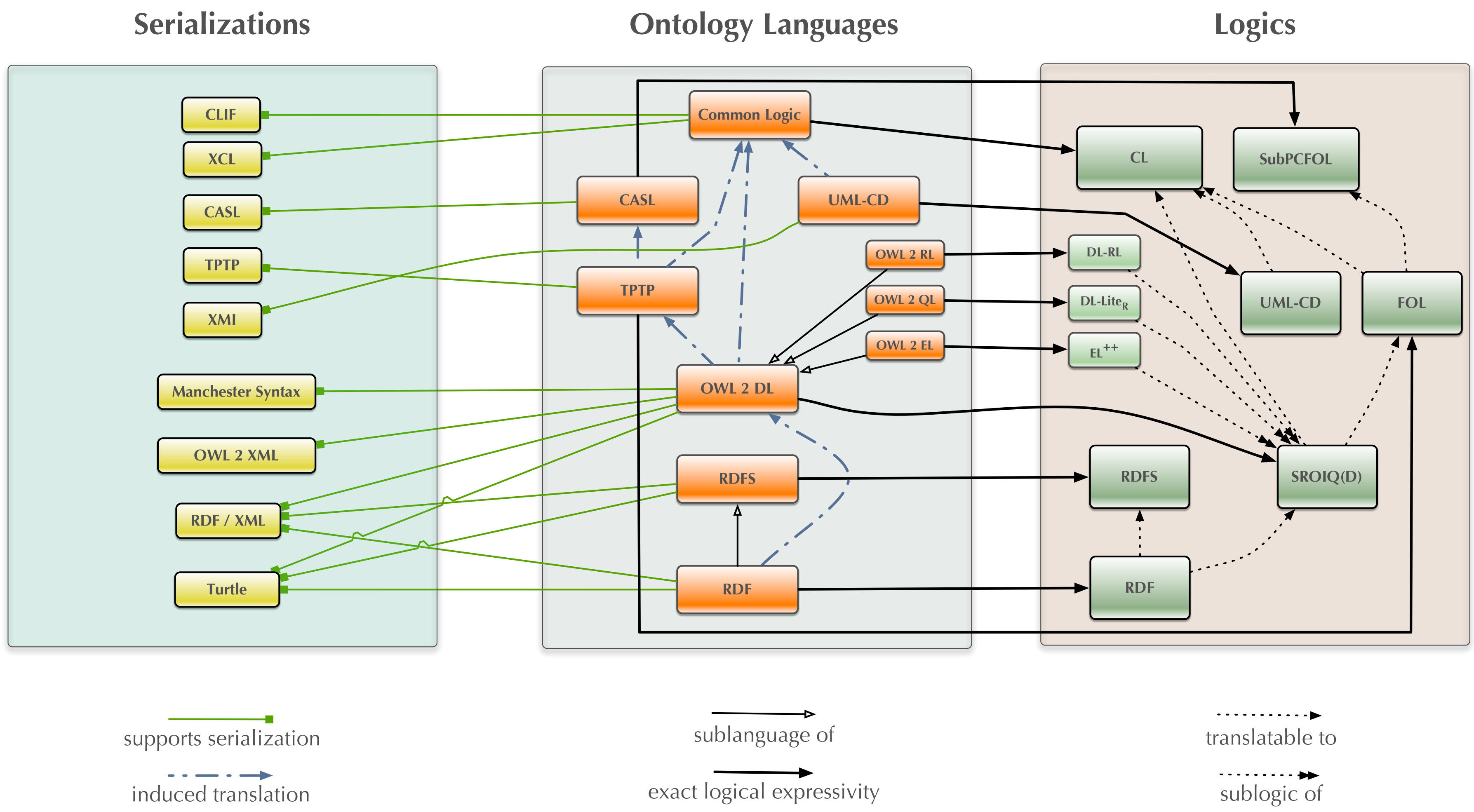


Figure I.1: Subset of the OntoIOp registry, shown as an RDF graph

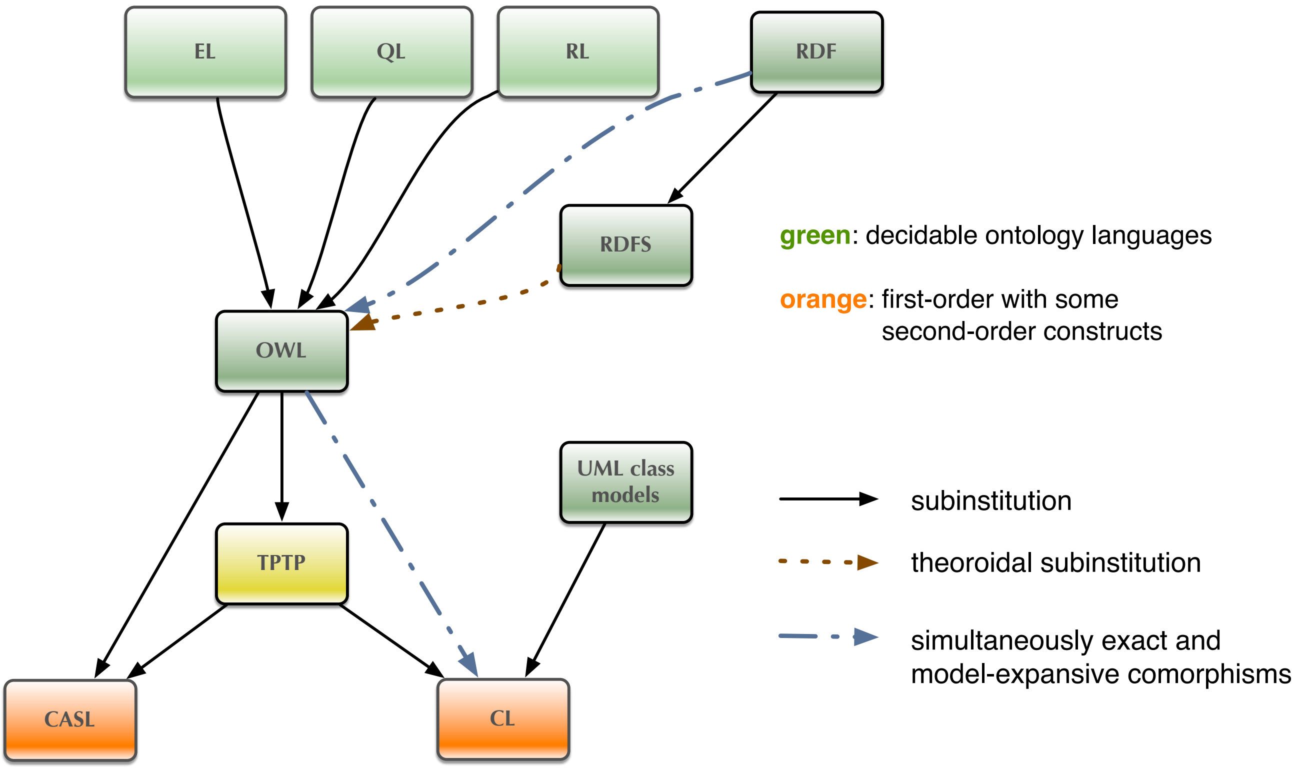


Figure I.2: Translations between conforming OMS languages

— CASL.**FOL** → CL

— CASL.**FOL** → CASL

— *UML-CD* → CL.

The translations are specified in [56],[53]. Properties of translations have been introduced in section 10.2. All translations are marked as default translations.

I.3 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 $TPTP$

— $UML\text{-}CD$, supported by $UML\text{-}CD$.

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** $\to$ CL

— **FOL** $\to SubPCFOL^{=}_{ms}$

— $UML\text{-}CD \to$ 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.4 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/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.5 Language and Logic Translations

I.5.1 EL → OWL and $\mathcal{EL}{++} \rightarrow \mathcal{SROIQ}(D)$

EL → OWL is the sublanguage inclusion obtained by the syntactic restriction according to the definition of EL, see **NR6**. Since by definition, $\mathcal{EL}{++}$ is a syntactic restriction of $\mathcal{SROIQ}(D)$, $\mathcal{EL}{++} \rightarrow \mathcal{SROIQ}(D)$ is the corresponding sublogic inclusion.

I.5.2 QL → OWL and DL-Lite$_R \rightarrow \mathcal{SROIQ}(D)$

QL → OWL is the sublanguage inclusion obtained by the syntactic restriction according to the definition of QL, see **NR6**. Since by definition, DL-Lite$_R$ is a syntactic restriction of $\mathcal{SROIQ}(D)$, DL-Lite$_R \rightarrow \mathcal{SROIQ}(D)$ is the corresponding sublogic inclusion.

I.5.3 RL → OWL and RL $\rightarrow \mathcal{SROIQ}(D)$

RL → OWL is the sublanguage inclusion obtained by the syntactic restriction according to the definition of RL, see **NR6**. Since by definition, RL is a syntactic restriction of $\mathcal{SROIQ}(D)$, RL $\rightarrow \mathcal{SROIQ}(D)$ is the corresponding sublogic inclusion.

I.5.4 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 translation of realizations needs to forget the fixed parts of RDF realizations. Since this part can always reconstructed in a unique way, the result is an isomorphic translation of realizations.

I.5.5 RDF $\rightarrow$ RDFS

This is entirely analogous to SimpleRDF $\rightarrow$ RDF.

I.5.6 SimpleRDF $\rightarrow$ $\mathcal{SROIQ}(D)$

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 $\mathcal{SROIQ}(D)$ realization $\mathcal{I}$, obtain a SimpleRDF realization 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.,

$$EXT_m(x) := \{(y, z) \mid \exists u.(u, x) \in pred^{\mathcal{I}}, (u, y) \in sub^{\mathcal{I}}, (u, z,) \in obj^{\mathcal{I}}\}.$$

RDF $\rightarrow$ $\mathcal{SROIQ}(D)$ 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 translation of realizations can add the built-ins.

I.5.7 OWL $\rightarrow$ FOL

I.5.7.1 Translation of signatures

$\Phi((\mathbf{C}, \mathbf{R}, \mathbf{I})) = (F, P)$ with

— function symbols: $F = \{a^{(1)} | a \in \mathbf{I}\}$

— predicate symbols $P = \{A^{(1)} | A \in \mathbf{C}\} \cup \{R^{(2)} | R \in \mathbf{R}\}$

I.5.7.2 Translation of sentences

Concepts are translated as follows:

— $\alpha_x(A) = A(x)$

— $\alpha_x(\top) = true$

— $\alpha_x(\bot) = false$

— $\alpha_x(\neg C) = \neg \alpha_x(C)$

— $\alpha_x(C \sqcap D) = \alpha_x(C) \wedge \alpha_x(D)$

— $\alpha_x(C \sqcup D) = \alpha_x(C) \vee \alpha_x(D)$

— $\alpha_x(\exists R.C) = \exists y.(R(x, y) \wedge \alpha_y(C))$

— $\alpha_x(\exists U.C) = \exists y.\alpha_y(C)$

— $\alpha_x(\forall R.C) = \forall y.(R(x,y) \rightarrow \alpha_y(C))$

— $\alpha_x(\forall U.C) = \forall y.\alpha_y(C)$

— $\alpha_x(\exists R.\mathrm{Self}) = R(x,x)$

— $\alpha_x(\leq nR.C) = \forall y_1,\ldots,y_{n+1}.\bigwedge_{i=1,\ldots,n+1}(R(x,y_i) \wedge \alpha_{y_i}(C)) \rightarrow \bigvee_{1\leq i<j\leq n+1} y_i = y_j$

— $\alpha_x(\geq nR.C) = \exists y_1,\ldots,y_n.\bigwedge_{i=1,\ldots,n}(R(x,y_i) \wedge \alpha_{y_i}(C)) \wedge \bigwedge_{1\leq i<j\leq n} y_i \neq y_j$

— $\alpha_x(\{a_1,\ldots a_n\}) = (x = a_1 \vee \ldots \vee x = a_n)$

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:

— $\alpha_\Sigma(C \sqsubseteq D) = \forall x.\,(\alpha_x(C) \rightarrow \alpha_x(D))$

— $\alpha_\Sigma(a : C) = \alpha_x(C)[x \mapsto a]$[49)]

— $\alpha_\Sigma(R(a,b)) = R(a,b)$

— $\alpha_\Sigma(R \sqsubseteq S) = \forall x,y.R(x,y) \rightarrow S(x,y)$

— $\alpha_\Sigma(R_1;\ldots;R_n \sqsubseteq R) =$
$\forall x,y.(\exists z_1,\ldots,z_{n-1}.R_1(x,z_1) \wedge R_2(z_1,z_2) \wedge \ldots \wedge R_n(z_{n-1},y)) \rightarrow R(x,y)$

— $\alpha_\Sigma(\mathrm{Dis}(R_1,R_2)) = \neg\exists x,y.R_1(x,y) \wedge R_2(x,y)$

— $\alpha_\Sigma(\mathrm{Ref}(R)) = \forall x.R(x,x)$

— $\alpha_\Sigma(\mathrm{Irr}(R)) = \forall x.\neg R(x,x)$

— $\alpha_\Sigma(\mathrm{Asy}(R)) = \forall x,y.R(x,y) \rightarrow \neg R(y,x)$

— $\alpha_\Sigma(\mathrm{Tra}(R)) = \forall x,y,z.R(x,y) \wedge R(y,z) \rightarrow R(x,z)$

I.5.7.3 Translation of realizations

— For $M' \in \mathrm{Real}^{FOL}(\Phi\Sigma)$ define $\mathcal{I} = \beta_\Sigma(M') := (\Delta, \cdot^{\mathcal{I}})$ with $\Delta = |M'|$ and $A^{\mathcal{I}} = M'_A, a^{\mathcal{I}} = M'_a, R^{\mathcal{I}} = M'_R$.

Proposition 24 $C^{\mathcal{I}} = \{m \in \Delta | M' + \{x \mapsto m\} \models \alpha_x(C)\}$

Proof. By induction over the structure of C.

— $A^{\mathcal{I}} = M'_A = \{m \in \Delta | M' + \{x \mapsto m\} \models A(x)\}$

— $(\neg C)^{\mathcal{I}} = \Delta \setminus C^{\mathcal{I}} =^{I.H.} \Delta \setminus \{m \in \Delta | M' + \{x \mapsto m\} \models \alpha_x(C)\} = \{m \in \Delta | M' + \{x \mapsto m\} \models \neg\alpha_x(C)\}$

49) $t[x \mapsto a]$ means "in t, replace x by a".

□

The other cases are similar.

The satisfaction condition now follows easily.

I.5.8 $FOL \to$ 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 realization is translated to a FOL realization 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.5.9 OWL $\to$ CL

This comorphism is the composition of the comorphisms described in the previous two sections.

I.5.10 UML class models $\to$ CL

This translation has been described in annex F. Translation of signatures is detailed in section F.4.3, translation of sentences in section F.4.5. Realizations are translated identically.

I.5.11 $FOL \to$ CASL

This is an obvious sublogic.

I.5.12 UML class model to OWL

Let $\Sigma = ((C, \leq_C), P, O, A, M)$ be a *class/data type net* representing a UML class model as described in annex F. This net can be translated to OWL2 using the approach described in [76]. The ontology is extended by translating parts of this net and its multiplicity constraints $Mult(\Sigma)$:

— For each class $c \in C$ with superclasses $c_1, c_2, \ldots, c_n \in C$ (i.e., $c \leq_C c_i$ for $i = 1, \ldots, n$):

```
Class: c
        SubClassOf: c1
        ...
        SubClassOf: cn
```

— For each attribute declaration $c.p : c'$ in P

```
ObjectProperty: p
        Domain: c
        Range: c′
```

— For each attribute multiplicity $n \leq c.p : \tau[c']$ in $Mult(\Sigma)$ extend the description of class c by:

```
SubClassOf: p min n c′
```

— For each attribute multiplicity $c.p : \tau[c'] \leq n$ in $Mult(\Sigma)$ extend the description of class c by:

```
SubClassOf: p max n c′
```

— For each unidirectional binary association declaration $a(p_1 : \tau_1[c_1], p_2 : \tau_2[c_2])$ in A:

```
ObjectProperty: p
        Domain: c1
        Range: c2
```

— For each bidirectional binary association declaration $a(p_1 : \tau_1[c_1], p_2 : \tau_2[c_2])$ in A:

```
ObjectProperty: p1
        Domain: c
        Range: c′

ObjectProperty: p2
        Characteristics: InverseFunctional
        Domain: c
        Range: c′
        InverseOf: p1
```

— For each binary association $n \leq a(p_1 : \tau_1[c_1], p_2 : \tau_2[c_2]).p_i$, with $i \neq j \in \{1, 2\}$ in $Mult(\Sigma)$ extend the description of class c_j by:

```
SubClassOf: pi min n ci
```

— For each binary association $a(p_1 : \tau_1[c_1], p_2 : \tau_2[c_2]).p_i \leq n$, with $i \neq j \in \{1, 2\}$ in $Mult(\Sigma)$ extend the description of class c_j by:

```
SubClassOf: pi max n ci
```

— For each composition declaration $m(\mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2])$ in M:

```
ObjectProperty: p
        Characteristics:
                Functional,
                Irreflexive
        Domain: c1
        Range: c2
```

— For each binary association $n \leq a(p_1 : \tau_1[c_1], \blacklozenge p_2 : \tau_2[c_2]).p_i$, with $i \neq j \in \{1, 2\}$ in $Mult(\Sigma)$ extend the description of class c_j by:

```
SubClassOf: pi min n ci
```

— For each binary association $a(p_1 : \tau_1[c_1], \blacklozenge p_2 : \tau_2[c_2]).p_i \leq n$, with $i \neq j \in \{1, 2\}$ in $Mult(\Sigma)$ extend the description of class c_j by:

```
SubClassOf: pi max n ci
```

I.6 Formal Representation of Language and Logic Translations

A formal representation of language and logic translations still needs to be developed. For the syntax aspects of these translations, QVT could be a useful option. However, it would have added value to choose a representation of translations that allows their correctness to be proven easily. Such a representation would have to interact with suitable representations of languages and logics in a logical framework. See [7] for some work in this direction.

Annex J
(informative)
Extended Logic Graph

This annex extends the graph of logics and translations given in annex I by a list of OMS languages whose inclusion in the registry is planned. 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)

— OBO^{OWL} and OBO1.4

— RIF **NR30** (Rule Interchange Format)

— EER (Enhanced Entity-Relationship Models)

— Datalog

— ORM (object role modeling)

— the meta model of schema.org

— different model types of the UML (Unified Modeling Language), with possibly different logics according to different UML semantics

— SKOS (Simple Knowledge Organization System; **NR22**)

— $\text{FOL}^{=}$ (untyped first-order logic, as used for the TPTP format)

— F-logic

The actual translations are specified in [56].

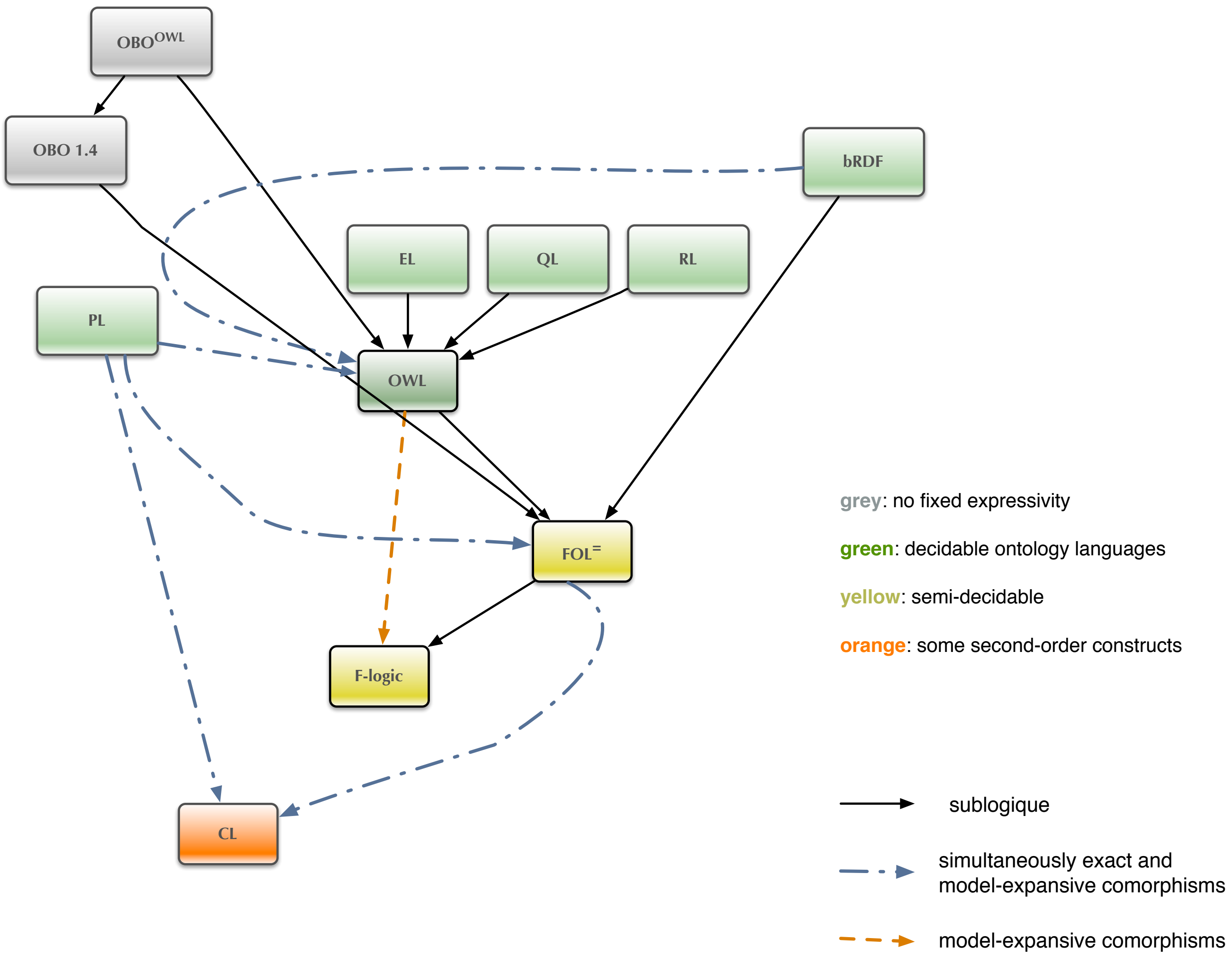


Figure J.1: Translations between conforming OMS languages (extended)

Annex K
(informative)
DOL Abstract Syntax in EBNF

K.1 General

The following subclauses specify the abstract syntax of DOL in EBNF. Note that it deviates from the EBNF specification in **NR3** in favor of a more concise EBNF syntax. More precisely, **NR3** requires commas between the (non-)terminals of a right-hand side, which are omitted for the sake of better readability. Also, the separator = between left and right hand-side of a rule is replaced with `::=`, and the notation `N+` is used for one or more repetitions of `N`.

Note that the EBNF abstract syntax is constructor-based. E.g., `translation` is a constructor that can be used to build more complex OMS from smaller OMS (and in general, the constructors can be used to form syntax trees). By contrast, the MOF-based abstract syntax in clause 9 is selector-based, i.e., it features selectors that, given a complex OMS, extract the simpler OMS that are its building blocks. While the metaclasses of the MOF metamodel largely match the non-terminal symbols of the EBNF abstract syntax, there is no such direct match between selectors and constructors.

The non-terminals of the EBNF abstract syntax largely match those of the concrete syntax given in clause 9. Some non-terminals of the concrete syntax have been omitted in the abstract syntax, because they serve merely syntactical purposes and do not contribute to a useful syntax tree. Otherwise, the productions of the concrete syntax match those of the abstract syntax, but the abstract syntax constructors are replaced with specific strings (possibly interspersed at different positions) expressing the concrete syntax.

K.2 Documents

```
Document           ::= DOLLibrary | NativeDocument
DOLLibrary         ::= library [PrefixMap] LibraryName Qualification
                               LibraryItem*
NativeDocument     ::= <language specific>
LibraryItem        ::= LibraryImport | Definition | Qualification
Definition         ::= OMSDefinition
                     | NetworkDefinition
                     | MappingDefinition
                     | QueryRelatedDefinition
LibraryImport      ::= lib-import LibraryName
Qualification      ::= LanguageQualification
                     | LogicQualification
                     | SyntaxQualification
LanguageQualification ::= lang-select LanguageRef
LogicQualification ::= logic-select LogicRef
SyntaxQualification ::= syntax-select SyntaxRef
LanguageRef        ::= IRI
LogicRef           ::= IRI
SyntaxRef          ::= IRI
LibraryName        ::= IRI
PrefixMap          ::= prefix-map PrefixBinding*
Prefix             ::= String
Separators         ::= separators LibraryOMSSeparator OMSSymbolSeparator
LibraryOMSSeparator ::= String
OMSSymbolSeparator ::= String
```

K.3 OMS Networks

```
NetworkDefinition ::= network-definition NetworkName
                                         [ConservativityStrength] Network
NetworkName         ::= IRI
Network             ::= network NetworkElement* ExcludedElement*
NetworkElement      ::= network-element [Id] IRI
ExcludedElement     ::= PathReference | ExcludedElementRef
PathReference       ::= path IRI IRI
ExcludedElementRef  ::= IRI
```

K.4 OMS

```
BasicOMS           ::= < language specific >
OMS                ::= ExtendingOMS
                     | ClosureOMS
                     | TranslationOMS
                     | ReductionOMS
                     | ExtractionOMS
                     | ApproximationOMS
                     | FilteringOMS
                     | UnionOMS
                     | ExtensionOMS
                     | QualifiedOMS
                     | CombinationOMS
                     | ApplicationOMS
ClosureOMS         ::= closure-symbols OMS Closure
TranslationOMS     ::= translation OMS OMSTranslation
ReductionOMS       ::= reduction OMS Reduction
ExtractionOMS      ::= module-extract OMS Extraction
ApproximationOMS   ::= approximation OMS Approximation
FilteringOMS       ::= filtering OMS Filtering
UnionOMS           ::= union OMS [ConservativityStrength] OMS
ExtensionOMS       ::= extension OMS Extension
QualifiedOMS       ::= qualified-oms Qualification Qualification* OMS
CombinationOMS     ::= combination Network
ApplicationOMS     ::= application OMS SubstName
OMSDefinition      ::= oms-definition OMSName [ConservativityStrength] OMS
ConservativityStrength ::= consequence-conservative
                     | model-conservative
                     | not-consequence-conservative
                     | not-model-conservative
                     | implied
                     | monomorphic
                     | weak-definitional
                     | definitional
OMSName            ::= IRI
SubstName          ::= IRI


OMSReference       ::= oms-reference OMSRef [ImportName]
Extension          ::= extension [ConservativityStrength]
                                 [ExtensionName] ExtendingOMS
ExtendingOMS       ::= ClosableOMS | RelativeClosureOMS
RelativeClosureOMS ::= relative-closure ClosureType ClosableOMS
```

```
Closure            ::= ClosureType CircClosure CircVars
ClosureType        ::= minimize | maximize | free | cofree
CircClosure        ::= Symbol Symbol*
CircVars           ::= Symbol*
ExtensionName      ::= IRI
ImportName         ::= IRI
OMSRef             ::= IRI
```

```
Reduction          ::= reduction RemovalKind OMSLanguageTranslation*
                                 [SymbolList]
SymbolList         ::= Symbol Symbol*
SymbolMap          ::= symbol-map GeneralSymbolMapItem
                                  GeneralSymbolMapItem*
Extraction         ::= extraction RemovalKind InterfaceSignature
Approximation      ::= approx RemovalKind [InterfaceSignature] [LogicRef]
Filtering          ::= filter RemovalKind BasicOMSOrSymbolList
BasicOMSOrSymbolList ::= BasicOMS | SymbolList
InterfaceSignature ::= SymbolList
SymbolMapItem      ::= symbol-map-item Symbol Symbol
GeneralSymbolMapItem ::= Symbol | SymbolMapItem
OMSLanguageTranslation ::= NamedTranslation | DefaultTranslation
NamedTranslation   ::= named-trans OMSLanguageTranslationRef
DefaultTranslation ::= default-trans LanguageRef
RemovalKind        ::= keep | remove
OMSLanguageTranslationRef ::= IRI
Symbol             ::= IRI
```

K.5 OMS Mappings

```
                                                        InterpretationName
                                                        [ConservativityStrength]
                                                        InterpretationType
                                                        OMSLanguageTranslation*
                                                        [SymbolMap]
RefinementDefinition ::= refinement InterpretationName Refinement
InterpretationName ::= IRI
InterpretationType ::= interpretation-type OMS OMS
Refinement         ::= RefinementOMS
                     | RefinementNetwork
                     | SimpleOMSRefinement
                     | SimpleNetworkRefinement
RefinementOMS      ::= refinement-oms OMS
RefinementNetwork  ::= refinement-network Network
SimpleOMSRefinement ::= simple-oms-ref Refinement OMSRefinementMap Refinement
SimpleNetworkRefinement ::= simple-network-ref Refinement
                                                NetworkRefinementMap Refinement
OMSRefinementMap   ::= oms-refmap [OMSLanguageTranslation] [SymbolMap]
NetworkRefinementMap ::= network-refmap Refinement*
```

```
                     | RefinementDefinition
                     | EntailmentDefinition
                     | EquivalenceDefinition
                     | ConservativeExtensionDefinition
```

```
                    | AlignmentDefinition
EntailmentDefinition ::= entailment EntailmentName EntailmentType
OMSOMSEntailment   ::= oms-oms-entailment OMS OMS
NetworkOMSEntailment ::= network-oms-entailment Network OMSName OMS
NetworkNetworkEntailment ::= network-network-entailment Network Network
EntailmentType     ::= OMSOMSEntailment
                    | NetworkOMSEntailment
                    | NetworkNetworkEntailment
EntailmentName     ::= IRI
EquivalenceDefinition ::= equivalence-definition
                                                 EquivalenceName
                                                 EquivalenceType
EquivalenceName    ::= IRI
EquivalenceType    ::= OMSEquivalence | NetworkEquivalence
OMSEquivalence     ::= oms-equivalence OMS OMS [OMS]
NetworkEquivalence ::= network-equivalence Network Network [Network]
```

```
                           [AlignmentCardinality] [AlignmentCardinality]
                           AlignmentType Correspondence*
                           [AlignmentSemantics]<\footnote{%
                     Note that this grammar uses ``type'' as in
                     ``the type of a function'', whereas the Alignment API
                     \cite{AlignmentAPI} uses ``type''  for\:  the
                     totality/injectivity of the relation/function.  For the
                     latter, this grammar uses ``cardinality''.}>
AlignmentName      ::= IRI
AlignmentCardinality ::= injective-and-total
                    | injective
                    | total
                    | neither-injective-nor-total
AlignmentType      ::= alignment-type OMS OMS
AlignmentSemantics ::= single-domain
                    | global-domain
                    | contextualized-domain
Correspondence     ::= CorrespondenceBlock
                    | SingleCorrespondence
                    | DefaultCorrespondence
DefaultCorrespondence ::= default-correspondence
CorrespondenceBlock ::= correspondence-block [Relation]
                                             [Confidence] Correspondence
                                             Correspondence*
SingleCorrespondence ::= correspondence Symbol [Relation]
                                        [Confidence] GeneralizedTerm
                                        [CorrespondenceID]
CorrespondenceID   ::= IRI
GeneralizedTerm    ::= Symbol
Symbol             ::= IRI
Relation           ::= RelationReference | StandardRelation
StandardRelation   ::= StandardRelationValues
StandardRelationValues ::= subsumes
                    | is-subsumed
                    | equivalent
                    | incompatible
                    | has-instance
                    | instance-of
                    | default-relation
RelationReference  ::= relation-ref IRI
Confidence         ::= Double
Double             ::= < a number ∈ [0,1] >
```

```
                                           [ConservativityStrength] ConservativeExtensionType
                                           InterfaceSignature
ConservativeExtensionName         ::= IRI
ConservativeExtensionType         ::= cons-ext-type OMS OMS
```

K.6 IRIs and Prefixes

```
IRI      ::= FullIRI | CurieIRI50)
CurieIRI ::= curie CURIE
FullIRI ::= < as defined by the IRI production in NR11 >
CURIE  ::= String
```

[50] Specified below in clause 9.7.2.

This page intentionally left blank.

Annex L
(informative)
Extension of DOL with Queries

L.1 General

This annex describes the syntax of queries. A semantics still needs to be developed. DOL's metaclass `LibraryItem` is extended with a new subclass `QueryRelatedDefinition` for definitions related to queries.

L.2 Terms and Definitions

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.

L.3 MOF Abstract Syntax

Queries are a means to extract information from an OMS. DOL's `QueryDefinitions` cover "select"-type queries that deliver an answer substitution for the query variables. (Answer) substitutions can be stored separately, using a `SubstitutionDefinition`. A `ResultDefinition` 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. Note that by default, OMS are employed with an open world semantics, but using minimizations, (part of) OMS can be equipped with a closed world semantics. The corresponding extension of the DOL metamodel is shown in Figure L.1.

L.4 EBNF Concrete Syntax

```
Term                   ::= < an expression specific to an OMS language >
GeneralizedTerm        ::= Term | Symbol
QueryRelatedDefinition ::= QueryDefinition
                         | SubstitutionDefinition
```

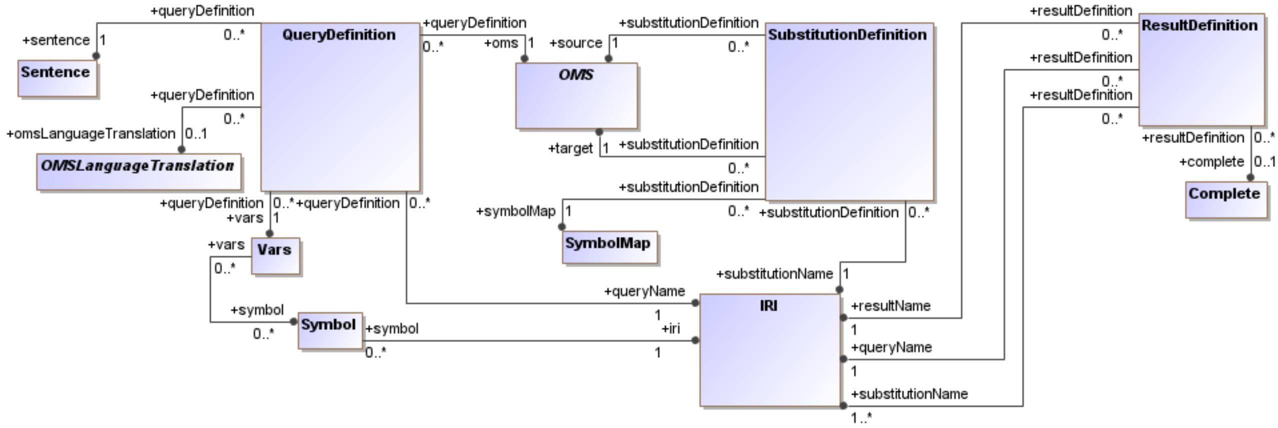


Figure L.1: Extension of DOL metamodel with queries

```
                          | ResultDefinition
QueryDefinition     ::= 'query' QueryName '=' 'select' Vars 'where'
                          Sentence 'in' GroupOMS
                          ['along' OMSLanguageTranslation] 'end'
SubstitutionDefinition ::= 'substitution' SubstitutionName ':'
                              GroupOMS 'to' GroupOMS '=' SymbolMap
                              'end'
ResultDefinition    ::= 'result' ResultName '=' SubstitutionName
                          ( ',' SubstitutionName )* 'for' QueryName
                          ['%complete'] 'end'
OMS                 ::= ... | OMS 'with' SubstitutionName
QueryName           ::= IRI
SubstitutionName    ::= IRI
ResultName          ::= IRI
Vars                ::= Symbol ( ',' Symbol )*
```

L.5 EBNF Abstract Syntax

```
QueryRelatedDefinition ::= QueryDefinition
                          | SubstitutionDefinition
                          | ResultDefinition
QueryDefinition     ::= select-query-definition
                                                QueryName Vars Sentence OMS
                                                [OMSLanguageTranslation]
SubstitutionDefinition ::= substitution-definition
                                                    SubstitutionName OMS OMS
                                                    SymbolMap
ResultDefinition    ::= result-definition ResultName
                                          SubstitutionName SubstitutionName*
                                          QueryName [Complete]
Sentence            ::= < an expression specific to an OMS language >
OMS                 ::= ... | application  OMS  SubstitutionName
QueryName           ::= IRI
SubstitutionName    ::= IRI
ResultName          ::= IRI
Vars                ::= Symbol*
Complete            ::= complete
```

L.6 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 [55], 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.

This page intentionally left blank.

Annex M
(informative)
Example Uses of all DOL Constructs

M.1 General

This annex provides example uses of DOL constructs. Jointly with clause 7, which contains DOL examples for the usage scenarios, all DOL constructs (although not necessarily all variants of each construct) are covered. The examples follow the DOL Text Serialization (clause 9). The following table provides an overview of which DOL language constructs have been covered where.

Top-level declarations in DOL 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 7.10
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 7.10
refinement IRI = OMS refined via translation IRI to OMS	section 7.12
refinement IRI = IRI refined to IRI	section 7.10
refinement IRI = Network refined to Network	section 7.11
entailment IRI = OMS entails OMS	PropositionalExamples
entailment IRI = OMSName in Network entails OMS	section 7.11
entailment IRI = Network entails Network	section 7.11
equivalence IRI : OMS <-> OMS = OMS end	Algebra
cons-ext IRI : OMS of OMS for Symbols	section 7.4
alignment IRI : OMS to OMS = Correspondences	Alignments
alignment IRI : OMS to OMS = Correspondences assuming SingleDomain	[9]
alignment IRI : OMS to OMS = Correspondences assuming GlobalDomain	[9]
alignment IRI : OMS to OMS = Correspondences assuming ContextualizedDomain	[9]
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 7.10
OMS with translation IRI	Mereology
OMS hide SymbolList	Algebra
OMS reveal Symbols	Datatypes
OMS hide along IRI	section 7.11, MetricSpaces
OMS extract Symbols	section 7.4
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	[45]
OMS then %mcons OMS	Propositional
OMS then %notccons OMS	[45]
OMS then %notmcons OMS	[45]
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

M.2 Simple Examples in Propositional Logic

```
%prefix( :      <http://www.example.org/prop#>
         log:   <http://purl.net/DOL/logics/>
                %% descriptions of logics ...
         ser:   <http://purl.net/DOL/serializations/> )%
                %% ... and serializations

library PropositionalExamples

%% non-standard serialization built into Hets:
logic log:Propositional serialization ser:Propositional/Hets

oms Consistent = %consistent
  props A, B
  . A => B
end

oms Inconsistent = %inconsistent
```

```
  props A
  . A /\ not A
end

oms SingleRealization = %def
  props A, B
  . A /\ not B
end

entailment Ent = SingleRealization entails { . not ( A=>B ) }
end

%% repeat prefix declarations from above

library PropositionalMereology

%% non-standard serialization built into Hets:
logic log:Propositional serialzation ser:Propositional/Hets

%% basic taxonomic information about mereology reused from DOLCE:
ontology Taxonomy = %consistent
  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
```

M.3 Engine Diagnosis and Repair

```
%prefix( log: <http://purl.net/DOL/logics/> )%

library Engine

logic log: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
```

M.4 Mereology: Distributed and Heterogeneous Ontologies

```
%prefix( :      <http://www.example.org/mereology#>
         owl:   <http://www.w3.org/2002/07/owl#>
         lang:  <http://purl.net/DOL/languages/>
                %% definitions of conforming languages ...
         ser:   <http://purl.net/DOL/serializations/>
                %% ... and their serializations
         log:   <http://purl.net/DOL/logics/>
                %% descriptions of logics ...
         trans: <http://purl.net/DOL/translations/> )%
                %% ... and translations

library Mereology

import PropositionalMereology

%% OWL Manchester syntax declaration:
language lang:OWL2 logic log:SROIQ serialization 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 ]%
end

logic log:CommonLogic serialization 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)))))))
```

M.5 Defined Concepts

```
%prefix( lang:  <http://purl.net/DOL/languages/> )%

library Persons
language lang:OWL

ontology Persons =
  Class: Person
  Class: Female
then %def
  Class: Woman  EquivalentTo: Person and Female
end
```

M.6 Blocks World: Minimization

```
%prefix( lang:  <http://purl.net/DOL/languages/>)%

library BlocksWithCircumscription
language lang: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 vars 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 vars Ontable BlockNotAbnormal
then %implied
  Individual: B2 Types: Ontable
     %% B2 is on the table
end
```

M.7 Alignments

```
%prefix( :      <http://www.example.org/alignment#>
         owl:   <http://www.w3.org/2002/07/owl#>
         lang:  <http://purl.net/DOL/languages/>
                %% definitions of conforming languages ...
```

```
         ser:   <http://purl.net/DOL/serializations/>
                %% ... and their serializations
         log:   <http://purl.net/DOL/logics/>
                %% descriptions of logics ...
         trans: <http://purl.net/DOL/translations/> )%
                %% ... and translations

library Alignments

language lang:OWL2 logic log:SROIQ serialization 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
```

M.8 Distributed Description Logics

```
%prefix( :      <http://www.example.org/mereology#>
         owl:   <http://www.w3.org/2002/07/owl#>
         lang:  <http://purl.net/DOL/languages/>
                %% definitions of conforming languages ...
         ser:   <http://purl.net/DOL/serializations/>
                %% ... and their serializations
         log:   <http://purl.net/DOL/logics/>
                %% descriptions of logics ...
         trans: <http://purl.net/DOL/translations/> )%
                %% ... and translations

library Publications

language lang:OWL2 logic log:SROIQ serialization 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

%% repeat prefix declarations from above

library Market

language lang:OWL2 logic log:SROIQ serialization ser:OWL2/Manchester

ontology One = Class: PurchaseOrder end
ontology Two =
  ObjectProperty: Buyer
  ObjectProperty: Good
  ObjectProperty: BoughtBy
end


ontology Purchases =
combine
  1:One,
  2:Two
  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
```

M.9 Algebra

```
%prefix( :      <http://www.example.org/algebra#>
         lang:  <http://purl.net/DOL/languages/>
                %% descriptions of languages ...
         ser:   <http://purl.net/DOL/serializations/>
                %% ... serializations ...
         trans: <http://purl.net/DOL/translations/> )%
                %% ... and translations

library Algebra

language lang:CommonLogic serialization 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
```

```
language lang:CASL

equivalence e : algebra:BooleanAlgebra <-> algebra:BooleanRing =
  sort E
  forall x,y:E
  . x ∧ y = x · y
  . x ∨ y = x + y + x · y
  . ¬ x = 1 + x
  . x · y = x ∧ y
  . x + y = (x ∨ y) ∧ ¬(x ∧ y)
end
```

```
language lang: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
```

M.9.1 Groups specified with different forms of hiding and forgetting

M.9.1.1 Groups and hiding

```
%prefix( lang: <http://purl.net/DOL/languages/> )%
```

```
library All_kinds_of_group_specifications

language lang: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:

```
language lang: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
```

M.9.1.2 Groups and module extraction

```
language lang: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:

```
language lang: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.

M.9.1.3 Groups via interpolation

```
language lang: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`.

M.9.1.4 Groups and filtering

```
language lang: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):

```
language lang: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
```

M.10 Real Numbers and Metric Spaces

```
%prefix( lang:  <http://purl.net/DOL/languages/> )%

library MetricSpaces

language lang:CASL
```

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

spec CommutativeMonoid =
     Monoid
then
     op __ * __: Elem * Elem -> Elem, comm
end

spec Group =
     Monoid
then
     forall x: Elem
     . exists x': Elem . x' * x = e   %(inv_Group)%
end

spec AbelianGroup =
     Group
and
     CommutativeMonoid
end

spec Ring =
     AbelianGroup with sort Elem,
                       ops __ * __ |-> __ + __,
                           e       |-> 0
and
     Monoid with ops e, __*__
then
     forall x,y,z:Elem
     . (x + y) * z = (x * z) + (y * z)          %(distr1_Ring)%
     . z * ( x + y ) = (z * x) + (z * y)        %(distr2_Ring)%
end

view AbelianGroup_in_Ring_add:
     AbelianGroup to Ring =
     ops e |-> 0,
         __ * __ |-> __ + __
end

spec CommutativeRing =
     Ring with ops 0,  __ + __, e, __ * __
and
     CommutativeMonoid with ops e, __ * __
end

spec ConstructField =
{     CommutativeRing
then
     . not e = 0 %(zeroNeqOne_Field)%
     sort NonZeroElem = { x: Elem . not x = 0 } %(NonZeroElem_def)%
}
```

```
and
     {Group with sort Elem |-> NonZeroElem, ops e, __*__}
end

spec BasicField =
     ConstructField hide sort NonZeroElem
end

spec Field =
     BasicField with op e |-> 1
then %def
     op -__: Elem -> Elem
     forall x: Elem
     . -x + x = 0       %(Field_unary_minus_idef)%
end

spec FieldWithOrdering =
     Field and TotalOrder
then
    vars a, b, c:Elem
    . (a + c) <= (b + c) if a <= b              %(FWO_plus_left)%
    . (a * c) <= (b * c) if a <= b /\ 0 <= c;   %(FWO_times_left)%
then %implied
    vars a, b, c, d: Elem
    . (a + b) <= (a + c) if b <= c              %(FWO_plus_right)%
    . (a * b) <= (a * c) if b <= c /\ 0 <= a    %(FWO_times_right)%
    . (a + b) <= (c + d) if a <= c /\ b <= d    %(FWO_plus)%
end


spec OrderedField =
     Field
then
     pred Pos: Elem
     forall x,y: Elem
     . Pos(x) /\ Pos(y) => Pos(x*y)             %(OF_plus)%
     . Pos(x) /\ Pos(y) => Pos(x+y)             %(OF_times)%
     . Pos(x) /\ Pos(-x) => x = 0               %(OF_mutex)%
     . Pos(x) \/ Pos(-x)                        %(OF_exhaust)%
end

spec RichOrderedField =
  OrderedField
then %def
  ops min, max: Elem * Elem -> Elem, comm,assoc %implied
  preds __ <= __, __ < __,
           __ >= __, __ > __: Elem * Elem;
  forall x,y:Elem
     . x >= y <=> y <= x                   %(geq_def_ExtPartialOrder)%
     . x < y <=> (x <= y /\ not (x=y))     %(less_def_ExtPartialOrder)%
     . x > y <=> y < x                     %(greater_def_ExtPartialOrder)%
  forall x,y: Elem
    . min(x,y) = x when x <= y else y      %(min_def_ExtTotalOrder)%
    . max(x,y) = y when x <= y else x      %(max_def_ExtTotalOrder)%
  forall x,y: Elem
       . x <= y  <=> Pos(y + -x)
```

```
end

%% real numbers, using a specification of fields
language lang:HasCASL
spec Real =
     RichOrderedField with Elem |-> Real
then
  free type Nat ::= 0 | suc Nat
  ops __<=__ :      Pred(Real * Pred(Real));
      __<=__ :      Pred(Pred(Real) * Real);
      isBounded : Pred(Pred(Real));
      inf,sup :   Pred(Real) ->? Real;
      inj:        Nat -> Real
   forall r,s:Real; M:Pred(Real); n: Nat
   . M <= r <=> forall s:Real . M(s) => s <= r          %(Real_ub_def)%
   . r <= M <=> forall s:Real . M(s) => r <= s          %(Real_lb_def)%
   . inf(M)=r <=> r <= M /\ forall s:Real . s <= M => s <= r  %(Real_inf_def)%
   . sup(M)=r <=> M <= r /\ forall s:Real . M <= s => r <= s  %(Real_sup_def)%
   . isBounded(M) <=> exists ub,lb:Real . lb <= M /\ M <= ub
                %(Real_isBounded_def)%  %% this is the single higher-order axiom
   . isBounded(M) => def inf(M) /\ def sup(M)  %(completeness)%
   . inj 0 = 0                                          %(Real_inj_0)%
   . inj (suc n) = 1 + inj n                    %(Real_inj_suc)%
   . exists n: Nat. r <= inj n                  %(Real_archimedian)%
end

%% metric spaces in a first-order setting
language lang:CASL
spec MetricSpace =
     Real hide along HasCASL2CASL
then
     sort S
     op d:S*S->Real
     var x,y,z:S
     . d(x,y) = 0 <=> x = y
     . d(x,y) = d(y,x)
     . d(x,z) <= d(x,y) + d(y,z)
end
```

M.11 Datatypes

```
%prefix( lang:  <http://purl.net/DOL/languages/> )%
library Datatypes
language lang:CASL

spec Bag =
  sort Elem
  then free {
     sort Bag
     ops mt:Bag;
         __union__:Bag*Bag->Bag, assoc, comm, unit mt
           }
end
```

```
spec Bag_variant =
  sort Elem
then minimize { %% select term generated realizations
     sort Bag
     ops mt:Bag;
         __union__:Bag*Bag->Bag, assoc, comm, unit mt
           }
then
  pred __elem__ : Elem * Bag
  forall x:Elem; b1,b2:Bag
  . not x elem mt
  . x elem (b1 union b2) <=> (x elem b1 \/ x elem b2)
  . b1=b2 <=> forall y:Elem . (y elem b1 <=> y elem b2) %(extensionality)%
     %% term generatedness and extensionality together
     %% select the standard bag realization
end

equivalence e : Bag <-> Bag_variant = {}
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)
  pred __<__ : Nat * Nat
  forall m,n:Nat
  . 0 < suc(n)
  . not n < 0
  . suc(m) < suc(n) <=> m < n
  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
```

M.12 Queries

```
%prefix( lang:  <http://purl.net/DOL/languages/> )%
library MyQuery
language lang: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
```

Annex N
(informative)
Tools for DOL

N.1 The Heterogeneous Tool Set (Hets)

The Heterogeneous Tool Set (Hets) is an implementation of DOL. Hets is 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 models 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 a 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.

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 another 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 DOL 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,

— 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, see also the end of this section)

— 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/Spechub provide APIs for the interchange with other tools[51]. Ontohub/Modelhub/Spechub 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 API4KP.

[51] See `https://github.com/spechub/Hets/wiki/RESTful-Interface` and `https://github.com/ontohub/ontohub/wiki/`.

Annex O
(informative)
Ontohub loc/id v2

O.1 General

This annex describes the way how Ontohub assigns IRIs to DOL libraries, OMS, symbols etc. Ontohub[52] is an implementation for DOL, and it is suggested that other tools supporting DOL should adopt the same or a similar scheme for IRIs.

O.2 Concept

Generally an Ontohub loc/id (locator/identifier) is just an IRI of a DOL library (contained in a document), 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. Ontohub tries to infer them from the path of the repository, the path of the OMS and the specific name. Additionally, Ontohub ensures 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's 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.3 Ontohub-Style

Identifying OMS and their members in Ontohub is a hierarchical task. A DOL document belongs to a repository. An OMS may belong directly to a repository, or indirectly through a DOL library. Mappings, symbols and sentences in turn belong to an OMS. So one 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 a *qualified loc/id* is expected 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.3.1 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.

[52] 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.3.2 Examples

DOL *document*

DOL document	`/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

DOL document	`/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.4 Specification

A qualified loc/id IRI can be specified as a special case of RFC 3987 (IRI, [15]). Code-excerpt O.1 on page 185 contains this specification of qualified loc/ids in Augmented Backus-Naur Form (ABNF, [12]). ABNF is used, because RFC 3987 itself specifies IRIs using ABNF and it is desirable 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 [3], 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.5 ref/ special form loc/ids

There is one additional syntax-element that has not been 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. For many use cases it is important to 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 [65], which should guarantee basic support for tools which expect the MMT-style.

O.5.1 References inside of the tree

It is important 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, Ontohub 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.6 Disambiguation

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 Ontohub serves the OMS. This is in part because there is no reasonable representation of a directory that could be supported. 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), one needs to serve that, even if the loc/id would be ambiguous in a normal request.

This page intentionally left blank.

Annex P
(informative)
Introduction to Category Theory

P.1 Categories

Definition 25 *A category C consists of*

— *a class of* objects*, denoted $|C|$,*

— *for each two objects a and b, a class of* morphisms *(or* arrows*), denoted $C(a, b)$,*

— *for each three objects a, b and c, a* composition *operation, denoted $; : C(a, b) \times C(b, c) \to C(a, c)$ such that the following axioms hold:*

  — *if $f \in C(a, b)$, $g \in C(b, c)$ and $h \in C(c, d)$ for four objects a, b, c, d, then $f; (g; h) = (f; g); h$*

  — *for each object a there is a morphism $id_a \in C(a, a)$ such that for every $f \in C(a, b)$ and every $g \in C(b, a)$ for some object b we have that $id_a; f = f$ and $g; id_a = g$.*

Example $\mathbb{S}et$ is the category whose class of objects is the class of all sets, $Set(A, B)$ is the set of all functions from A to B for any sets A and B, id_A is the identity function on a set A and the composition is the usual composition of functions.

Example Rel is the category whose class of objects is the class of all sets, $Rel(A, B)$ is the class of all relations $R \subseteq A \times B$, for any sets A and B, id_A is the diagonal relation $\{(a, a) \mid a \in A\}$ for a set A and the composition of $R \in Rel(A, B)$ with $S \in Rel(B, C)$ for three sets A, B, C is defined as $\{(a, c) \mid \text{exists } b \in B \text{ such that } (a, b) \in R \text{ and } (b, c) \in S\}$.

Example The category of unsorted first-order signatures has as objects tuples of the form $F = (F_i)_{i \in \mathbb{N}}$ where F_i is a set (of function symbols of arity i, for each natural number i). Given two objects F and G, a morphism $\sigma : F \to G$ is a family of functions $(\sigma_i : F_i \to G_i)_{i \in \mathbb{N}}$, which means that the arities of function symbols are preserved by morphisms. The identity morphism for an object F is the family of identity functions $(id_{F_i})_{i \in \mathbb{N}}$ and the composition is defined component-wise: if $\sigma : F \to G$ and $\tau : G \to H$ are signature morphisms between the signatures F, G and H, then $\sigma; \tau = (\sigma_i; \tau_i)_{i \in \mathbb{N}}$.

Example Given an unsorted first-order signature F, a realization M of F consists of an universe M_U together with an interpretation of each function symbol $f \in F_i$ as a function M_f taking i arguments in M_U with result in M_U. Given two such realizations M and N, a homomorphism of realizations $m : M \to N$ is a function $m : M_U \to N_U$ such that for each $i \in \mathbb{N}$ and each $f \in F_i$ we have that $m(M_f(x_1, \ldots, x_n)) = N_f(m(x_1), \ldots, m(x_n))$ for every $x_1, \ldots x_n$ in M_U. The identity function on M_U is a homomorphism of realizations on M and the composition is the usual composition of functions. This gives us the category of first-order realizations of F.

Definition 26 *Let C be a category. Its dual or opposite category, denoted C^{op}*

— *has the same objects as C: $|C^{op}| = |C|$,*

— *for two objects $a, b \in |C|$, $C^{op}(a, b) = C(b, a)$,*

— $;^{op} : C^{op}(a,b) \times C^{op}(b,c) \to C^{op}(a,c)$ *is defined as* $f;^{op} g = g;f$ *for any* $f \in C^{op}(a,b) = C(b,a)$ *and* $g \in C^{op}(b,c) = C(c,b)$. *The result* $g;f$ *is a morphism in* $C(c,a) = C^{op}(a,c)$,

— *for each object* a, $id_a \in C^{op}(a,a) = C(a,a)$ *is the identity w.r.t. the composition* $;^{op}$.

Definition 27 *An object* A *is called an initial object in a category* C *if for each object* B *of* C *there is exactly one morphism from* A *to* B.

Definition 28 *An object* A *is called a terminal object in a category* C *if for each object* B *of* C *there is exactly one morphism from* B *to* A.

EXAMPLE In $\mathbb{S}et$, the empty set is the initial object and each singleton set is a terminal object.

P.1.1 Limits and colimits

Definition 29 *A* network[53] *in a category* C *is a functor* $D : G \to C$, *where* G *is a small category*[54], *and can be thought of as the shape of the graph of interconnections between the objects of* C *selected by the functor* D.

Definition 30 *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,* $e \colon i \longrightarrow i'$ *it holds that* $D(e); \alpha_{i'} = \alpha_i$.

Definition 31 *A* colimiting cocone *(or colimit)* $(c, \{\alpha_i\}_{i \in |G|})$ *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, a *weak* colimit is obtained.

When G is the category $\bullet \longleftarrow \bullet \longrightarrow \bullet$, G-colimits are called *pushouts*. When G is a discrete category (i.e., no arrows between objects other than identities), G-limits are called *coproducts*.

Definition 32 *A* cone *of a network* $D : G \to C$ *consists of an object* c *of* C *and a family of morphisms* $\alpha_i \colon c \longrightarrow D(i)$, *for each object* i *of* G, *such that for each edge of the network,* $e \colon i \longrightarrow i'$ *it holds that* $\alpha_{i'} = \alpha_i; D(e)$.

Definition 33 *A* limiting cone *(or limit)* $(c, \{\alpha_i\}_{i \in |G|})$ *has the property that for any cone* $(d, \{\beta_i\}_{i \in |G|})$ *there exists a unique morphism* $\gamma \colon c \longrightarrow d$ *such that* $\gamma; \alpha_i = \beta_i$.

When G is the category $\bullet \longrightarrow \bullet \longleftarrow \bullet$, G-limits are called *pullbacks*. When G is a discrete category, G-limits are called *products*.

P.2 Functors

Definition 34 *Let* C *and* D *be two categories. A* functor $F : C \to D$ *is a mapping that*

— *assigns to each object* c *of* C *an object* $F(c)$ *in* D,

— *assigns to each morphism* $f \in C(c,d)$ *a morphism* $F(f) \in D(F(c), F(d))$ *such that*

[53] A network is called a diagram in category theory texts. This terminology is introduced to disambiguate OMS networks from UML diagrams.

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

— $F(id_c) = id_{F(c)}$ *for each* $c \in |C|$,

— $F(f;g) = F(f); F(g)$ *for each* $f \in C(a,b)$, $g \in C(b,c)$ *and* $a,b,c \in |C|$.

EXAMPLE For each category C, the identity functor $id_C : C \to C$ takes each object and each morphism to itself.

EXAMPLE The forgetful functor F from the category of unsorted first-order realizations of a signature F to Set takes each realization M to the set M_U and each morphism of realizations $m : M \to N$ to its underlying function $m : M_U \to N_U$.

EXAMPLE The covariant powerset functor $\mathcal{P} : \mathbb{S}et \to \mathbb{S}et$ maps each set A to the set of all subsets of A and each function $f : A \to B$ to the function that takes a subset X of A to the set $\{f(x) \mid x \in X\}$, which is a subset of B.

EXAMPLE The covariant finite powerset functor $\mathcal{P}_{\mathit{fin}} : \mathbb{S}et \to \mathbb{S}et$ maps each set A to the set of all finite subsets of A and each function $f : A \to B$ to the function that takes a subset X of A to the set $\{f(x) \mid x \in X\}$, which is a subset of B.

P.3 Natural transformations

Definition 35 *Let* C, D *be two categories and let* F *and* G *be two functors between* C *and* D. *A* natural transformation $\eta : F \to G$ *assigns to each object* $c \in |C|$ *a morphism* $\eta_c : F(c) \to G(c)$ *such that for every* $f \in C(c,d)$ *we have that* $F(f); \eta_d = \eta_c; G(c)$, *which means that the following diagram commutes*

$$\begin{tikzcd} F(c) \arrow[r, "F(f)"] \arrow[d, "\eta_c"'] & F(d) \arrow[d, "\eta_d"] \\ G(c) \arrow[r, "G(f)"'] & G(d) \end{tikzcd}$$

EXAMPLE There is an inclusion natural transformation $\iota : \mathcal{P}_{\mathit{fin}} \to \mathcal{P}$, i.e., for each set A, $\iota : \mathcal{P}_{\mathit{fin}}(A) \to \mathcal{P}(A)$ is the inclusion function (each finite subset of a set is also a subset of the set).

This page intentionally left blank.

Annex Q
(informative)
References

[1] J. Adámek, H. Herrlich, and G. Strecker. *Abstract and Concrete Categories.* Wiley, New York, 1990.

[2] K. J. Bagstad, F. Villa, G. 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.

[3] 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.

[4] A. Borgida and L. Serafini. Distributed description logics: Assimilating information from peer sources. *Journal of Data Semantics*, 1:153–184, 2003.

[5] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. Fabio Savo. The mastro system for ontology-based data access. *Semantic Web*, 2(1):43–53, 2011.

[6] 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.

[7] M. Codescu, F. Horozal, M. Kohlhase, T. Mossakowski, and F. Rabe. Project abstract: Logic atlas and integrator (latin). In J. H. Davenport, W. M. Farmer, J. Urban, and F. Rabe, editors, *Intelligent Computer Mathematics 18th Symposium, Calculemus 2011, and 10th International Conference, MKM 2011, Bertinoro, Italy, July 18-23, 2011. Proceedings*, volume 6824 of *Lecture Notes in Computer Science*, pages 289–291. Springer-Verlag Berlin Heidelberg, 2011.

[8] 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.

[9] M. Codescu, T. Mossakowski, and O. Kutz. A categorical approach to ontology alignment. In *Proc. of the 9th International Workshop on Ontology Matching (OM-2014), ISWC-2014, Riva del Garda, Trentino, Italy.*, CEUR-WS online proceedings, 2014.

[10] CoFI (The Common Framework Initiative). CASL *Reference Manual.* Lect. Notes Comp. Sci.2960 (IFIP Series). Springer, 2004.

[11] European Comission. INSPIRE Geoportal: Enhancing access to European spatial data, 2014. http://inspire-geoportal.ec.europa.eu/.

[12] D. Crocker and P. Overell. Augmented BNF for Syntax Specifications: ABNF. RFC 5234 (INTERNET STANDARD), January 2008. Updated by RFC 7405.

[13] J. David, J. Euzenat, F: Scharffe, and C. Trojahn dos Santos. The alignment API 4.0. *Semantic Web*, 2(1):3–10, 2011.

[14] R. Diaconescu, J. Goguen, and P. Stefaneas. Logical support for modularisation. In G. Huet and G. Plotkin, editors, *Proceedings of a Workshop on Logical Frameworks*, 1991.

[15] M. Duerst and M. Suignard. Internationalized Resource Identifiers (IRIs). RFC 3987 (Proposed Standard), January 2005.

[16] J. Euzenat and P. Shvaiko. *Ontology Matching, Second Edition.* Springer, 2013.

[17] 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.

[18] J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *J. ACM*, 39:95–146, 1992.

[19] 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.

[20] J. A. Goguen and G. Rosu. Composing hidden information modules over inclusive institutions. In O. Owe, S. Krogdahl, and T. 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.

[21] 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.

[22] Object Management Group. Ontology, model and specification integration and interoperability request for proposal. *OMG Document Number ad/13-12-02*, 2013.

[23] W3C OWL Working Group. Owl 2 web ontology language manchester syntax. *W3C Working Group Note, 27 October 2009.* http://www.w3.org/TR/owl2-manchester-syntax/, 2009.

[24] W3C OWL Working Group. Owl 2 web ontology language: Rdf-based semantics. *W3C Recommendation, 27 October 2009.* http://www.w3.org/TR/owl2-rdf-based-semantics/, 2009.

[25] W3C OWL Working Group. Owl 2 web ontology language: XML serialization (second edition). *W3C Recommendation, 11 December 2012.* http://www.w3.org/TR/owl2-xml-serialization/, 2012.

[26] W3C RDF Core Working Group. RDF 1.1 XML syntax. *W3C Recommendation, 25 February 2014.* http://www.w3.org/TR/rdf-syntax-grammar/, 2014.

[27] 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/swbp-vocab-pub/, 2008.

[28] I. Horrocks, O. Kutz, and U. Sattler. The Even More Irresistible $\mathcal{SROIQ}$. In P. Doherty, J. Mylopoulos, and C. Welty, editors, *Proc. of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR-06).* AAAI Press, 2006.

[29] Y. A. Ibanez, T. Mossakowski, A. Tarlecki, and D. Sannella. Modularity of ontologies in an arbitrary institution. In N. 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.

[30] ISO/IEC. 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.

[31] C. M. Keet, F.C. Fernández-Reyes, and A. Morales-González. Representing mereotopological relations in owl ontologies with ontoparts. In *Proceedings of the 9th Extended Semantic Web Conference (ESWC'12), 29-31 May 2012, Heraklion, Crete, Greece*, volume 7295 of *Lecture Notes in Computer Science*, pages 240–254. Springer, 2012.

[32] 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.

[33] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages,. *Journal of the ACM*, 42:741–843, 1995.

[34] A. Knapp, T. Mossakowski, and M. Roggenbach. An Institutional Framework for Heterogeneous Formal Development in UML, 2014. CoRR abs/1403.7747.

[35] A. Knapp and J. Wuttke. Model checking of UML 2.0 interactions. In T. Kühne, editor, *Reports Rev. Sel. Papers Wsh.s Symp.s MoDELS 2006*, number 4364 in Lect. Notes Comp. Sci., pages 42–51. Springer, 2007.

[36] R. Kontchakov, F. Wolter, and M. Zakharyaschev. Logic-based ontology comparison and module extraction, with an application to DL-Lite. *Artif. Intell.*, 174(15):1093–1141, 2010.

[37] O. Kutz, C. Lutz, F. Wolter, and M. Zakharyaschev. $\mathcal{E}$-connections of Abstract Description Systems. *Artificial Intelligence*, 156(1):1–73, 2004.

[38] O. Kutz, T. Mossakowski, J. Hois, M. Bhatt, and J. Bateman. Ontological Blending in DOL. In T. R. Besold, K-U. Kuehnberger, M. Schorlemmer, and A. Smaill, editors, *Computational Creativity, Concept Invention, and General Intelligence, Proc. of the 1st Int. Workshop C3GI@ECAI*, volume 01-2012, Montpellier, France, August 27 2012. Publications of the Institute of Cognitive Science, Osnabrück.

[39] O. Kutz, T. Mossakowski, and D. Lücke. Carnap, Goguen, and the Hyperontologies: Logical Pluralism and Heterogeneous Structuring in Ontology Design. *Logica Universalis*, 4(2):255–333, 2010. Special Issue on 'Is Logic Universal?'.

[40] O. Kutz, I. Normann, T. Mossakowski, and D. Walther. Chinese Whispers and Connected Alignments. In *Proc. of the 5th International Workshop on Ontology Matching (OM-2010)*, 9th International Semantic Web Conference ISWC-2010, November 7, 2010, Shanghai, China., 2010.

[41] C. Lange, T. Mossakowski, O. Kutz, C. Galinski, M. Grüninger, and D. Couto Vale. The Distributed Ontology Language (DOL): Use Cases, Syntax, and Extensibility. In *Proc. of the 10th Terminology and Knowledge Engineering Conference (TKE 2012)*, Madrid, Spain, 2012.

[42] Berners-T. Lee. Design issues: Linked data. *27 July 2006.* *http://www.w3.org/DesignIssues/LinkedData.html*, 2006.

[43] V. Lifschitz. Circumscription. In *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3*, pages 297–352. Oxford University Press, 1994.

[44] 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.

[45] C. Lutz, D. Walther, and F. Wolter. Conservative extensions in expressive description logics. In M. 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.

[46] C. Lutz and F. Wolter. Foundations for uniform interpolation and forgetting in expressive description logics. In Toby Walsh, editor, *IJCAI*, pages 989–995. IJCAI/AAAI, 2011.

[47] S. Mac Lane. *Categories for the Working Mathematician.* Springer, 1998. Second edition.

[48] C. Masolo, S. Borgo, A. Gangemi, N. Guarino, and A. Oltramari. Descriptive ontology for linguistic and cognitive engineering. http://www.loa.istc.cnr.it/DOLCE.html.

[49] 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.

[50] J. McCarthy. Circumscription: A Form of Non-monotonic Reasoning. *Artificial Intelligence*, 13:27–39, 1980.

[51] J. Meseguer. General logics. In *Logic Colloquium 87*, pages 275–329, 1989.

[52] T. Mossakowski. Relating CASL with Other Specification Languages: the Institution Level. *Theoretical Computer Science*, 286:367–475, 2002.

[53] T. Mossakowski, M. Codescu, O. Kutz, C. Lange, and M. Gruninger. Proof support for common logic. In *Proc. Wsh. Automated Reasoning for Quantified Non-Classical Logic (ARQNL)*, 2014.

[54] T. Mossakowski, M. Codescu, F. Neuhaus, and O. Kutz. The distributed ontology, modelling and specification language - DOL. In A. Koslow and A. Buchsbaum, editors, *The Road to Universal Logic–Festschrift for 50th birthday of Jean-Yves Beziau, Volume II*, Studies in Universal Logic. Birkhäuser, 2015.

[55] T. Mossakowski, U. Krumnack, and T. 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.

[56] T. Mossakowski and O. Kutz. The Onto-Logical Translation Graph. In *Modular Ontologies—Proceedings of the Fifth International Workshop (WoMO 2011)*, volume 230 of *Frontiers in Artificial Intelligence and Applications*, pages 94–109. IOS Press, 2011.

[57] T. Mossakowski, O. Kutz, M. Codescu, and C. Lange. The Distributed Ontology, Modeling and Specification Language. In C. Del Vescovo et al., editor, *Proceedings of the 7th International Workshop on Modular Ontologies (WoMO-13)*, volume 1081. CEUR-WS, 2013.

[58] T. Mossakowski, C. Lange, and O. Kutz. Three Semantics for the Core of the Distributed Ontology Language. In M. Donnelly and G. Guizzardi, editors, *7th International Conference on Formal Ontology in Information Systems (FOIS)*, volume 239 of *Frontiers in Artificial Intelligence and Applications*, pages 337–352. IOS Press, 2012. FOIS Best Paper Award.

[59] T. Mossakowski and L. Schröder. On inconsistency and unsatisfiability. *International Journal of Software and Informatics*, 9(2), 2015.

[60] T. Mossakowski and A. Tarlecki. Heterogeneous logical environments for distributed specifications. In A. Corradini and U. Montanari, editors, *WADT 2008*, number 5486 in Lecture Notes in Computer Science, pages 266–289. Springer, 2009.

[61] B. Motik, P. F. Patel-Schneider, and B. Cuenca Grau. OWL 2 web ontology language: Direct semantics. W3C recommendation, World Wide Web Consortium (W3C), October 2009.

[62] B. Motik, P. F. Patel-Schneider, and B. Parsia. OWL 2 web ontology language: Structural specification and functional-style syntax. W3C recommendation, World Wide Web Consortium (W3C), October 2009.

[63] OMG. Unified Modeling Language, Version 2.5. Specification formal/2015-03-01, Object Management Group, 2015.

[64] R. S. Pressman. *Software Engineering: A Practitioner's Approach.* The McGraw-Hill Companies, 8th edition, 2014.

[65] F. Rabe and M. Kohlhase. A scalable module system. *Information & Computation*, pages 1–95, 2013.

[66] D. Sannella and A. Tarlecki. *Foundations of Algebraic Specification and Formal Software Development*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2012.

[67] M. Schneider, S. Rudolph, and G. Sutcliffe. Modeling in OWL 2 without restrictions. In M. Rodriguez-Muro, S. Jupp, and K. 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.

[68] E. Seidewitz. Model Semantics and Mathematical Logic. *Syntax And Semantics*, pages 1–13, August 2008.

[69] H. Stuckenschmidt, C. Parent, and S. Spaccapietra, editors. *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, volume 5445 of *Lecture Notes in Computer Science*. Springer, 2009.

[70] 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.

[71] G. Sutcliffe. Tptp documents file: Syntaxbnf. http://www.cs.miami.edu/~tptp/cgi-bin/SeeTPTP?Category=Documents&File=SyntaxBNF.

[72] G. Sutcliffe. The tptp problem library for automated theorem proving. http://www.tptp.org.

[73] G. Sutcliffe. The TPTP problem library and associated infrastructure. *J. Autom. Reasoning*, 43(4):337–362, 2009.

[74] G. Sutcliffe, C. B. Suttner, and T. Yemenis. The TPTP problem library. In A. 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.

[75] 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.

[76] J. Zedlitz, J. Jörke, and N. Luttenberger. From uml to owl 2. In *Knowledge Technology*, pages 154–163. Springer, 2012.

[77] 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.