Date: Tuesday 17th November, 2015

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

Version 0.99

OMG Document Number: ad/~~2015-08-01~~ 2015-10-01
Machine readable files~~(normative)~~:
Normative: ad/~~2015-08-03, ad-2015-07-02~~2015-10-02
Informative: ad/2015-10-03

OMG's Issue Reporting Procedure

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

Table of Contents

Preface

OMG

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

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

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

OMG Specifications

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

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

Specifications are organized by the following categories:

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

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

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

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

Typographical Conventions

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

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

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

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

Helvetica/Arial - 10 pt.: Exceptions

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

Issues

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

0 Submission-Specific Material

0.1 Submission Preface

Fraunhofer FOKUS, MITRE, and Thematix Partners LLC are pleased to submit this joint proposal in response to the Ontology, Modeling and Specification Integration and Interoperability (OntoIOp) RFP (OMG document ad/2013-12-02). The joint proposal is supported by Athan Services and the Otto-von-Guericke University Magdeburg. The contacts for this submission are:

- Fraunhofer FOKUS, Andreas Hoffmann, andreas.hoffmann@fokus.fraunhofer.de
- MITRE, Leo Obrst, lobrst@mitre.org
- Thematix Partners LLC, Elisa Kendall, ekendall@thematix.com
- Athan Services, Tara Athan, taraathan@gmail.com
- Otto-von-Guericke University Magdeburg, Till Mossakowski, till@iws.cs.uni-magdeburg.de (*lead contact*)

0.2 Mandatory Requirements

ID	RFP requirement	How this proposal addresses requirement
6.5.1(a)	Proposals shall provide a specification of a metalanguage for relationships between the components of logically heterogeneous OMS, particularly, given a language translation from a language L1 to another language L2, the application of the language translation to an OMS that is written in the language L1.	DOL provides the required translation construct using syntax `O with translation t`, see **??** and **??**. Moreover, DOL provides heterogeneous interpretations between OMS, see **??** and **??**.
6.5.1(b)	Proposals shall provide a specification of a metalanguage for the union of OMS written in different languages, which implicitly involves the application of suitable default translations in order to reach a common target language.	The syntax for unions is `O1 and O2`, see **??** and **??**. Default translations are discussed in **??**, and DOL's notion of heterogeneous logical environment explicitly specifies default translations, see **??**.
6.5.1(c)	Proposals shall provide a specification of a metalanguage for importation in modular OMS.	DOL allows the import of OMS by their IRI, see **??** and **??**.
6.5.1(d)	Proposals shall provide a specification of a metalanguage for relationships between OMS and their extracted modules e.g. the whole theory is a conservative extension of the module.	DOL provides such a construct with syntax `module m : o1 of o2 for sig`, see **??** and **??**.
6.5.1(e)	Proposals shall provide a specification of a metalanguage for relationships between OMS and their approximation in less expressive languages such that the approximation is logically implied by the original theory, where the approximation generally has to be maximal in some suitable sense.	DOL provides such a construct with syntax `o keep logic`, see **??** and **??**.
6.5.1(f)	Proposals shall provide a specification of a metalanguage for links such as imports, interpretations, refinements, and alignments between OMS/modules.	DOL covers several metalogical relationships, namely entailments, interpretations, equivalences, refinements, alignments and module relations, see **??** and **??**.
6.5.1(g)	Proposals shall provide a specification of a metalanguage for combination of OMS along links.	DOL provides such a construct with syntax `combine n`, where `n` is a network of OMS and mappings (links), see **??** and **??**.
6.5.2(a)	The constructs of the metalanguage shall be applicable to different logics.	The semantics of DOL is based on a heterogeneous logical environment, which can contain arbitrary logics, see **??**.

Continued on next page

Table 0.1 – *Continued from previous page*

ID	RFP requirement	How this proposal addresses requirement
6.5.2(b)	The metalanguage shall neither be restricted to OMS in a specific domain, nor to OMS represented in a specific logical language.	The semantics of DOL is based on a heterogeneous logical environment, which can contain arbitrary logics, see ??.
6.5.2(c)	The metalanguage shall not replace the object language constructs of the conforming logical languages.	The syntax of a `NativeDocument` is left unspecified in this standard. Rather, here this standard relies on other standards and language definitions. See ?? and ??.
6.5.2(d)	The metalanguage shall provide syntactic constructs for (i) structuring OMS regardless of the logic in which their sentences are formalized and (ii) basic and structured OMS and facilities to identify them in a globally unique way.	The structuring constructs for OMS in ?? and ?? can be used for any logic, see the semantics in ??. DOL uses IRIs for referencing both basic and structured OMS, see ??.
6.5.3(a)	An abstract syntax specified as an SMOF compliant meta model.	The abstract syntax is specified using SMOF, see clause ??. An EBNF variant is given in annex ??.
6.5.3(b)	A human-readable lexical concrete syntax in EBNF and serialization in XML, for the latter XMI shall be used.	The concrete syntax (in EBNF) is specified in clause ??. The XMI representation is automatically derived from the SMOF meta model.
6.5.3(c)	Complete round-trip mappings from the human-readable concrete syntax to the abstract syntax and vice versa.	The metaclasses of the MOF abstract syntax are used as non-terminals of the EBNF concrete syntax (clause ??); this makes a round-trip mapping between both straight-forward. Moreover, the round-trip mapping has been implemented in form of a parser and a printer as part of the heterogeneous tool set (see appendix ?? and `http://hets.eu`).
6.5.3(d)	A formal semantics for the abstract syntax.	The formal semantics is given in clause ??.
6.5.4(a)	Existing OMS in existing serializations shall validate as OMS in the metalanguage with a minimum amount of syntactic adaptation.	Any document providing an OMS in a serialization of a DOL conforming language can be used as-is in DOL, by reference to its IRI. See ??.
6.5.4(b)	It shall be possible to refer to existing files/documents from an OMS implemented in the metalanguage without the need for modifying these files/-documents.	Documents can be referenced by IRIs, see ??.
6.5.4(c)	Translations between logical languages shall preserve (possibly to different degrees) the semantics of the logical languages. Between a given pair of logical languages, several translations are possible.	The semantics of DOL is based on a heterogeneous logical environment, which contains institution comorphisms as translations, see ??. Institution comorphisms preserve semantics in a weak form through their satisfaction condition. The DOL Ontology specifies properties of translations (comorphisms) preserving more and more of the semantics, see annex ??.
6.5.5(a)	Informative annexes shall establish the conformance of a number of relevant logical languages. An initial set of language translations may be part of an informative annex.	For conformance of logical languages, see 6.5.5(b) below. Conformance of some translations is established in annex ??.
6.5.5(b)	Conformance of the following subset of logical languages shall be established: OWL2 (with profiles EL, RL, QL), CLIF, RDF, UML class diagrams.	Conformance of the following languages is established: OWL 2 (annex ??), CLIF (annex ??), RDF and RDF Schema (annex ??), UML class diagrams (annex ??).
6.5.5(c)	Conformance of a suitable set of translations among the languages mentioned in the previous bullet point shall be established.	Conformance of some translations is established in annex ??.
6.5.6	Existing standards and best practices for allocating globally unique identifiers shall be reused. The same standards and best practices shall also be applied to associate different representations of the same content to one unique identifier.	DOL uses IRIs to reference documents (both DOL documents, as well as documents written in some conforming language). See ??.

0.3 Optional Requirements

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

0.4 Issues to be Discussed

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

0.5 Evaluation Criteria

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

0.6 Proof of Concept

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

0.7 Changes to Adopted OMG Specifications

This specification proposes no changes to adopted OMG specifications.

1 Scope

This OMG Specification specifies the Distributed Ontology, Modeling and Specification Language (DOL). DOL is designed to achieve integration and interoperability of ontologies, specifications and MDE 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 [?].

1.1 Background Information

Logical languages are used in several fields of computing for the development of formal, machine-processable texts that carry a formal semantics. Among those fields are 1) **O**ntologies formalizing domain knowledge, 2) (formal) **M**odels of systems, and 3) the formal **S**pecification of systems. Ontologies, MDE 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, MDE models) to properties and behaviors of hardware and software systems (MDE 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 MDE models are developed to represent different viewpoints or perspectives on a domain or system. Hence, interoperability becomes a crucial issue, in particular, formal interoperability, i.e. interoperability that is based on the formal semantics of the different viewpoints. Interoperability is both about the ability to interface different domains and systems and the ability to use several OMS in a common application scenario. Further, interoperability is about coherence and consistency, ensuring at an early stage of the development that a coherent system can be reached.

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

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

1.2 Features Within Scope

The following are within the scope of this OMG Specification:

1. homogeneous OMS as well as heterogeneous OMS (OMS that consist of parts 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

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

2.1 Conformance of an OMS Language/a Logic with DOL

Rationale: for an OMS language to conform with DOL,

- its logical language aspect either needs to satisfy certain criteria 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 **??**);
2. **serialization conformance:** it has at least one serialization in the sense of section 2.2;
3. **semantic conformance:** either there exists a translation of it into a conforming language[1], 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 **??** in clause **??** , and there is a mapping from the abstract syntax of the OMS language to signatures and sentences of the institution. Note that one OMS language can have several sublanguages or profiles corresponding to several logics (for example, OWL 2 has profiles EL, RL and QL, apart from the whole OWL 2 itself).

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

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

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 2.3.

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

2.1.1 Conformance of language/logic translations with DOL

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

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

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

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

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).

2.2 Conformance of a Serialization of an OMS Language With DOL

Rationale: The main reason for the following specifications is identifier injection. DOL is capable of assigning identifiers to entities (symbols, axioms, modules, etc.) inside fragments of OMS languages that occur in a DOL document, even if that OMS language 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 ~~any elements. (This is~~ 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 **??**. ~~)~~

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 **??** 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 ~~An OMS language is standoff markup conforming with if one of its serializations~~ 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 w.r.t. CURIEs (compact URIs) as a means of abbreviating IRIs (grammar specified in clause **??**), listed from highest to lowest:

Prefixed CURIE conformance The given serialization allows non-logical symbol identifiers to have the syntactic form of a CURIE, or any subset of the CURIE grammar that allows named prefixes (`prefix:reference`, where a declaration of DOL-conformance of a serialization **may** redefine the separator character to a character different from `:`). A serialization that conforms w.r.t. a 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 **??**)
- 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 2.3.

2.3 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 DOLregistry.

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 ~~. As a minimal requirement, there must be a~~ **NR1**. Further, it is required that the language/logic/serialization is registered in the DOL registry (see annex **??**).

There may be an RDF description conforming to the vocabulary specified in annex **??**. That description ~~must~~ 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.

2.4 Conformance of a Document With DOL

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

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

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

2.5 Conformance of an Application With DOL

In the ~~sequel~~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 **??**. 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 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 **??**. 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 2.4. 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.

3 Normative References

NR1 W3C/TR REC-ldp-20150226:2015 Linked Data Platform 1.0. W3C Recommendation, 26 February 2015.
`http://www.w3.org/TR/2015/REC-ldp-20150226/`

NR2 W3C/TR REC-owl2-syntax:2012 OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax (Second Edition). W3C Recommendation, ~~27 October 2009.~~11 December 2012.
`http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/`

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

NR4 W3C/TR REC-xml:2008 Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C Recommendation, 26 November 2008.
`http://www.w3.org/TR/2008/REC-xml-20081126/`

NR5 W3C/TR REC-owl2-primer:2012 OWL 2 Web Ontology Language: Primer (Second Edition). W3C Recommendation, 11 December 2012.
`http://www.w3.org/TR/2012/REC-owl2-primer-20121211/`

NR6 W3C/TR REC-owl2-profiles:2012 OWL 2 Web Ontology Language: Profiles (Second Edition). W3C Recommendation, ~~27 October 2009.~~ 11 December 2012.
`http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/`

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 REC-xptr-framework:2003 XPointer Framework. W3C Recommendation, 25 March 2003.
`http://www.w3.org/TR/2003/REC-xptr-framework-20030325/`

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 – ~~Second~~ Third Edition. Syntax and processing rules for embedding RDF through attributes. W3C Recommendation, ~~22 August 2013.~~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 REC-rdf-schema:2014 RDF Schema 1.1. W3C Recommendation, 25 February 2014.
`http://www.w3.org/TR/2014/REC-rdf-schema-20140225/`

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 REC-skos-reference:2009 SKOS Simple Knowledge Organization System Reference. W3C Recommendation, 18 August 2009
`http://www.w3.org/TR/2009/REC-skos-reference-20090818/`

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

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

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 MDE 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 ~~diagram~~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 ~~diagrams~~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 logical theory that is used as a shard conceptualization

MDE model logical theory that is used as an abstract representation of a domain or of a system, in the sense of model-driven engineering (MDE)
NOTE Not to be confused with the term model in the sense of logic (model theory).

specification logical theory that is used to express formal constraint in mathematical structures, software systems and/or hardware systems

OMS (ontology, specification or MDE model) basic OMS or structured OMS.
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 model class 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 model.
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.

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

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 model, i.e. which is assigned a truth value in this model.
NOTE In a model, on the one hand, a sentence is always true or false. In an OMS, on the other hand, a sentence can have several logical statuses. 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 models and sentences indicating which sentences hold true in the model.
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 model 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 model).

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 ~~consequence-theoretic~~ 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 models of an OMS.
NOTE An extension O_2 of an OMS O_1 is a model-theoretic conservative extension, if each model of O_1 can be expanded to a model 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 model of O_1 can be expanded to a model of O_2 that is unique up to isomorphism.
NOTE Each monomorphic extension is also a model-theoretic conservative extension but not vice versa.

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

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

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

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 [?] for a more detailed discussion.

satisfiability property of an OMS expressing that it is satisfied by least one model.
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 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 ~~consequence-theoretic~~ 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 [?].
NOTE The goal of module extraction is "decomposing an OMS into smaller, more manageable modules with appropriate dependencies" [?].
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 [?].

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 models 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 [?], [?].

minimization form of closure that restricts the models to those that are minimal (with respect to the local environment).

maximization form of closure that restricts the models to those that are maximal (with respect to the local environment).

freeness special type of closure, restriction of models 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 models 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 [?].

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. [?] 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 [?, ?].

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 [?], 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), models (interpretations of these), sentences (constraints on models), and a satisfaction relation between models and sentences.
NOTE Most OMS languages have an underlying logic.
EXAMPLE $\mathcal{SROIQ}(D)$ is the logic underlying OWL 2 DL.
NOTE See annex **??** 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, model, 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, model, sentence and satisfaction.

NOTE See Definition **??** in clause **??** 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 models) 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 models) that forgets those parts of the logical structure not fitting the target logic.
~~translation that maps signatures of the source logic to theories (i.e. signatures and sets of sentences) of 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 **??**.

faithful mapping logic mapping that preserves and reflects logical consequence.

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

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

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 model translations) 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 model translations, 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 MDE models intersect and there is a need to register details of the correspondence between the structures in the MDE models that semantically represent this overlap. Within these standards, an MDE 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 the global knowledge about OMS.

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**[1].
NOTE The linked data principles (adapted from **NR1** and its paraphrase at [?]) are the following:

1. Use IRIs as names for things.

[1] The original source is widely accepted but not formally a standard [?] .

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 [?], is the most common format for publishing linked data. However, its usage is not mandatory.
NOTE Using HTTP content negotiation [?] 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 (subject, predicate, object) triple as defined by **NR14**, Section 3.1. The subject of an annotation is an element of an OMS. The predicate is an RDF property defined in an external OMS and describes in what way the annotation object is related to the annotation subject.
NOTE According to ~~note 4.11~~ 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 [?].

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 [?]
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) [?]
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 [?]
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

6 Additional Information

(Informative)

6.1 Changes to Adopted OMG Specifications

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

6.2 How to Read This Specification

The initial 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 **??**).

Clause **??** 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 **??** contains the abstract syntax specified using Extended Backus–Naur Form (EBNF) (*informative*).

Clause **??** 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 **??** is about the DOLregistry, which allows to register DOLconforming languages and translations (*normative*).

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

Various languages are shown to conform to DOL in informative annexes: OWL2 (annex **??**), Common Logic (annex **??**), RDF and RDF Schema (annex **??**), UML class ~~diagrams~~ models (annex **??**), TPTP (annex **??**), and CASL (annex **??**).

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

Annex **??** 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 **??** provides of DOL texts, which provide examples for all DOL constructs, which are specified in the abstract syntax (*informative*). ~~Annex **??** sketches scenarios that outline how is intended to be applied (*informative*). For each scenario, a brief description is provided, and the utilized features as well as the status of its implementation are listed.~~

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

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

6.3 Acknowledgments

6.3.1 Submitting and supporting organizations

The following OMG members are submitting this specification:

- Fraunhofer FOKUS
- MITRE
- Thematix Partners LLC

The following organizations are supporting this specification:

- Otto-von-Guericke University Magdeburg
- Athan Services

6.3.2 Participants

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

- Tara Athan, Athan Services, USA
- Conrad Bock, National Institute of Standards and Technology, USA
- Mihai Codescu, ~~Otto-von-Guericke University Magdeburg, Germany~~ 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 ~~Bolzano~~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

7 Goals and Usage Scenarios

(Informative)

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

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

In order to achieve interoperability during ontology development it is often necessary to describe concepts in a language more expressive than OWL. Therefore, it is common practice to informally annotate OWL ontologies with FOL axioms (e.g., Keet's mereotopological ontology ~~Part-Whole~~[?] , Dolce Lite ~~Dolce-lite~~[?] , BFO-OWL). OWL is used because of better tool support, FOL because of greater expressiveness. However, relegating FOL axioms to informal annotations means that these are not available for machine processing. Another example of this problem is the following: For formally representing concept schemes (including taxonomies, thesauri and classification schemes) and provenance information there are the two W3C standards SKOS (Simple Knowledge Organization System; **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:CommonLogic
```

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

OWL can express transitivity, but not together with asymmetry.

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

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

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

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

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

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

```
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
```

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

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

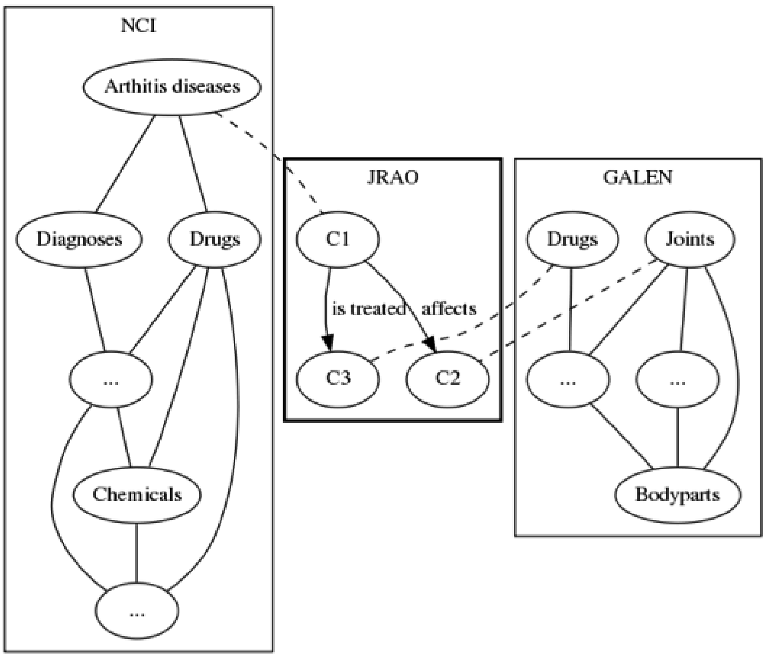


Figure 7.1: JRAO – Example for Module Extraction

```
  %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
```

```
 * ~~module myGalenIsAModule : myGalen of~~ *    * cons-ext myGalenIsAModule : <*  http://purl.bioontolo
>of myGalen*  for Drugs, Joints, Bodyparts
end
```

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

Data collection has become easier and much more widespread over the years. This data has to be assigned a meaning somehow, which occurs traditionally in the form of metadata annotations. For instance, consider geographical datasets derived from satellite data and raw sensor readings. Current implementations in, e.g., ecological economics [?] require manual annotation of datasets with the information relevant for their processes. While there have been attempts to standardize such information [?], 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 MDE model of the object of observation. These metadata require a more expressive language, such as OWL or Common Logic, which operate under an open-world semantics. However, it is unfeasible to translate the whole large dataset into OWL or first-order logic. To 'meet in the middle', it is possible to declare bridge rules (i.e., a mapping layer) that can link the metadata to the data. This approach can be used for intelligent data analysis that combines the data and metadata through querying the system. It enables the analysis of the data on the conceptual layer, instead of users having to learn the SQL/SPARQL query languages and how the data is stored. There are various tools and theories to realize this, which is collectively called Ontology-Based Data Access/Management, see also ~~OBDA~~[?] .

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

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

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

The Dublin Core Metadata terms, which have been formalized as an RDF Schema vocabulary, developed initially by the digital library community, are less comprehensive but more widely used than PROV (cf.~~Use Case Onto-1~~ subclause 7.1). The rules for translating Dublin Core to the OWL subset of PROV (and, with restrictions, vice versa) are not known to yield valid instances of the PROV data model, i.e. they are not known to yield OWL ontologies consistent with respect to the OWL axioms that capture part of the PROV data model. This may disrupt systems that would like to reason about the provenance of an entity, and thus the assessment of the entity's quality, reliability or trustworthiness. The Dublin Core to PROV ontology translation[5] is expressed partly by a symbol mapping and partly by FOL rules. These FOL rules are implemented by CONSTRUCT patterns in the SPARQL RDF query language.[6] SPARQL has a formal specification of the evaluation semantics of its algebraic expressions, which is different from the model-theoretic semantics of the OWL and RDF Schema languages; nevertheless SPARQL CONSTRUCT is a popular and immediately executable syntax for expressing translation rules between ontologies in RDF-based languages in a subset of FOL. DOL not only supports the reuse of the existing Dublin Core RDF Schema and PROV OWL ontologies as modules of a distributed ontology (= OMS network), but it is also able to support the description of the FOL translation rules in a sufficiently expressive ontology language, e.g. Common Logic, and thus enable formal verification of the translation from Dublin Core to PROV.

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

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

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") [?] . 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

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

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

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"?

DOLsupports 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.

7.7 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) [7] 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[8], without specifying a formal semantics for them:

module provenance author, date, version, description, keyword, parent ontology[9]

axiom source provenance name, author, year[10]

direct relations maps (signature morphisms), definitional extension, conservative extension, inconsistency between ontologies, imports, relative interpretation, faithful interpretation, definable equivalence

DOLprovides 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 DOLlibrary.

7.8 Use Case Spec-1: Modularity of Specifications

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

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

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

library Sorting

%DIF < % refinement from abstract sorting to insert sort
 * *  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)
```

[7] http://stl.mie.utoronto.ca/colore/

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

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

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

```
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.

7.9 Use Case Spec-2: Specification Refinements

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

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

A simple example is the refinement of the (purely declarative) sorting specification from use case in section 7.8 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

%DIF > % 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 Fig. **??**. 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 =Natthengenerated type Int ::= __ - __(Nat;Nat)forall a,b,c,d:  Nat.  a - b = c - d <=> a + d
= c + bsort Nat < Intforall a:  Nat .  a = a - 0end *
```

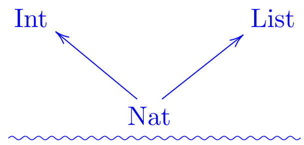


Figure 7.2: An OMS network consisting of three specifications

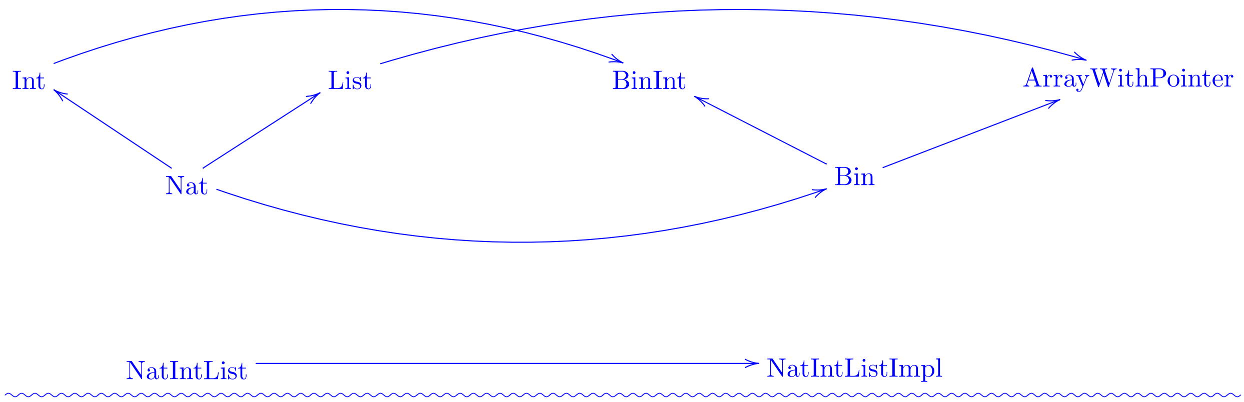


Figure 7.3: A refinement between OMS networks

```
* spec List =Natthensort Elemfree type List ::=  * [] * | __ :: __ (Elem; List)op #__:  List -> Nat;forall
x,:  Elem; L: List.  # [] = 0 .  # (x ::  L) = suc( # L ) end *
 * network NatIntList = Nat, Int, Listend *
```

The network in Fig. ?? can be refined to a network that is closer to implementation. This amounts to refining the specifications of the network individually, see Fig. ??. 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 ...endspec ArrayWithPointer = NatBin then ...endnetwork NatIntListImpl = NatBin,
IntBin, ArrayWithPointerendrefinement NetworkRefinement =NatIntList refined viaR2, Int refined via sort Int
|-> BinInt to IntBin,List via sort List |-> Array to ArrayWithPointerto NatIntListImplend *
```

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

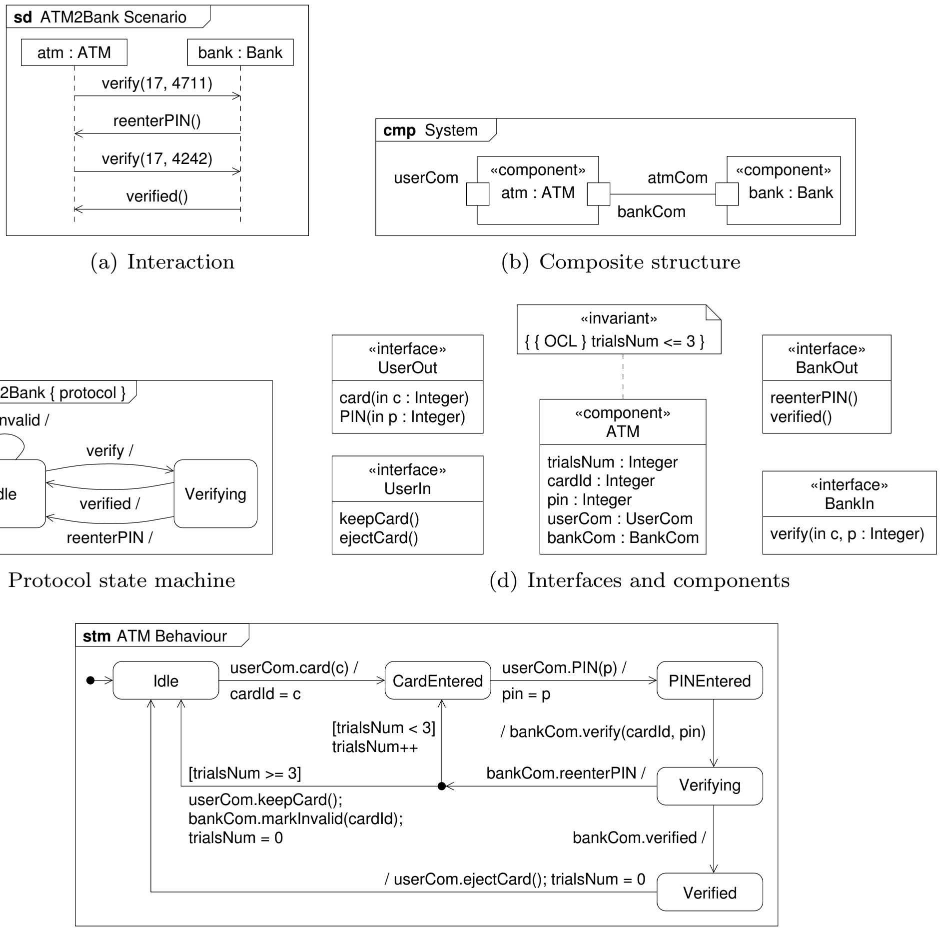
A typical UML model involves ~~diagrams~~ models of different types. Such UML models may have intrinsic errors because ~~diagrams~~ models of different types may specify conflicting requirements. Typical questions that arise in this context are ask for semantic consistency, e.g.,

- whether the multiplicities in a class ~~diagram are~~ model are semantically consistent with each other; ~~whether the attributes and operations in a state machine are available in a class diagram;~~
- whether the sequential composition of actions in an interaction diagram is justified by an accompanying OCL specification;
- whether cooperating state machines comply with pre-/post-conditions and invariants;
- whether the behavior prescribed in an interaction ~~diagram~~ model is realizable by several state machines cooperating according to a composite structure ~~diagram~~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 [?]. Once a formal semantics for the different ~~diagram~~ model types has been chosen (see, e.g. [?]), it is possible to use DOL to specify in which sense the ~~diagrams~~ models need to be consistent, and check this by suitable tools.

7.10.1 The ATM Example

The ATM example, which illustrates model-driven development using UML, is taken from [?]. 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 ?? shows a possible *interaction* between an atm and a bank object, which consists of four messages: the atm requests the bank to verify if a card and PIN number combination is valid, in the first case the bank requests to reenter the PIN, in the second case the verification is successful. This interaction presumes that the system has an atm and a bank as objects. This can, e.g., be ensured by a *composite structure* ~~*diagram*~~*model*, see Fig. ??, 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 Fig. ??. The protocol state machine fixes in which order the messages verify, verified, reenterPIN, and markInvalid between atm and bank may occur. Figure ?? provides structural information in form of ~~an interface~~ 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 MDE model elements have to implement. In our case, the ~~interface is~~ interfaces are described in a *class* ~~*diagram*~~*model*. ~~Here, the operation is~~ Its component type ATM is further enriched with the OCL constraint trialsNum <= 3, which refines its semantics ~~: can only be invoked if the OCL constraints holds~~requiring that trialsNum must not exceed three.

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

The ATM example in Fig. **??** 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/` ~~, that is under URL , where~~ 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
Fig. ??	sd	sequence diagram
Fig. ??	cmp	composite structure diagram
Fig. ??	psm	protocol state machine
Fig. ??	cd	class diagram
Fig. ??	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

view cd2stm = cd  * refined * to { atm hide along stm2cd} end
view 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 = ...
refinement r = ATM_network refined to Some_refined_ATM_network
entailment e = Some_refined_ATM_network entails ATM_network
```

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

7.11 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 ~~diagrams~~ 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 Fig. **??**, which is a refinement of the protocol state machine in Fig. **??**. This can be stated as follows in DOL. [11]

```
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.

[11] 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`

7.12 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 ~~(class diagrams~~ **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),
- 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 **??** 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  * @* endontology ExtendedTaxonomy = NoLivesto
thenObjectProperty *@ :  @* FinancialIntermediary.owns.ATMSubPropertyOf:  ownsDomain:  FinancialInte
ATMendontology clTaxonomy = ExtendedTaxonomy withtranslation OWL22CommonLogicoms JointModel =
clStateModel andclClassModel andclTaxonomyend
```

7.13 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 MDE 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.

8 Design Overview

(Informative)

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 **??**.

8.1 DOL in a Nutshell

As the usage scenarios in clause 7 illustrate, the use of multiple OMS may lead to lack of interoperability. The goal of DOL is to enable users to overcome these interoperability issues by providing a language for representing structured OMS and the relations between OMS as part of an OMS network in a semantically well-defined way. One particular challenge that needs to be addressed is that OMS are written in a wide variety of OMS languages, which differ in style, expressivity and logical properties. 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 ~~class diagrams~~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.[1] DOL provides a number of operations for this purpose; e.g., it is possible to define a structured OMS C as the union of an OWL ontology A and a Common Logic ontology B.
- DOL provides for defining connections between two OMS by using *OMS mappings*. DOL provides a variety of mappings; e.g., one can align terminology between different OMS or specify that some OMS is an extension of another. A set of OMS and OMS mappings may form together an *OMS network*.
- Native OMS inherit their semantics from the underlying OMS languages. The DOL operations for defining structured OMS, OMS mappings, and OMS networks have a declarative model-theoretic semantics, which is defined in clause **??**.

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.

8.2 Features of DOL

DOL is a language enabling OMS interoperability. DOL is

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.

[1] 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.

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

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

8.3 OMS Languages

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

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

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

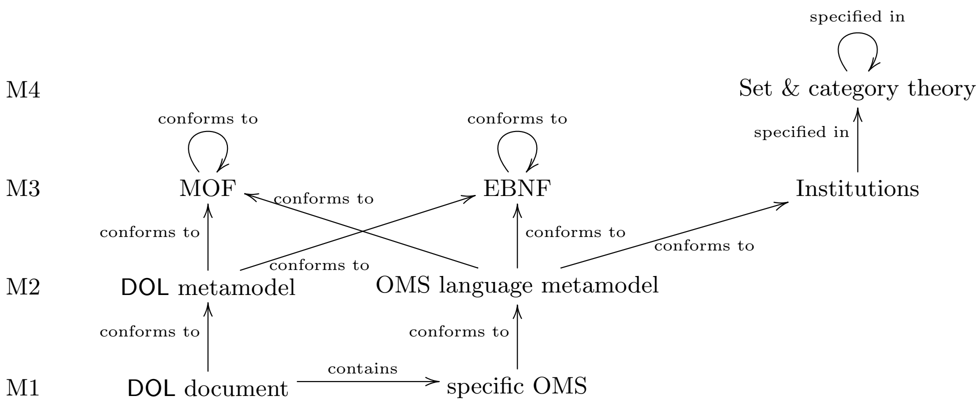
Annotation A means of enabling the attachment of human-readable descriptions to OMS symbols, addressing knowledge engineers and service developers, but also end users of OMS-based services.

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

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

8.4 DOL in the Metamodeling Hierarchy

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



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

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.

8.5 Semantic Foundations of DOL

A large variety of OMS languages in use can be captured at an abstract level using the concept of *institutions* [?]. 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 models and the sentences of a signature.

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

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

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

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

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 **??** 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 **??**). 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.

8.7 DOL Includes Provisions for Expressing Mappings Between OMS

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

- imports (particularly including imports that lead to conservative extensions), see the MOF metaclasses `OMSReference` and `ExtensionOMS` in clause **??**.
- interpretations (both between OMS and OMS networks), see the MOF metaclass `InterpretationDefinition` in clause **??**.
- alignments between OMS, see the MOF metaclass `AlignmentDefinition` in clause **??**.
- conservative extensions, e.g. mappings between OMS and their modules, see the MOF metaclass `ConservativeExtensionDefi` in clause **??**.

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

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 **??**.

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

DOL provides a mechanism for identifying anything of relevance in OMS by assigning an IRI to it. With RDF there is a standard mechanism for annotating things identified by IRIs. Thus, DOL supports annotations in the full generality specified in clause 4.11.

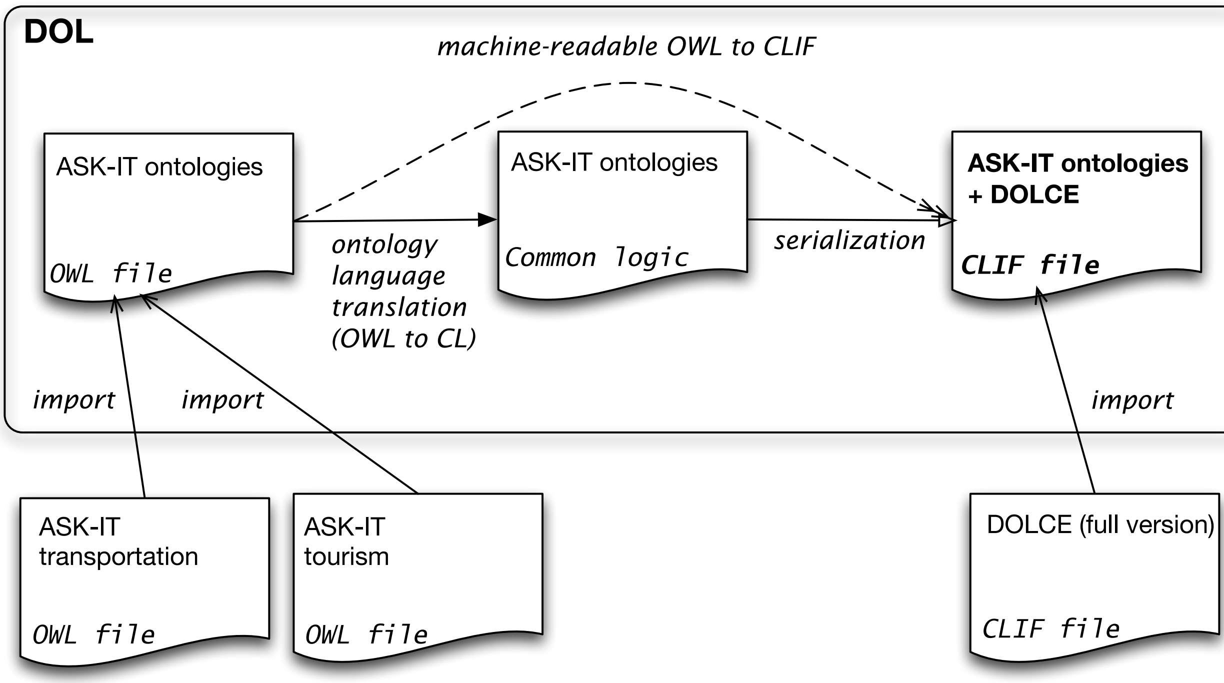


Figure 8.1: Mapping between two OMS formulated in different OMS languages

9 DOL Syntax

This clause specifies the DOL abstract syntax as a MOF **NR25** metamodel. In annex **??**, 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. The latter is only the case if the non-terminal translates to a single non-terminal (e.g. IRI or String) and is not part of an alternative in another rule. Each generalization in MOF yields a rule that consists solely of alternatives of single non-terminals. The properties of a class yield a EBNF rule for this class as a concatenation of the property types with syntactic terminals. Options are represented by cardinalities of the respective properties from 0 to 1 and respectively the *-Functor by cardinalities from 0 to *.

The DOL document types are as follows

MIME type *application/dol+text*

Filename extension .dol

9.1 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

~~Additionally, the MOF~~ The DOL metaclasses `NativeDocument` and `BasicOMS` are abstract metaclasses without any instances within the normative DOLmetamodel. In order to use DOLwith some specific conforming OMS language, the top-level MOF metaclass of the abstract ~~syntaxes of any conforming OMS languages~~ syntax of this language (cf. clause 2.1) ~~are subclasses~~ has to be a subclass (in the sense of SMOF multiple classification) of the DOL metaclass `NativeDocument`~~.~~ ~~If a~~, see Fig. ??. Likewise, if the conforming OMS language has a metaclass for basic OMS, this ~~is~~ has to be a subclass of the metaclass `BasicOMS`, see Fig. ??. See the informative annexes ?? to ?? for details.

9.2 Documents

9.2.1 Abstract Syntax

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

- a DOL library, or

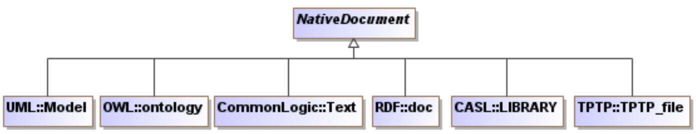


Figure 9.1: Informative diagram showing subclasses of `NativeDocument`

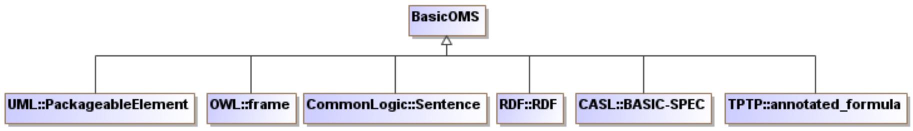


Figure 9.2: Informative diagram showing subclasses of `BasicOMS`

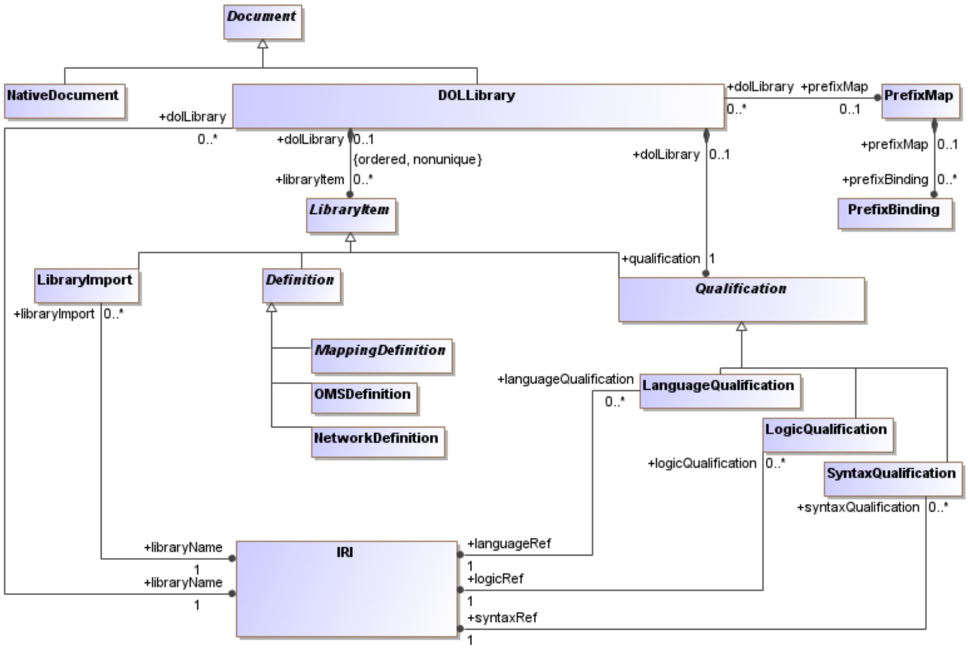


Figure 9.3: DOL metamodel: Documents and libraries

- a `NativeDocument`, which is the verbatim inclusion of an OMS written in an OMS language that conforms with DOL; cf. 2.1).

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 **??** informatively introduces `QueryRelatedDefinition`.

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

9.2.2 Concrete Syntax

9.2.2.1 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
```

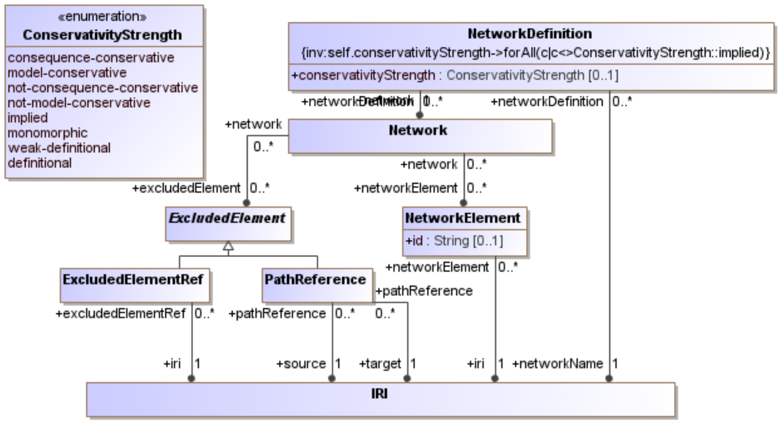


Figure 9.4:

```
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 ??>
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 [?] but differs from the RDFa Core 1.1 syntax.

9.3 OMS Networks

9.3.1 Abstract Syntax

The DOL metamodel for documents and libraries is shown in Fig. ??. 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 ?? 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 ~~disambugation~~ disambiguation in a `combination`. An optional `ConservativityStrength` specifies e.g. consistency of the network (analogously to `OMSDefinitions`, see clause ?? 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 7.9, ?? and Appendix ?? for an example network and of the use of `combination`.

9.3.2 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*
```

9.4 OMS

9.4.1 Abstract Syntax

The DOL metamodel for OMS is shown in Fig. ??. 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 ??, ??, for examples);
- a UnionsOMS, uniting two given OMS (see Appendix ?? for an example);
- a ClosureOMS, applying a closure operator (given by a Closure) to an OMS (see Appendix ?? and ?? 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 ??, ??, ?? 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 ?? for an example);
- an ApproximationOMS, applying an approximation operator (given by an Approximation) to an OMS (see Appendix ?? for an example);
- a CombinationOMS, giving a combination of (the OMS contained in) an OMS network (technically, this is a colimit, see [?]) (see Appendix ?? for an example of the use of combination);
- a ReductionOMS, applying a reduction (given by an Reduction) to an OMS (see use cases 7.8, 7.9 and ?? and Appendix ?? and ?? for examples);
- a ExtractionOMS, applying a module extraction operator (given by an Extraction) to an OMS (see use case 7.3 for an example);
- a QualifiedOMS, which is an OMS qualified with the OMS language that is used to express it.

Moreover, annex ?? informatively introduces Applications, which apply a ~~substition~~ 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 model using ConservativityStrength. More precisely, 'consequence-conservative' here requires the OMS to have only tautologies as signature-free logical consequences, while 'notconsequence-conservative' expresses that this is not the case. 'model-conservative' requires satisfiability of the OMS, 'not-model-conservative' its unsatisfiability. 'definitional' expresses that the OMS has a unique model (see Appendix ?? for an example); this may be interesting for characterizing OMS (e.g. returned by model finders) that are used to describe single models.

The DOL metamodel for extension OMS is shown in Fig. ??. 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)[1];
- a reference (through an IRI) to an OMS ~~existing on the Web~~ (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. 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 ?? for examples of the former two, and Annex ?? for examples of the latter two.

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

Using ExtendingOMS, ~~further OMS can be built:~~ extensions of an OMS with an ~~, which is an~~ ExtendingOMS ~~optionally~~ 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)~~. Recall that the local environment is the OMS built from all previously-declared symbols and axioms.~~

  ~~Furthermore, OMS can be constructed using~~ ;
- 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 model structure 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 Fig. ??, that for translation and reduction OMS in Fig. ??.

9.4.2 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~~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.

[1] In this place, any OMS in a conforming serialization of a conforming OMS language is permitted. However, DOL's module sublanguage should be ~~given preference over~~ 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.

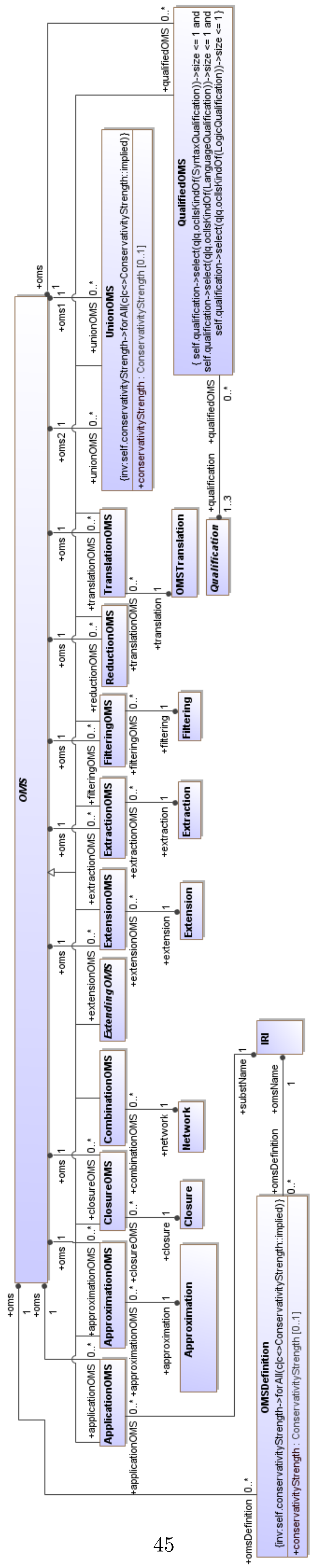


Figure 9.5: DOL metamodel: OMS

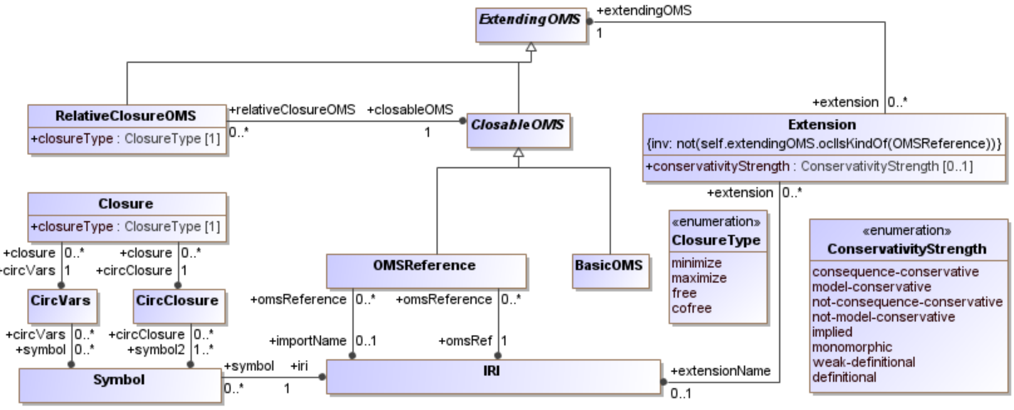


Figure 9.6: DOL metamodel: Extension and closure OMS

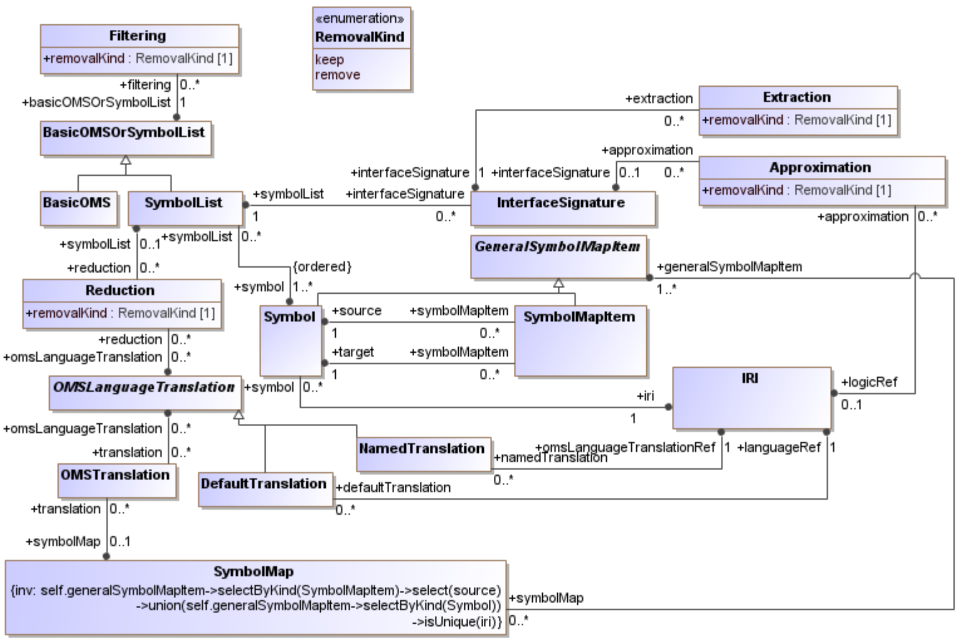


Figure 9.7: DOL metamodel: translation and reduction OMS

```
BasicOMS            ::= <language and serialization specific>
ClosableOMS         ::= 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*~~  +  LanguageTranslation ::=
Reduction           ::= 'hide' LogicReduction* SymbolList
                      | 'hide' LogicReduction ~~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           ::=  ~~'select' SymbolList~~  RemovalKind BasicOMSOrSymbolListRemovalKind ::=
'reject'  | 'select'
  ~~BasicOMS| 'reject'~~  BasicOMSOrSymbolList ::= '' BasicOMS '' |  SymbolList
  ~~| 'reject' BasicOMS~~ ExtensionOMS      ::= [ExtConservativityStrength]
                        [ExtensionName]
                        ExtendingOMS
ConservativityStrength ::= Conservative | '%mono' | '%wdef' | '%def'
ExtConservativityStrength ::= ConservativityStrength | '%implied'
Conservative        ::= '%DIF␣>␣cons'
                        | '  %ccons'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣'%mcons'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣'%notccons'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣'%notmcons'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣'%consistent'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣'%inconsistent'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣'%satifsiable'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣'%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.

9.5 OMS Mappings

9.5.1 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 ?? for an example), entailment (`EntailmentDefinition`, see use case ?? for an example), refinement (`RefinementDefinition`, see use cases 7.9 and ?? for examples) or equivalence (`EquivalenceDefinition`, see Annex ?? for an example), a named declaration of the relation between a module of an OMS and the whole OMS (`ConservativeExtensionDefinition`, see use case 7.3 for an example), or a named alignment (`AlignmentDefinition`, see use case 7.2 and Annex ?? for examples).

~~Both interpretation~~ The DOL metamodel for interpretations and refinements is shown in Fig. ??. 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 ?? 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 ~~composed (), provided that the target of the first refinement matches the source of the second one~~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 7.9 for an example). A refinement between OMS networks ~~has to specify both a mapping () between the nodes of the OMSnetwork, as well as,~~ consists of a list of ordinary refinements (between OMS), one for each node ~~, a symbol map from the OMS of that node to the target OMS to which it is mapped (this is a)~~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 7.9.

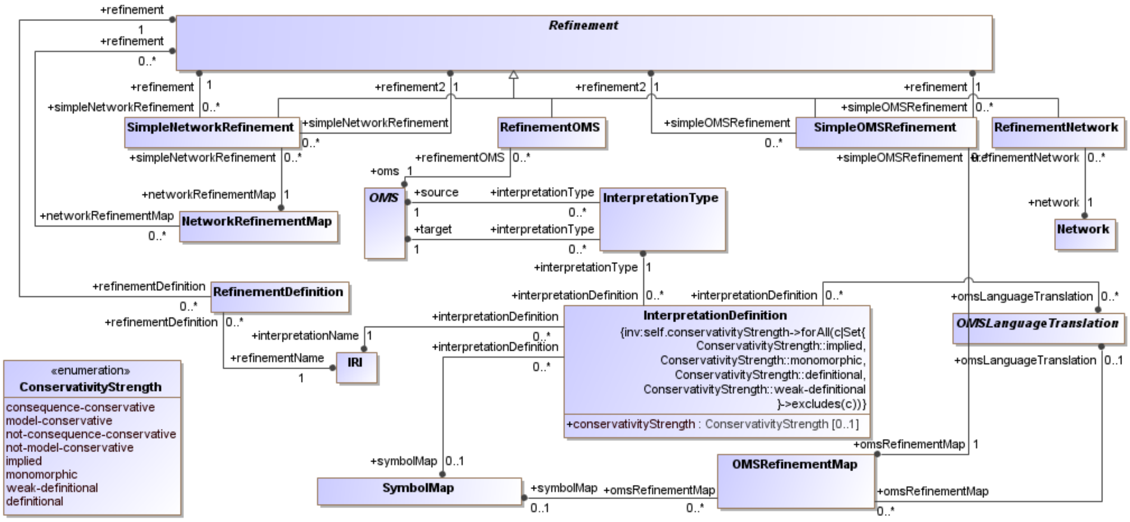


Figure 9.8: DOL metamodel: Interpretations and refinements

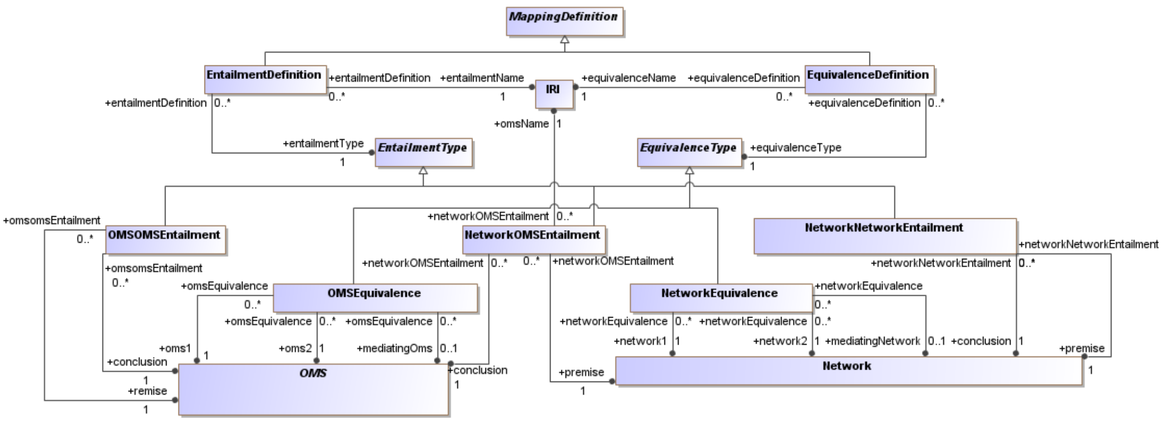


Figure 9.9: DOL metamodel: Entailments and equivalences

The DOL metamodel for entailments and equivalences is shown in Fig. ??. An entailment is a variant of an interpretation where all symbols are mapped identically, while an equivalence states that the model classes 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 models of the network, when restricted to a given node (given by an IRI), are models of the OMS.

The DOL metamodel for alignments is shown in Fig. ??. 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 [?]. The relations that can be used in a correspondence are equivalence, disjointness, subsumption, membership (the last two with a variant for each direction) or a user-defined relation that is stored in a registry and must be prefixed with `http://www.omg.org/spec/DOL/correspondences/`. A default correspondence can be used; it is applied to all pairs of non-logical symbols with the same local names. The default relation in a correspondence is equivalence, unless a different relation is specified in a surrounding 'CorrespondenceBlock'. Using an `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.

~~A~~ The DOL metamodel for conservative extension definitions is shown in Fig. ??. A `ConservativeExtensionDefinition`

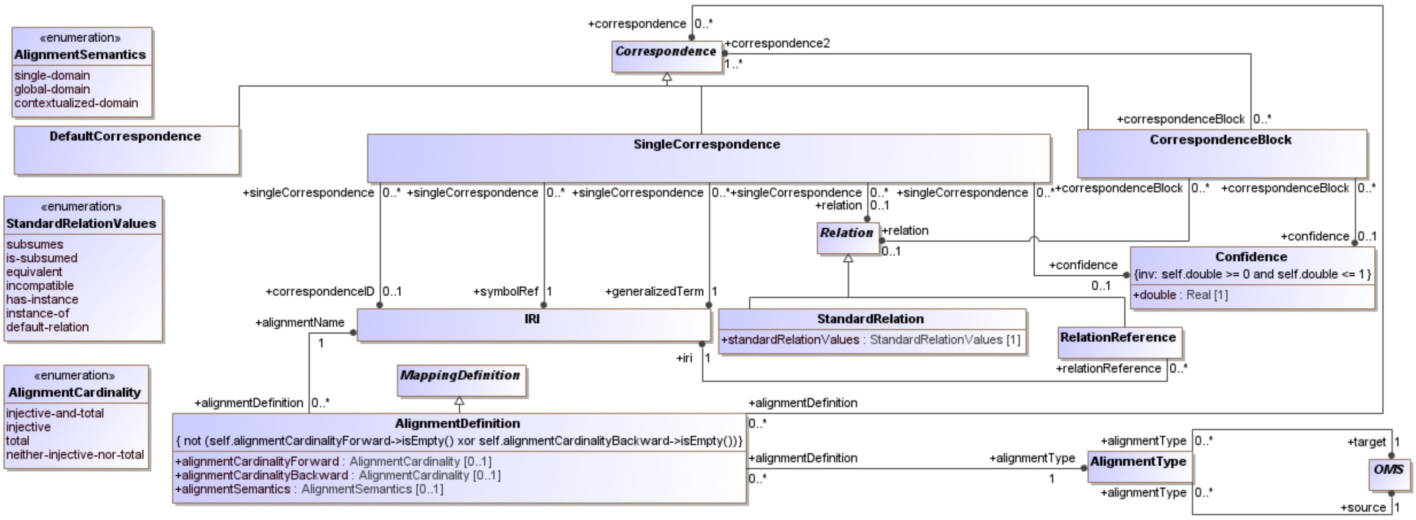


Figure 9.10: DOL metamodel: Alignments

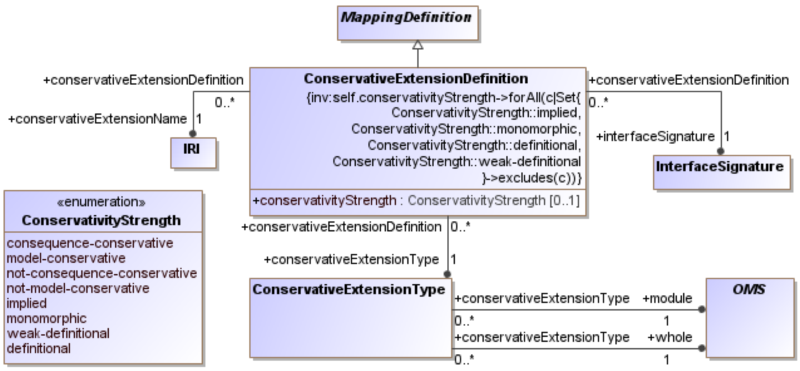


Figure 9.11: DOL metamodel: conservative extension definitions

declares that a certain ("~~module"~~whole) OMS actually is a ~~module of~~ conservative extension some other ("~~whole~~module") OMS with respect to the InterfaceSignature.

9.5.2 Concrete Syntax

MappingDefinition ::= InterpretationDefinition
| EntailmentDefinition
| EquivalenceDefinition
| ~~ModuleRelDefinition~~ ConservativeExtensionDefinition | AlignmentDefinitio
InterpretationDefinition ::= ~~InterpretationKeywordInterpretationNameConservative':'InterpretationTy~~
~~'end'~~ InlineInterpretation | RefinementDefinitionInlineInterpretation ::= InterpretationKeyw
[Conservative] ':' InterpretationType
['=' LanguageTranslation* [SymbolMap]] 'end'
~~|~~ RefinementDefinition ::= InterpretationKeyword InterpretationName '='
Refinement
'end'
InterpretationKeyword ::= 'interpretation' | 'view' | 'refinement'
InterpretationName ::= IRI
InterpretationType ::= GroupOMS 'to' GroupOMS
Refinement ::= GroupOMS
| NetworkName
| Refinement '␣ ~~then' Refinement | GroupOMS '~~ refined'

[RefMap] 'to' Refinement
| ~~NetworkName 'refined~~ Refinement 'interpreted '␣[RefMap]␣' ~~to~~
by '␣Refinement
RefMap␣␣␣␣␣␣␣␣␣␣␣␣␣::=␣'via'␣␣ (OMSRefinementMap | NetworkRefinementMap)OMSRefinementMap ::=
LanguageTranslation [SymbolMap]
| ~~'via'~~ [LanguageTranslation] SymbolMap
~~| 'via' NodeMap~~ NetworkRefinementMap ::= Refinement (','␣␣ ~~NodeMap~~ Refinement
)*
~~NodeMap ::= OMSName '|->' OMSName'using' LanguageTranslation* SymbolMap~~ EntailmentDefinition ::= '
entailment'␣EntailmentName␣'='
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣EntailmentType␣'end'
EntailmentName␣␣␣␣␣::=␣IRI
EntailmentType␣␣␣␣␣::=␣GroupOMS␣'entails'␣GroupOMS
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣OMSName␣'in'␣Network␣'entails'␣GroupOMS
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣|␣Network␣'entails'␣Network
EquivalenceDefinition␣::=␣'equivalence'␣EquivalenceName␣':'
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣EquivalenceType␣'end'
EquivalenceName␣␣␣␣::=␣IRI
EquivalenceType␣␣␣␣::=␣GroupOMS␣'<->'␣GroupOMS␣␣ ['='␣OMS␣] | Network '
<->'␣Network␣␣ ['='␣Network␣ ~~ModuleRelDefinition~~]ConservativeExtensionDefinition ::= '
~~module' ModuleName~~ cons-ext' ConservativeExtensionName [Conservative] ':'
~~ModuleType~~ ConservativeExtensionType 'for' InterfaceSignature
~~ModuleName~~ ConservativeExtensionName ::= IRI
~~ModuleType~~ ConservativeExtensionType ::= GroupOMS 'of' GroupOMS
AlignmentDefinition ::= 'alignment' AlignmentName
[~~AlignmentCardinalityPair~~ AlignmentCardinality AlignmentCardinality
] ':'
AlignmentType
['=' Correspondence (',' Correspondence)*]
['assuming' AlignmentSemantics] 'end'
AlignmentName ::= IRI
~~AlignmentCardinalityPair ::= AlignmentCardinalityForwardAlignmentCardinalityBackwardAlignmentCardi~~
~~::= AlignmentCardinalityAlignmentCardinalityBackward ::= AlignmentCardinality~~ AlignmentCardinality :
AlignmentType ::= GroupOMS 'to' GroupOMS
AlignmentSemantics ::= 'SingleDomain'
| 'GlobalDomain'
| 'ContextualizedDomain'
Correspondence ::= CorrespondenceBlock | SingleCorrespondence | DefaultCorrespondenceDefaultCo
::= '*'
CorrespondenceBlock ::= 'relation' [Relation] [Confidence] '{'
Correspondence (',' Correspondence)* '}'
SingleCorrespondence ::= ~~SymbolRef~~ Symbol [Relation] [Confidence]
GeneralizedTerm [CorrespondenceId]
CorrespondenceId ::= '%(' IRI ')%'
~~SymbolRef~~ Symbol ::= IRI
GeneralizedTerm ::= ~~SymbolRef~~ Symbol Relation ::= RelationReference | StandardRe
::= IRIStandardRelation ::= '>' | '<' | '=' | '%DIF␣<␣' | 'ni' | 'in' | IRI
%DIF > '␣|␣'ni'␣|␣'in'
Confidence␣␣␣␣␣␣␣␣␣::=␣Double
Double␣␣␣␣␣␣␣␣␣␣␣␣␣::=␣␣(␣\DIFdelbegin␣\DIFdel{(}\DIFdelend␣)␣␣␣(␣\DIFaddbegin␣\DIFadd{*}\DIFaddend␣)
}\DIFdelend) (\DIFaddbegin \DIFadd{@*
}\DIFaddend)

9.6 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.

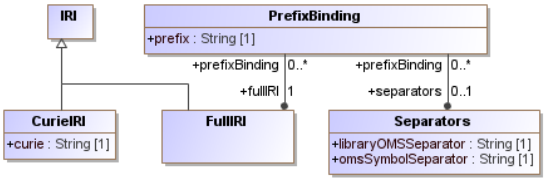


Figure 9.12: DOL metamodel: Prefixes

9.6.1 IRIs

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

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

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

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

9.6.2 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  * syntax *   * 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 **??**. The CURIE grammar of DOLis summarized in clause **??**.

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

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

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.

9.6.3 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 [**?**] 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 **??**).

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 2.2) **then**
      $sep \leftarrow cs$
    else if S forbids prefixed CURIEs **then**
      $sep \leftarrow$ undefined
    else
      $sep \leftarrow :$ {the standard CURIE separator character}
    end if
    {The following statements construct a modified EBNF grammar of CURIEs; see **NR3** for EBNF, and clause **??** 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 **??**) 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.

9.6.4 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 **??**).

9.7 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.[2] 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).

9.7.1 Key words and signs

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

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

9.7.1.1 Key words

Key words are always written lowercase. The following key words are reserved, and are not available for use as variables or as CURIEs with no prefix[3], although they can be used as parts of tokens.

```
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
module
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
```

[3]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.

Table 9.1: Key Signs

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

```
where
with
%DIF > cons
%ccons
%complete
%consistent
%def
%implied
%inconsistent
%mcons
%mono
%notccons
%notmcons
%prefix
%wdef
```

9.7.1.2 Key signs

Table ?? 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.

9.8 Integration of Serializations of Conforming Languages

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

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

XML conformance ~~Identifiers are added to XML elements by using the IRI-valued~~

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 ~~, or, if the serialization does not support this attribute, by adding a *dol:id* XML element~~ 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 ~~, containing~~ of the element, as specified in XML conformance requirement 3b, it is expected to contain exactly one text node ~~with the IRI~~ whose value is the IRI identifier.

It is a DOLsyntax 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[4] 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

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

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 ~~Standard mechanisms like XPointer () or~~ 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 ~~XML or~~ 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 ~~a use case for XPointer~~an example, consider the identification of ~~axioms~~ imports in the OWL~~2~~ /XML serialization [?], which does not provide a native way for assigning identifiers to ~~axioms. If, for~~ imports unless modified as suggested in annex ??. For example, in an OMS file ~~*birds*~~*cars.owx*, the ~~axiom~~ import <~~SubClassOf~~Import>~~<Class IRI="Penguin"~~http:/~~><Class IRI="FlightlessBird"~~/~~>~~example.org/engines</~~SubClassOf~~Import> ~~is the first one (in document order) to declare a superclass for *Penguin*, it~~ 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~~, whereas unique reference by the axiom's structure rather than by position would require a more complex expression~~. The same ~~axiom~~ import in the text-based OWL Manchester syntax [?] could be referred to as ~~according to~~ `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 ~~URI.~~ 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 ~~Common Logic sentences.)~~ sentences in the XCL serialization of Common Logic.)

10 DOL Semantics

DOL is a logical language with a precise formal semantics. The semantics gives DOL a rock-solid foundation, and provides increased trustworthiness in applications based on OMS written in DOL. The semantics of DOL is moreover the basis for formal interoperability, as well as for the meaningful use of logic-based tools for DOL, such as theorem provers, model-checkers, 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.

10.1 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, ~~respectively~~ logic translation ~~, respectively~~ 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 ~~diagrams~~models) can be equipped with one.

An abstract notion of logical system is given by the notion of satisfaction system [?], called 'rooms' in the terminology of [?]. They capture the Tarskian notion of satisfaction of a sentence in a model in an abstract way.

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

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

While this signature-free treatment enjoys simplicity and is wide-spread in the literature, many concepts and definitions found in logics, e.g. the notion of a conservative extension, involve the *vocabulary* or *signature* Σ used in sentences. Signatures can be extended with new non-logical symbols, or some of these symbols can be renamed; abstractly, this is captured using signature morphisms. Moreover, morphisms between models are also needed in order to give a semantics to `minimize`, `maximize`, `free` and `cofree`—these constructs use model morphisms to select certain models, e.g. the 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.*[1] *An* ***institution*** *[?] is a quadruple* $I = (\mathsf{Sig}, \mathbf{Sen}, \mathbf{Mod}, \models)$ *consisting of the following:*

- *a category*[2] 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* $\mathbf{Mod} : \mathsf{Sig}^{op} \to \mathbb{C}at$ *giving, for each signature* Σ*, the category of* models $\mathbf{Mod}(\Sigma)$*, and for each signature morphism* $\sigma\colon \Sigma \longrightarrow \Sigma'$*, the* reduct functor $\mathbf{Mod}(\sigma) : \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$*, where often* $\mathbf{Mod}(\sigma)(M')$ *is written as* $M'|_\sigma$*, and* $M'|_\sigma$ *is called the* σ-reduct *of* M'*, while* M' *is called a* σ-expansion *of* $M'|_\sigma$*,*

[1] 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.

[2] See [?, ?] for an introduction into category theory.

- *a satisfaction relation* $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ *for each* $\Sigma \in |\mathsf{Sig}|$,

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

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

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

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

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

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

- *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 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 [**?**].*

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.*[3] *Model reducts leave* UR*,* UD*, rel and fun untouched, while int and seq are composed with the appropriate signature morphism component.* □

Further examples of institutions are: $\mathcal{SROIQ}(D)$, unsorted first-order logic, many-sorted first-order logic, and many others. Note that ~~reduct~~ the reduct of a model is generally given by forgetting ~~parts of the model.~~ 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-models is the class of all Σ-models M such that $M \models \delta$, for each sentence $\delta \in \Delta$. A theory (Σ, Δ) is **consistent** if ~~there exists a~~ at least one Σ-model ~~M such that $M \models \varphi$ for $\varphi \in \Delta$~~ exists. **Semantic entailment** is defined as usual: for a theory $\Delta \subseteq \mathbf{Sen}(\Sigma)$ and $\varphi \in \mathbf{Sen}(\Sigma)$, Δ entails φ, written $\Delta \models \varphi$, if all models 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.

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

Definition 7 (Institution Comorphism) *An **institution comorphism** from an institution* $I = (\mathsf{Sig}^I, \mathsf{Mod}^I, \mathsf{Sen}^I, \models^I)$ *to an institution* $J = (\mathsf{Sig}^J, \mathsf{Mod}^J, \mathsf{Sen}^J, \models^J)$ *consists of a functor* $\Phi : \mathsf{Sig}^I \longrightarrow \mathsf{Sig}^J$*, and two natural transformations* ~~$\beta : \mathsf{Mod}^J \circ \Phi \Longrightarrow \mathsf{Mod}^I$~~ $\beta : \mathsf{Mod}^J \circ \Phi^{op} \Longrightarrow \mathsf{Mod}^I$ *and* $\alpha : \mathsf{Sen}^I \Longrightarrow \mathsf{Sen}^J \circ \Phi$*, such that for each* I*-signature* Σ*, each sentence* $\varphi \in \mathsf{Sen}^I(\Sigma)$ *and each model* ~~$M' \in \mathsf{Mod}^J(\Phi(\Sigma))$~~

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

$M' \in |\mathsf{Mod}^J(\Phi(\Sigma))|$

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

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

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

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

A comorphism is:

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

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

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

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

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

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

  admits (weak) amalgamation, i.e. any for any two models ~~$M_2 \in \mathsf{Mod}^I(\Sigma_2)$ and $M_1' \in \mathsf{Mod}^J(\Phi(\Sigma_1))$~~ $M_2 \in |\mathsf{Mod}^I(\Sigma_2)|$ and $M_1' \in |\mathsf{Mod}^J(\Phi(\Sigma_1))|$ with $M_2|_\sigma = \beta_{\Sigma_1}(M_1')$, there is a unique (not necessarily unique) ~~$M_2' \in \mathsf{Mod}^J(\Phi(\Sigma_2))$~~ $M_2' \in |\mathsf{Mod}^J(\Phi(\Sigma_2))|$ with $\beta_{\Sigma_2}(M_2') = M_2$ and $M_2'|_{\Phi(\sigma)} = M_1'$;

- a *subinstitution comorphism* if Φ is an embedding, each α_Σ is injective and each β_Σ is bijective[4];
- 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 $I = (\mathsf{Sig}^I, \mathsf{Mod}^I, \mathsf{Sen}^I, \models^I)$, the institution of its theories~~can be defined~~, 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, ~~that is~~ denoted* Th^I. *For each theory (Σ, Δ), its sentences are just Σ-sentences in I, and its models are just Σ-models in I that satisfy the sentences in Δ, while the (Σ, Δ)-satisfaction is the Σ-satisfaction of sentences in models of I.* □

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

Definition 9 *Let $I = (\mathsf{Sig}^I, \mathsf{Mod}^I, \mathsf{Sen}^I, \models^I)$ and $J = (\mathsf{Sig}^J, \mathsf{Mod}^J, \mathsf{Sen}^J, \models^J)$ be two institutions. ~~An~~ 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 $I = (\mathsf{Sig}^I, \mathsf{Mod}^I, \mathsf{Sen}^I, \models^I)$ to an institution $J = (\mathsf{Sig}^J, \mathsf{Mod}^J, \mathsf{Sen}^J, \models^J)$ consists of a functor $\Phi : \mathsf{Sig}^I \longrightarrow \mathsf{Sig}^J$, and two natural transformations ~~$\beta : \mathsf{Mod}^I \Longrightarrow \mathsf{Mod}^J \circ \Phi$~~ $\beta : \mathsf{Mod}^I \Longrightarrow \mathsf{Mod}^J \circ \Phi^{op}$ 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 model $M \in \mathsf{Mod}^I(\Sigma)$*

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

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

*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 [?].

~~A *network*[5] in a category C is a functor $D : G \longrightarrow C$, where G is a small category[5], and can be thought of as the shape of the graph of interconnections between the objects of C selected by the functor D. A *cocone* of a network $D : G \longrightarrow C$ consists of an object c of C and a family of morphisms $\alpha_i : D(i) \longrightarrow c$, for each object i of G, such that for each edge of the network, $e : i \longrightarrow i'$ it holds that $D(e);\alpha_{i'} = \alpha_i$. A *colimiting cocone* (or colimit) $(c, \{\alpha_i\}_{i\in|G|})$ can be intuitively understood as a minimal cocone, i.e.has the property that for any cocone $(d, \{\beta_i\}_{i\in|G|})$ there exists a unique morphism $\gamma : c \longrightarrow d$ 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 $\bullet \longleftarrow \bullet \longrightarrow \bullet$ with 3 objects and 2 non-identity arrows, G-colimits are called *pushouts*~~For a formal mathematical definition, see ??.

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

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

Definition 11 *Given a network $D : J \longrightarrow \mathsf{Sig}^I$, a family of models $\mathcal{M} = \{M_p\}_{j\in|J|}$ is* consistent with D *(or sometimes* compatible with D*) if for each node p of D, $M_p \in Mod(D(p))$ and for each edge $e : p \to q$, $M_p = M_q|_{D(e)}$. A cocone $(\Sigma, (\mu_j)_{j\in|J|})$ over the network $D : J \longrightarrow \mathsf{Sig}^I$ is called* weakly amalgamable *if it is mapped to a weak limit by* Mod. *For models, this means that for each D-compatible family of models $(M_j)_{j\in|J|}$, there is a Σ-model M, called an amalgamation of $(M_j)_{j\in|J|}$, with $M|_{\mu_j} = M_j$ ($j \in |J|$), and similarly for model morphisms. If this model is unique, the cocone is called* amalgamable. *I (or* Mod*) admits* (finite) (weak) amalgamation *if (finite) colimit cocones are (weakly) amalgamable. Finally, I is called* (weakly) semi-amalgamable *if it has pushouts and admits (weak) amalgamation for these.* □

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

[5]~~A network is called a diagram in category theory texts. This terminology is introduced to disambiguate OMS networks from UML diagrams.~~

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

[?] 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 models compatible with the network. This allows us to define a function *colimit* taking as argument a network of heterogeneous signatures and returning the selected weakly amalgamable cocone for the network and a function $\oplus$ taking as argument a family of models compatible with a network and returning its selected amalgamation.

~~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: An *inclusive category with symbols* is an inclusive category~~ ~~$\mathbb{C}$ equipped with a faithful functor $|\_| : \mathbb{C} \to \mathbb{S}et$[5] that preserves inclusions.~~

10.2 Semantics of DOL Language Constructs

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

A heterogeneous logical environment is given by a collection of OMS languages and OMS language translations[5], a collection of institutions, institution morphisms and institution comorphisms (serving as logics, logic reductions and logic translations), and a collection of serializations. Moreover, some of the institution comorphisms are marked as default translations (but only at most one between a given source and target institution), and there is a binary ~~supports relation~~ relation *supports* between OMS languages and institutions, and a binary ~~supports relation~~ 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.

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

~~For each logic in the heterogeneous logical environment, it is further required that there is:~~

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

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

~~where Σ is a signature giving the local environment, $\Sigma' \supseteq \Sigma$ is the resulting signature, and Δ' the resulting set of sentences, a function that turns a symbol map into a signature morphism , a relativization procedure taking as argument a theory and giving as result a theory, and three procedures for translating correspondences of alignments into sentences in the logic, as needed in Section ??.~~

~~Further, for each institution, it is required that there exist (possibly partial)~~ We are going to require existence of union and difference operations on ~~signatures~~the signatures of an institution in the heterogeneous logical environment. These concepts could be captured in a categorical setting using *inclusion systems* [?]. However, inclusion systems are too strong for the purposes of this specification. Therefore, weaker assumptions will be used.

Definition 12 *An* inclusive category *[?] is a category having a broad subcategory*[6] *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

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

---

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

[5] 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.

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

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

for each span where one arrow is an inclusion.

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

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

- Sig *is inclusive and has pushouts which preserve inclusions,*
- Sen *is inclusive, and*
- *each model category ~~have~~ 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 $\Sigma_1 \setminus \Sigma_2$.

~~This concludes the definition of~~ 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'-model 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* $|\_| : \mathbb{C} \to \mathbb{S}et$[7] *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 model class $\mathcal{M}_\emptyset$ 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 ~~the assumptions made about it~~ 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.
- 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 **??**.

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 models 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 models given by the model-level semantics is exactly the model class of the theory given by the theory-level semantics.

The following unifying notation is used for the two semantics of an OMS O:

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

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

Moreover, the semantics of O is the tuple $sem(O) = (I, \Sigma, \mathcal{M}, \Delta)$ where $\mathbf{Inst}(O) = I$, $\mathsf{Sig}(O) = \Sigma$, $\mathsf{Mod}(O) = \mathcal{M}$ 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 $\mathbf{Inst}(O)$, $\mathsf{Sig}(O)$, $\mathsf{Mod}(O)$ 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, $\mathsf{Mod}(O)$ can be obtained as $\mathsf{Mod}(O) = \{M \in \mathsf{Mod}(\mathsf{Sig}(O)) \mid M \models \mathsf{Th}(O)\}$.

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

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

and which has two kinds of edges:

- import links (written using single arrows, $S \to T$)
- 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 models of that node is the class of models 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~~(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 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 $\Gamma.prefix$, 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 ~~Γ in~~ `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 $\Gamma_1.prefix$ and $\Gamma_2.prefix$ 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 $\Gamma_1 \cup \Gamma_2.prefix = \Gamma_1.prefix \cup \Gamma_2.prefix$. $\Gamma.\{prefix = PMap\}$ 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 ~~structured OMS O~~ document D in a language L, logic L' and serialization S, ~~$sem_{(L,L',S)}(O)$~~ $semNative_{(L,L',S)}(D)$ denotes the language-specific semantics of ~~O. Further, assumes similar language-specific semantics of a basic OMS fragment O in the context of previous declarations, which is denoted $sem^{(\mathcal{I},\Sigma,\mathcal{M},\Delta)}_{(L,L',S)}(O)$.~~ D.

10.2.1 Semantics of Documents

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

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

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

,
$postfixLogicIRI(o, l)$ 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 ~~library~~ native document does not conform with the logic determined by the extension of the ~~library~~file where the document is stored, $sem(nativeDocument)$ will be undefined.

The rule for `DOLLibrary` is given below.

10.2.1.1 Semantics of libraries

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

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*,

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

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

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.

10.2.1.2 Semantics of lists of library items

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

If $libItem_1, \ldots, libItem_n$ are all `LibraryItems`,

$$sem(\Gamma, Sequence\{libItem_1, \ldots, libItem_n\}) = \Gamma'$$

where
$sem(\Gamma, libItem_1) = \Gamma_1$,
$sem(\Gamma_1, libItem_2) = \Gamma_2, \ldots$
$sem(\Gamma_{n-1}, libItem_n) = \Gamma'$.

10.2.1.3 Semantics of library items

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

For a `LibraryImport` ~~*libImport*,~~ *libImport*,

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

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

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

10.2.1.4 Semantics of a list of qualifications

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

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

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

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

10.2.1.5 Semantics of qualifications

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

For a `LanguageQualification` q,

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

where $\Gamma' = \Gamma.\{current = (q.\mathtt{languageRef}, logic', ser')\}$ and

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

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

For a `LogicQualification` q,

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

where $\Gamma' = \Gamma.\{current = (lang', q.\mathtt{logicRef}, ser')\}$

$lang = lang(\Gamma.current)$, $ser = ser(\Gamma.current)$

$lang' = \begin{cases} lang, & \text{if } lang \text{ supports } q.\mathtt{logicRef} \\ \text{the unique language supporting } q.\mathtt{logicRef}, & \text{otherwise} \end{cases}$

$ser' = \begin{cases} ser, & \text{if } lang' \text{ supports } ser \\ \text{the default serialization for } lang', & \text{otherwise} \end{cases}$

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

10.2.2 Semantics of Networks

The semantics of networks of OMS is given with the help of a directed graph. Its nodes and edges are specified by the `NetworkElements`, which can be OMS, OMS mappings, or OMS networks. Intuitively, the graph of a network consists of the union of all graphs of the network elements it contains, where an OMS yields a graph with one isolated node. 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 $addImports(\Gamma, G, [iri_1, \dots, iri_n])$ adds to G all import edges in $\Gamma.imports$ between nodes which appear in the subgraph determined by $\Gamma(iri_1), \dots, \Gamma(iri_n)$.

10.2.2.1 Semantics of network definitions

$$sem(\Gamma, \mathtt{NetworkDefinition}) \quad = \Gamma' : LogicalEnvironment$$

If n is a `NetworkDefinition`,

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

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

If n.`ConservativityStrength` is `model-conservative`, the semantics is only defined if ~~$sem(\Gamma, n.\mathtt{network}) \neq \emptyset$~~the class of families of models compatible with the graph $sem(\Gamma, n.\mathtt{network})$ 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 models compatible with the graph $sem(\Gamma, n.\mathtt{network})$ consist of exactly one isomorphism class of families of models.

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

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

If n.`ConservativityStrength` is `not-model-conservative`, the semantics is only defined if ~~$sem(\Gamma, n.\mathtt{network}) = \emptyset$~~the class of families of models compatible with the graph $sem(\Gamma, n.\mathtt{network})$ 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.

10.2.2.2 Semantics of networks

$$sem(\Gamma, \mathtt{Network}) \quad = G : OMSGraph$$

If n is a network,

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

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

10.2.2.3 Semantics of sets of network elements

$$sem(\Gamma, Set(\mathtt{NetworkElement})) \quad = G : OMSGraph$$

If $elem_1, \dots, elem_n$ are all `NetworkElements`,

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

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

10.2.2.4 Semantics of network elements

$$\begin{array}{|ll|} \hline sem(\Gamma, G, \texttt{NetworkElement}) & = G' \\ & : OMSGraph \\ \hline \end{array}$$

If *networkElement* is a `NetworkElement`,

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

10.2.2.5 Semantics of sets of excluded elements

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

If $elem_1, \ldots, elem_n$ are all `ExcludedElements`,

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

where
$G_1 = sem(\Gamma, G, elem_1)$
$G_2 = sem(\Gamma, G_1, elem_2)$
...
$G' = sem(\Gamma, G_{n-1}, elem_n)$

10.2.2.6 Semantics of excluded elements

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

If *excludedElem* is a `ElementRef`,

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

.

If *excludedElem* is a `PathReference`,

$$sem(\Gamma, G, excludedElem) = removePaths(\Gamma, G, iri_1, iri_2)$$

where $iri_1 = excludedElem.\texttt{elementRef.iri}$ and $iri_2 = excludedElem.\texttt{elementRef2.iri}$).

10.2.3 Semantics of OMS

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

10.2.3.1 Semantics of basic OMS

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

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

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

where

- ~~$(\Sigma', \Delta') = semBasic_{(\Gamma.lang, \Gamma.logic, \Gamma.ser)}(\emptyset, basicOMS)$~~ $(\Sigma', \Delta') = semBasic_{(\Gamma.lang, \Gamma.logic, \Gamma.ser)}(O)$,
- $\mathcal{M}' = \{M \in \mathsf{Mod}(\Sigma') \mid M \models \Delta'\}$
- Γ' is obtained from Γ by adding to $\Gamma.imports$ a new node labeled with the name of ~~*basicOMS*~~O, $\Gamma.logic$,Σ', $\mathcal{M}'$ and Δ'.

10.2.3.2 Semantics of basic OMS in a local environment

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

For a `BasicOMS` ~~$basicOMS$~~ 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:

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

where

- ~~$(\Sigma', \Delta') = semBasic_{(\Gamma.lang, \Gamma.logic, \Gamma.ser)}(\Sigma, basicOMS)$~~ $(\Sigma', \Delta') = semBasic_{(\Gamma.lang, \Gamma.logic, \Gamma.ser)}(\Sigma, O)$
- $\mathcal{M}' = \{M \in \mathcal{M} \mid M \models \Delta'\}$
- Γ' is obtained from Γ by adding to $\Gamma.imports$ a new node labeled with the name of ~~$basicOMS$ and the other components as given by $sem_{(\Gamma.lang, \Gamma.logic, \Gamma.ser)}(basicOMS)$~~ O, $\Gamma.logic$, Σ', $\mathcal{M}'$ and Δ'.

10.2.3.3 Semantics of closable OMS

~~In the rest of this section, given a global environment Γ and an oms, the notation $Env(\Gamma, oms)$ is used for the global environment Γ' such that $sem(\Gamma, oms) = \Gamma'$.~~

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

The semantics of a `BasicOMS` has been defined above.
The semantics of an `OMSReference` ~~o~~ O is given by

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

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

- ~~$\mathbf{Inst}(o) = \mathbf{Inst}(\Gamma(o.\texttt{omsRef}))$~~ $postfixLogicIRI(o, l)$ is the string $o?logic = l$
- ~~$\mathsf{Sig}(o) = \mathsf{Sig}(\Gamma(o.\texttt{omsRef}))$~~ $\mathcal{I} = \mathbf{Inst}(\Gamma(locId))$,
- ~~$\mathsf{Mod}(o) = \mathsf{Mod}(\Gamma(o.\texttt{omsRef}))$~~ $\Sigma = \mathsf{Sig}(\Gamma(locId))$
- ~~$\mathsf{Th}(o) = \mathsf{Th}(\Gamma(o.\texttt{omsRef}))$~~ $\mathcal{M} = \mathsf{Mod}(\Gamma(locId))$
- ~~$Env(\Gamma, o)$~~ $\Delta = \mathsf{Th}(\Gamma(locId))$
- $Env(\Gamma, O)$ extends the graph of imports $\Gamma.imports$ with a new node for ~~o labeled~~ 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 ~~$o.\texttt{omsRef}$ to o, named $o.\texttt{importName}$~~ $locId$ to O, named $O.\texttt{importName}$ and labeled with the identity on ~~$\mathsf{Sig}(\Gamma(o.\texttt{omsRef}))$~~$\mathsf{Sig}(\Gamma(locId))$.

10.2.3.4 Semantics of closable OMS in a local environment

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

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

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

where

- ~~$\mathbf{Inst}(o) = \mathbf{Inst}(\Gamma(o.\texttt{omsRef}))$~~ $\mathcal{I}' = \mathbf{Inst}(\Gamma(locId))$
- ~~$\mathsf{Sig}(o) = \mathsf{Sig}(\Gamma(o.\texttt{omsRef})) \cup \Sigma$~~ $\Sigma' = \mathsf{Sig}(\Gamma(locId)) \cup \Sigma$
- ~~$\mathsf{Mod}(o) = \{M \in \mathsf{Mod}(\mathsf{Sig}(o)) \mid M|_\Sigma \in \mathcal{M} \text{ and } M|_{\mathsf{Sig}}(\Gamma(o.\texttt{omsRef})) \in \mathsf{Mod}(\Gamma(o.\texttt{omsRef}))\}$~~ $\mathcal{M}' = \{M \in \mathsf{Mod}(\Sigma') \mid M|_\Sigma \in \mathcal{M} \text{ and } M|_{\mathsf{Sig}(\Gamma(locId))} \in \mathsf{Mod}(\Gamma(locId))\}$
- ~~$\mathsf{Th}(o) = \iota_{\mathsf{Sig}(\Gamma(o.\texttt{omsRef})) \subseteq \mathsf{Sig}(o)}(\mathsf{Th}(\Gamma(o.\texttt{omsRef}))) \cup \iota_{\Sigma \subseteq \mathsf{Sig}(o)}(\Delta)$~~ $\Delta' = \iota_{\mathsf{Sig}(\Gamma(locId)) \subseteq \Sigma'}(\mathsf{Th}(\Gamma(locId))) \cup \iota_{\Sigma \subseteq \Sigma'}(\Delta)$
- ~~$Env(\Gamma, o)$~~ $Env(\Gamma, O)$ extends the graph of imports $\Gamma.imports$ with a new node for ~~o~~ O labeled as defined in the items above and with a new edge from the node labeled with ~~$o.\texttt{omsRef}$ to o, named $o.\texttt{importName}$~~ $locId$ to O, named $O.\texttt{importName}$ and labeled with the inclusion of Σ in ~~$\mathsf{Sig}(\Gamma(o.\texttt{omsRef}))$~~$\Sigma'$.

10.2.3.5 Semantics of ExtendingOMS

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

The semantics for `ClosableOMS` has been defined above.
If O is a `RelativeClosureOMS`, O.`closureType` = `minimize` and $O' = O$.`closableOMS`, then

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

where

- $\mathcal{I} = \mathbf{Inst}(O')$
- $\Sigma = \mathsf{Sig}(O')$
- $\mathcal{M} = \{M \in \mathsf{Mod}(O') \mid M \text{ is minimal in } \mathsf{Mod}(O')\}$ and "minimal" is interpreted in the pre-order defined by $M_1 \leq M_2$ if there is a model homomorphism $\mathcal{M}_1 \to \mathcal{M}_2$.
- $\Delta = \bot$
- Γ' is obtained from $\Gamma'' = Env(\Gamma, O')$ by adding to $\Gamma''.imports$ a new node labeled with $(Name(O), \mathbf{Inst}(O), \mathsf{Sig}(O), \mathsf{Mod}(O), \mathsf{Th}(O$ and an edge from the node of O' to the node of O labeled with the identity morphism on $\mathsf{Sig}(O')$.

The semantics of O is defined similarly for the other three alternatives of O.`closureType`, only the model class differs:

- if O.`closureType` = `maximize`, $\mathcal{M} = \{M \in \mathsf{Mod}(O') \mid M \text{ is maximal in } \mathsf{Mod}(O')\}$
- if O.`closureType` = `free`, $\mathcal{M} = \{M \in \mathsf{Mod}(O') \mid M \text{ is initial in } \mathsf{Mod}(O')\}$
- if O.`closureType` = `cofree`, $\mathcal{M} = \{M \in \mathsf{Mod}(O') \mid M \text{ is terminal in } \mathsf{Mod}(O')\}$

Here, initial and terminal models are defined as in category theory, see **??**.

10.2.3.6 Semantics of ExtendingOMS in a local environment

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

The semantics for `ClosableOMS` has been defined above.

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

Formally, if ~~O'~~ O is a `RelativeClosureOMS`, ~~O'.`closureType` = `minimize` and $O = O'$.`closableOMS`, then~~ O.`closureType` = `minimize` and $O' = O$.`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

- ~~$\mathbf{Inst}(O') = \mathbf{Inst}(O)$ $\mathsf{Sig}(O') = \mathsf{Sig}(O)$~~ $\mathcal{I}'' = \mathcal{I}'$
- ~~$\mathsf{Mod}(O') = \{M \in \mathsf{Mod}(O) \mid M \text{ is minimal in } \{M' \in \mathsf{Mod}(O) \mid M'|_\Sigma = M|_\Sigma\}\}$~~ $\Sigma'' = \Sigma'$
- ~~$\mathsf{Th}(O) = \bot$ where the semantics of the O is given relative to the environment Γ and the context $(\mathcal{I}, \Sigma, \mathcal{M}, \Delta)$,~~ $\mathcal{M}'' = \{M \in \mathcal{M} \mid M \text{ is minimal in } \{M' \in \mathcal{M} \mid M'|_\Sigma = M|_\Sigma\}\}$ and "minimal" is interpreted in the pre-order defined by ~~$\Sigma_1 \leq \Sigma_2$~~ $M_1 \leq M_2$ if there is a ~~signature morphism $\Sigma_1 \to \Sigma_2$.~~

  ~~The theory-level semantics for O' cannot be defined.~~

  ~~$Env(\Gamma, O')$~~ model homomorphism $\mathcal{M}_1 \to \mathcal{M}_2$
- $\Delta'' = \bot$
- Γ'' is obtained from ~~Γ~~ Γ' by adding to ~~$\Gamma.imports$~~ $\Gamma'.imports$ a new node labeled with ~~$(Name(O'), \mathbf{Inst}(O'), \mathsf{Sig}(O'), \mathsf{Mod}(O'), \mathsf{Th}(O$~~ $(Name(O), \mathbf{Inst}(O), \mathsf{Sig}(O), \mathsf{Mod}(O), \mathsf{Th}(O))$ and an edge from the node of ~~O~~ O' to the node of ~~O'~~ O labeled with the identity morphism on ~~$\mathsf{Sig}(O')$~~ Σ''.

The theory-level semantics for O cannot be defined.

The semantics of ~~O'~~ O is defined similarly for the other three alternatives of ~~O'.`closureType`~~ O.`closureType`, only the model class differs:

- if ~~O'.`closureType` = `maximize`, $\mathsf{Mod}(O') = \{M \in \mathsf{Mod}(O) \mid M \text{ is maximal in } \{M' \in \mathsf{Mod}(O) \mid M'|_\Sigma = M|_\Sigma\}\}$~~ O.`closureType` = `maximize`, $\mathcal{M}'' = \{M \in \mathcal{M} \mid M \text{ is maximal in } \{M' \in \mathcal{M} \mid M'|_\Sigma = M|_\Sigma\}\}$

- if ~~O'.closureType = free, $\mathsf{Mod}(O') = \{M \in \mathsf{Mod}(O) \mid M$ is initial in $\{M' \in \mathsf{Mod}(O) \mid M'|_\Sigma = M|_\Sigma\}\}$~~ O.closureType = fre
$\mathcal{M}'' = \{M \in \mathcal{M} \mid M$ is initial in $\{M' \in \mathcal{M} \mid M'|_\Sigma = M|_\Sigma\}\}$
- if ~~O'.closureType = cofree, $\mathsf{Mod}(O') = \{M \in \mathsf{Mod}(O) \mid M$ is terminal in $\{M' \in \mathsf{Mod}(O) \mid M'|_\Sigma = M|_\Sigma\}\}$~~ O.closureType =
$\mathcal{M}'' = \{M \in \mathcal{M} \mid M$ is terminal in $\{M' \in \mathcal{M} \mid M'|_\Sigma = M|_\Sigma\}\}$

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

10.2.3.7 Semantics of OMS

~~An is interpreted in a context similar to that~~

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

The semantics for a ClosableOMS ~~, the difference being that there is no local environment~~has been defined above.
The semantics for an ExtendingOMS has been defined above.
If o is a ClosureOMS,

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

where

$$(\Gamma', (I, \Sigma, \mathcal{M}, \Delta)) = sem(\Gamma, \cancel{o}O.\texttt{oms}), \qquad \Sigma_{\cancel{min}\,closure} = sem(\cancel{o}\Gamma', \Sigma, O.\texttt{closure.}\cancel{\Sigma}\texttt{circClosure}),$$

$$\Sigma_{var} = sem(\cancel{o}\Gamma', \Sigma, O.\texttt{closure.}\cancel{\Sigma}\texttt{circVars}), \qquad \Sigma_{fixed} = \Sigma \setminus (\Sigma_{\cancel{min}\,closure} \cup \Sigma_{var})$$

and

- if ~~o.closure.closureType = minimize~~O.closure.closureType = minimize, then
~~$\mathcal{M}' = \{M \in \mathcal{M} \mid M|_{\Sigma_{min} \cup \Sigma_{fixed}}$ is minimal in $\{M' \in \mathcal{M}|_{\Sigma_{min} \cup \Sigma_{fixed}} \mid M'|_{\Sigma_{fixed}} = M|_{\Sigma_{fixed}}\}\}$~~ $\mathcal{M}' = \{M \in \mathcal{M} \mid M|_{\Sigma_{closure} \cup \Sigma_{fixed}}$ is min
- if ~~o.closure.closureType = maximize~~O.closure.closureType = maximize, then
~~$\mathcal{M}' = \{M \in \mathcal{M} \mid M|_{\Sigma_{min} \cup \Sigma_{fixed}}$ is maximal in $\{M' \in \mathcal{M}|_{\Sigma_{min} \cup \Sigma_{fixed}} \mid M'|_{\Sigma_{fixed}} = M|_{\Sigma_{fixed}}\}\}$~~ $\mathcal{M}' = \{M \in \mathcal{M} \mid M|_{\Sigma_{closure} \cup \Sigma_{fixed}}$ is max
- if ~~o.closure.closureType = free~~O.closure.closureType = free, then
~~$\mathcal{M}' = \{M \in \mathcal{M} \mid M|_{\Sigma_{min} \cup \Sigma_{fixed}}$ is initial in $\{M' \in \mathcal{M}|_{\Sigma_{min} \cup \Sigma_{fixed}} \mid M'|_{\Sigma_{fixed}} = M|_{\Sigma_{fixed}}\}\}$~~ $\mathcal{M}' = \{M \in \mathcal{M} \mid M|_{\Sigma_{closure} \cup \Sigma_{fixed}}$ is initial
- if ~~o.closure.closureType = cofree~~O.closure.closureType = cofree, then
~~$\mathcal{M}' = \{M \in \mathcal{M} \mid M|_{\Sigma_{min} \cup \Sigma_{fixed}}$ is terminal in $\{M' \in \mathcal{M}|_{\Sigma_{min} \cup \Sigma_{fixed}} \mid M'|_{\Sigma_{fixed}} = M|_{\Sigma_{fixed}}\}\}$~~ $\mathcal{M}' = \{M \in \mathcal{M} \mid M|_{\Sigma_{closure} \cup \Sigma_{fixed}}$ is term

Γ'' is obtained from $\Gamma' = Env(\Gamma, O.\texttt{oms})$ by extending $\Gamma'.imports$ with a new node for O labeled as in the items above and with a new edge from the node of O.oms to the node of O labeled with the identity morphism on Σ.
The semantics of a TranslationOMS ~~O'~~ O is given by

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

where

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

The semantics of a ReductionOMS ~~O' is~~ O is

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

where

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

The semantics of an `ExtractionOMS` ~~O' is~~ O is

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

where

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

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

The semantics of an `ApproximationOMS` ~~O' is~~ O is

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

where

- ~~$\mathbf{Inst}(O) = \mathcal{I}$ when $(\Phi,\alpha,\beta) : \mathbf{Inst}(O'.\mathtt{oms}) \to \mathcal{I})$ is the default projection (in case $O'.\mathtt{approximation.logicRef}$ is missing, it is the identity on $\mathbf{Inst}(O'.\mathtt{oms})$)~~ $\mathcal{I}'' = \mathcal{I}'$,
- ~~$\mathsf{Sig}(O) = \Phi(\Sigma)$~~ $\Sigma' = \Phi(\Sigma)$
- ~~$\mathsf{Th}(O) = \alpha^{-1}_{\mathsf{Sig}(O'.\mathtt{oms})}(\mathsf{Th}(O'.\mathtt{oms})^\bullet) \cap \mathsf{Sen}^I(\mathsf{Sig}(O'.\mathtt{oms}))$~~[8] $\Delta = \alpha^{-1}_{\mathsf{Sig}(O.\mathtt{oms})}(\mathsf{Th}(O.\mathtt{oms})^\bullet) \cap \mathsf{Sen}^{\mathcal{I}'}(\mathsf{Sig}(O.\mathtt{oms}))$. , i.e. that part of ~~$\mathsf{Th}(O'.\mathtt{oms})$~~ $\mathsf{Th}(O.\mathtt{oms})$ 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.
- ~~$\mathsf{Mod}(O)$~~ $\mathcal{M}$ is the class of ~~$\mathsf{Th}(O)$~~ Δ-models
- ~~$Env(\Gamma, O')$~~ Γ'' is obtained from ~~$\Gamma'' = Env(\Gamma, O'.\mathtt{oms})$ by extending $\Gamma''.imports$~~ $\Gamma' = Env(\Gamma, O.\mathtt{oms})$ by extending $\Gamma'.imports$ with a new node for ~~O'~~ O labeled as in the items above and with a new edge from the node of ~~$O'.\mathtt{oms}$~~ $O.\mathtt{oms}$ to the node of ~~O' labeled with $((\Phi,\alpha,\beta),\iota : \Phi(\Sigma) \to \mathsf{Sig}(O'.\mathtt{oms}))$~~ O labeled with $(\rho, \iota : \Phi(\Sigma) \to \mathsf{Sig}(O.\mathtt{oms}))$

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

The semantics of a `FilteringOMS` O ~~, where $O' = O.\mathtt{filtering.basicOMS}$, is defined only if $\mathsf{Sig}(O') \subseteq \mathsf{Sig}(O.\mathtt{oms})$~~ is defined by case distinction. Let $(\Gamma', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)) = sem(\Gamma, O.\mathtt{oms})$ and ~~$\mathsf{Th}(O') \subseteq \mathsf{Th}(O.\mathtt{oms})$. Two cases are distinguished based on the value of $O.\mathtt{filtering.removalKind}$. If $O.\mathtt{filtering.removalKind} = keep$~~ $(c, \mathcal{I}', \Sigma', \Delta') = sem(\Gamma', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), O.\mathtt{fi}$

If $c = keep$, the semantics of O is given by

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

where

- ~~$\mathbf{Inst}(O) = \mathbf{Inst}(O')$~~ $\mathcal{I}'' = \mathcal{I}'$
- ~~$\mathsf{Sig}(O) = \Sigma'$ where Σ'~~ Σ'' is the smallest signature with ~~$\mathsf{Sig}(O') \subseteq \Sigma'$ and $\mathsf{Th}(O') \subseteq \mathsf{Sen}(\Sigma')$~~ $\Sigma' \subseteq \Sigma''$ and $\Delta' \subseteq \mathsf{Sen}(\Sigma'')$. (If this smallest signature does not exist, the semantics is undefined.)
- ~~$\mathsf{Th}(O) = (\mathsf{Th}(O.\mathtt{oms}) \cap \mathsf{Sen}(\mathsf{Sig}(O))) \cup \mathsf{Th}(O')$~~ $\Delta'' = (\Delta \cap \mathsf{Sen}(\Sigma'')) \cup \Delta'$
- $\mathsf{Mod}(O)$ is the class of all ~~$\mathsf{Th}(O)$~~ Δ-models.

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

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

If $\cancel{O.\mathtt{filtering.removalKind} = remove}c = remove$, the semantics of O is

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

where

- $\cancel{\mathbf{Inst}(O) = \mathbf{Inst}(O')}$ $cI'' = \mathcal{I}'$
- $\cancel{\mathsf{Sig}(O) = \mathsf{Sig}(O.\mathtt{oms}) \setminus \mathsf{Sig}(O')}$ $\Sigma'' = \Sigma \setminus \Sigma'$
- $\cancel{\mathsf{Th}(O) = \mathsf{Th}(O.\mathtt{oms}) \cap \mathsf{Sen}(\mathsf{Sig}(O)) \setminus \mathsf{Th}(O')}$ $\Delta'' = \Delta \cap \mathsf{Sen}(\Sigma'') \setminus \Delta'$
- $\cancel{\mathsf{Mod}(O)}$ $\mathcal{M}''$ is the class of all $\mathsf{Th}(O)$-models.
- $\cancel{Env(\Gamma, O')}$ Γ'' is obtained from ~~$\Gamma'' = Env(\Gamma, O.\mathtt{oms})$ by extending $\Gamma''.imports$~~ 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 $O.\mathtt{oms}$ labeled with the inclusion of ~~Σ' in $\mathsf{Sig}(O.\mathtt{oms})$~~ Σ'' in Σ.

The semantics of an `UnionOMS` O is

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

where

- ~~$\mathbf{Inst}(O) = I$ where $\mathbf{Inst}(O_1) \bigcup \mathbf{Inst}(O_2) = (I, (\Phi_1, \alpha_1, \beta_1) : \mathbf{Inst}(O_1) \to I, (\Phi_2, \alpha_2, \beta_2) : \mathbf{Inst}(O_2) \to I)$~~ $cI' = \mathcal{I}$ where $\mathbf{Inst}(O_1) \bigcup \mathbf{Inst}(O_2) = (\mathcal{I}, (\Phi_1, \alpha_1, \beta_1) : \mathbf{Inst}(O_1) \to \mathcal{I}, (\Phi_2, \alpha_2, \beta_2) : \mathbf{Inst}(O_2) \to \mathcal{I})$
- $\cancel{\mathsf{Sig}(O) = \Phi_1(\mathsf{Sig}(O_1)) \cup \Phi_2(\mathsf{Sig}(O_2))}$ $\Sigma' = \Phi_1(\mathsf{Sig}(O_1)) \cup \Phi_2(\mathsf{Sig}(O_2))$
- $\cancel{\mathsf{Mod}(O) = \{M \in \mathsf{Mod}(\mathsf{Sig}(O)) \mid \beta_{\Sigma_i}(M|_{\Phi_i(\mathsf{Sig}(O_i))}) \in \mathsf{Mod}(O_i), \text{ for } i = 1, 2\}}$ $\mathcal{M}' = \{M \in \mathsf{Mod}^{\mathcal{I}}(\Sigma') \mid \beta_{\Sigma_i}(M|_{\Phi_i(\mathsf{Sig}(O_i))}) \in \mathsf{Mod}(O_i$
- $\cancel{\mathsf{Th}(O) = \alpha_1(\mathsf{Th}(O_1)) \cup \alpha_2(\mathsf{Th}(O_2))}$ $\Delta' = \alpha_1(\mathsf{Th}(O_1)) \cup \alpha_2(\mathsf{Th}(O_2))$.
- $\cancel{Env(\Gamma, O')}$ Γ'' is obtained from ~~$\Gamma'' = Env(Env(\Gamma, O_1), O_2)$ by extending $\Gamma''.imports$~~ $\Gamma' = Env(Env(\Gamma, O_1), O_2)$ 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 $\cancel{(\Phi_i, \alpha_i, \beta_i, \iota_i : \Phi_i(O_i) \to \mathsf{Sig}(O))}(\Phi_i, \alpha_i, \beta_i, \iota_i : \Phi_i(O_i) \to \Sigma')$.

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

If $O.\mathtt{conservativityStrength}$ 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

- $\cancel{\mathbf{Inst}(O) = \mathbf{Inst}(O.\mathtt{oms}) = \mathbf{Inst}(O.\mathtt{extension})}$ $\mathcal{I}' = \mathbf{Inst}(O.\mathtt{oms}) = \mathbf{Inst}(O.\mathtt{extension})$ (which means that the institutions of $O.\mathtt{oms}$ and $O.\mathtt{extension}$ must be the same)
- $\cancel{\mathsf{Sig}(O) = \mathsf{Sig}(O.\mathtt{oms}) \cup \mathsf{Sig}((\mathbf{Inst}(O.\mathtt{oms}), \mathsf{Sig}(O.\mathtt{oms}), \mathsf{Mod}(O.\mathtt{oms}), \mathsf{Th}(O.\mathtt{oms})), O.\mathtt{extension})}$ $\Sigma' = \mathsf{Sig}(O.\mathtt{oms}) \cup \mathsf{Sig}(sem(\Gamma, ($
- $\cancel{\mathsf{Mod}(O) = \{M \in \mathsf{Mod}(\mathsf{Sig}(O)) \mid M|_{\mathsf{Sig}(O.\mathtt{oms})} \in \mathsf{Mod}(O.\mathtt{oms}) \text{ and } M|_{\mathsf{Sig}(O.\mathtt{extension})} \in \mathsf{Mod}(O.\mathtt{extension})\}}$ $\mathcal{M}' = \{M \in \mathsf{Mod}(\Sigma')$
- $\cancel{\mathsf{Th}(O) = \mathsf{Th}(O.\mathtt{oms}) \cup \mathsf{Th}(O.\mathtt{extension})}$ $\Delta' = \mathsf{Th}(O.\mathtt{oms}) \cup \mathsf{Th}(O.\mathtt{extension})$
- ~~$Env(\Gamma, O')$ 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 inclusion of $\mathsf{Sig}(O.\mathtt{oms})$ in $\mathsf{Sig}(O)$~~ Γ'' is $Env(Env(\Gamma, O.\mathtt{oms}), O.\mathtt{extension})$.

The semantics of a `QualifiedOMS` O in the context Γ is the same as the semantics of $O.\mathtt{oms}$ in the context Γ' given by the semantics of $O.\mathtt{qualification}$ in the context Γ. The change of context is local to $O.\mathtt{oms}$, 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 $(\Gamma'', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)) = sem(sem(\Gamma, O.\mathtt{qualification}), O.\mathtt{oms})$.

The semantics of a `CombinationOMS` O is

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

where

- ~~$\mathsf{Inst}(O) = I$~~ $\mathcal{I}' = \mathcal{I}$,
- ~~$\mathsf{Sig}(O) = \Sigma$, where $(I, \Sigma, \{\mu_i\}_{i\in|G|})$~~ $\Sigma' = \Sigma$, where $(\mathcal{I}, \Sigma, \{\mu_i\}_{i\in|G|})$ is the colimit of the graph ~~G given by the semantics of O.network~~ $G = sem(\Gamma, O.\texttt{network})$,
- ~~$\mathsf{Th}(O) = \bigcup_{i\in|G|}\mu_i(\mathsf{Th}(O_i))$~~ $\Delta' = \bigcup_{i\in|G|}\mu_i(\mathsf{Th}(O_i))$, where O_i is the OMS label of the node i in G
- ~~$\mathsf{Mod}(O) = \{M \in \mathsf{Mod}(\Sigma) \mid M|_{\mu_i} \in \mathsf{Mod}(O_i), i \in |G|\}$~~ $\mathcal{M}' = \{M \in \mathsf{Mod}(\Sigma) \mid M|_{\mu_i} \in \mathsf{Mod}(O_i), i \in |G|\}$, where O_i is the OMS label of the node i in G.
- ~~$Env(\Gamma, O)$~~ 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 $i \in |G|$.

10.2.3.8 Semantics of `CircClosure`

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

If c is a `CircClosure`,

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

10.2.3.9 Semantics of `CircVar`

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

If c is a `CircVar`,

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

10.2.3.10 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

- ~~$\mathsf{Inst}(O) = sem(O.\texttt{omsLanguageTranslation}) : \Gamma.logic \to logic'$~~ $\rho = sem(O.\texttt{omsLanguageTranslation}) : \Gamma.logic \to logic'$
- ~~$\mathsf{Mor}(O) = sem(\Gamma.\{current = (lang', logic', ser')\}, \Phi(\Sigma), O.\texttt{symbolMap})$~~ $\sigma = sem(\Gamma.\{current = (lang', logic', ser')\}, \Phi(\Sigma), O.\texttt{sym}$

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

10.2.3.11 Semantics of OMS language translations

~~The semantics of a O is $sem(\Gamma, O.\texttt{iri}) = (\Phi, \alpha, \beta)$,~~

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

If t is a `NamedLanguageTranslation`,

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

where ~~(Φ, α, β) is the institution comorphism named by $O.\texttt{iri}$ in~~ $\Gamma(O.\texttt{omsLanguageTranslationRef})$ 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 $t.$`languageRef` (if this is a logic) or to some logic supported by $t.$`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 `OMSLanguageTranslation`s, ~~$sem(\Gamma, Sequence\{t_1, \ldots, t_n\}) = (\Phi, \alpha, \beta)$, where $sem(\Gamma, t_i) = (\Phi_i, \alpha_i, \beta_i)$~~ $sem(\Gamma, Sequence\{t_1, \ldots, t_n\}) = \rho$, where $sem(\Gamma, t_i) = \rho_i$ for $i = 1, \ldots, n$ and ~~$(\Phi, \alpha, \beta) = (\Phi_1, \alpha_1, \beta_1); \ldots; (\Phi_n, \alpha_n, \beta_n)$~~ $(\Phi, \alpha, \beta) = \rho_1; \rho_2; \ldots$

10.2.3.12 Semantics of reductions

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

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

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

If $O.$`reduction.omsLanguageTranslation` is missing, it defaults to the identity morphism of the current logic of Γ.
The semantics of a reduction $O =$ with $O.$`reduction.removalKind` $= \mathit{keep}$ is

- ~~$\mathbf{Inst}(O)$~~ ρ is the identity morphism on the current logic of Γ
- ~~$\mathsf{Mor}(O)$~~ σ is the inclusion of $\mathit{sem}(\Gamma, \Sigma, O.\texttt{reduction.symbolList})$ in Σ.

10.2.3.13 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 `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.

10.2.3.14 Semantics of symbol maps

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

If ~~m~~ r is a `SymbolMap` ~~,~~

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

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

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

where ~~$\sigma = \mathit{makeMorphism}_{\mathit{logic}(\Gamma.\mathit{current})}((s_1, t_1), \ldots, (s_n, t_n))$and $(s, t) = \mathit{sem}(\Gamma, \Sigma_1, \Sigma_2, m.\texttt{generalSymbolMapItem}), \mathit{Set}\{(s_1, t_1), \ldots$~~ must be the unique signature morphism with the properties

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

~~Applications implicitly map those non-logical symbols of the source OMS, for which an explicit mapping is not given, to non-logical symbols of the same (local) name in the target OMS, wherever this is uniquely defined – in detail:~~ If σ does not exist or is not uniquely determined, the semantics of r is undefined.
~~O_s, O_t are OMS $M \subseteq |\mathsf{Sig}(O_s)| \times |\mathsf{Sig}(O_t)|$ maps non-logical symbols (elements of the signature) of O_s to non-logical symbols of O_t $n_s \leftarrow \mathrm{localname}(e_s)$ $N_t \leftarrow \{\mathrm{localname}(e) \mid e \in |\Sigma(O_t)|\}$ $M \leftarrow M \cup \{(e_s, e_t)\}$ M completely covers $|\Sigma(O_s)|$~~

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

~~The local name of a non-logical symbol is determined as follows~~[8] ~~is a non-logical symbol (identified by an IRI; clause ??) f $n \leftarrow$ the longest suffix of e that matches the production of XML n~~ If r is a `SymbolMap` such that for each `SymbolMapItem` $gItem$ in r.`generalSymbolMapItem` we have that
$gItem$.`source` $\in |\Sigma|$ and $gItem$.`target` $\in |\Sigma'|$
and for each `Symbol` s in r.`generalSymbolMapItem` we have that s is an element both of $|\Sigma|$ and $|\Sigma'|$,

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

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.

10.2.3.15 Semantics of extractions

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

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~~(see [?] for the definition in a description logic context and [?] for a generalization to an arbitrary institution)~~, i.e. the smallest sub-theory $\langle \Sigma', \Delta' \rangle$ of (Σ, Δ) such that the following model-theoretic inseparability holds

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

~~(In [?] , it is shown that the smallest depleting Σ''-module exists in description logics, and in [?] this is generalized to arbitrary institutions.)~~

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

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

[?] 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. [?] generalizes both the definition of smallest depleting Σ''-module and the mentioned result to arbitrary institutions.

10.2.3.16 Semantics of approximations

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

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 $sem(a.\texttt{logicRef}) = \mathcal{I}$~~, 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.

10.2.3.17 Semantics of filtering

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

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

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

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

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

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

$$sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), \texttt{BasicOMSOrSymbolList}) \quad = (\mathcal{I}', \Sigma', \mathcal{M}', \Delta') \\ : (Institution, Signature, Sentences)$$

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

10.2.3.18 Semantics of extension

$$sem(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta), \texttt{Extension}) \quad = (\Gamma', (\mathcal{I}, \Sigma', \mathcal{M}', \Delta')) \\ : (LogicalEnvironment, (Institution, Signature, ModelClass, Sentences))$$

If e is an Extension,

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

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

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

If $e.$extensionName is present, the inclusion link is labeled with this name.

10.2.3.19 Semantics of interface signatures

$$sem(\Gamma, \Sigma, \texttt{RemovalKind}, \texttt{InterfaceSignature}) \quad = \Sigma' \\ : Signature$$

If r is a RemovalKind and s is an InterfaceSignature, ~~$sem(\Gamma, \Sigma,$ Qual-Symbol+$) = \Sigma'$,~~

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

where

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

10.2.3.20 Semantics of OMS definitions

$$sem(\Gamma, \texttt{OMSDefinition}) \quad = \Gamma' \\ : LogicalEnvironment$$

An OMSDefinition O extends the global environment~~:~~

$$sem(\Gamma, O) = \Gamma[O.\text{omsName} \mapsto sem(\Gamma, O.\text{oms})]$$

.

~~$\Gamma.logic$ and **Inst**(O.oms) must be the same~~

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

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

The conservativity strength annotations refer to the semantics of O.oms in the current logic of Γ. Therefore, let $(\Gamma_0, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)) = sem(\Gamma, O.\text{oms})$.
If O.conservativityStrength is model-conservative, the semantics is only defined if ~~$sem(\Gamma, O.\text{oms}) \neq \emptyset$~~.$\mathcal{M} \neq \emptyset$.
If O.conservativityStrength is consequence-conservative, the semantics is only defined if ~~$sem(\Gamma, O.\text{oms})$~~ Δ has

only tautologies[8] as signature-free[9] logical consequences.
If O.`conservativityStrength` is `monomorphic`, the semantics is only defined if ~~$sem(\Gamma, O.\texttt{oms})$~~ $\mathcal{M}$ consist of exactly one isomorphism class of models.
If O.`conservativityStrength` is `weak-definitional`, the semantics is only defined if ~~$sem(\Gamma, O.\texttt{oms})$~~ $\mathcal{M}$ is empty or a singleton.
If O.`conservativityStrength` is `definitional`, the semantics is only defined if ~~$sem(\Gamma, O.\texttt{oms})$~~ $\mathcal{M}$ is a singleton.

10.2.3.21 Semantics of OMS references

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

~~If O is an~~
The rule for `OMSReference`~~,~~

$$\underline{sem(\Gamma, O) = \Gamma(O.\texttt{omsRef})}$$

s has been given above, as `OMSReference`s are a particular case of `ClosableOMS`.

10.2.3.22 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.`iri` from $|\Sigma|$. If such symbol does not exist, the semantics is undefined.

10.2.3.23 Semantics of symbol map items

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

If *smi* is a `SymbolMapItem`,

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

where ~~$sem(\Gamma, \Sigma_1, smi.\texttt{symbol1}) = s_1$ and $sem(\Gamma, \Sigma_2, smi.\texttt{symbol2}) = s_2$~~ $sem(\Gamma, \Sigma_1, smi.\texttt{source}) = s_1$ and $sem(\Gamma, \Sigma_2, smi.\texttt{target}) =$

10.2.3.24 Semantics of general symbol map items

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

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 $sem(\Gamma, \Sigma_1, gsmi) = s$ ~~.~~
~~If *sen* is a ,~~

$$\underline{sem(\Gamma, \Sigma, sen) = \varphi}$$

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

10.2.3.25 Semantics of references

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

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

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

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

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

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

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

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

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

~~If t is a , $sem(\Gamma, t) = \rho$ where ρ is the institution comorphism from the heterogeneous logical environment named by t.`omsLanguageTranslationRef`. This is defined only if the domain of ρ is the current logic of Γ.~~

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

10.2.4 Semantics of OMS Mappings

10.2.4.1 Semantics of mapping definitions

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

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

10.2.4.2 Semantics of interpretation definitions

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

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.`OMSLanguageTranslation` is missing, the default translations between the logics is selected.
- ~~$sem(\Gamma.\{current = (lang, logic', ser)\}, \Phi(\mathsf{Sig}(L_1)), \mathsf{Sig}(L_2), d.\texttt{symbolMap}) = \sigma$,~~ $\sigma = sem(\Gamma.\{current = (lang, logic', ser)\}, \Phi(\mathsf{Sig}(L$
  where $\Gamma.current = (lang, logic, ser)$ and $logic'$ is the target logic of ρ, or, if d.`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 $\beta_{\mathsf{Sig}(L_1)}(M_2|_\sigma) \in \mathsf{Mod}(L_1)$ for each $M_2 \in \mathsf{Mod}(L_2)$.
If the optional argument d.`conservativityStrength` is

- `model-conservative`, for each model ~~$M_1 \in \mathsf{Mod}(L_1)$~~ $M_1 \in |\mathsf{Mod}(L_1)|$ there must exist a model ~~$M_2 \in \mathsf{Mod}(L_2)$~~ $M_2 \in |\mathsf{Mod}(L_2)|$ 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 model ~~$M_1 \in \mathsf{Mod}(L_1)$~~ $M_1 \in |\mathsf{Mod}(L_1)|$ such that there is no model ~~$M_2 \in \mathsf{Mod}(L_2)$~~ $M_2 \in |\mathsf{Mod}(L_2)|$ 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 $\mathcal{M}_1 \not\models \varphi$.

10.2.4.3 Semantics of refinement definitions

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

If d is a `RefinementDefinition`,

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

where ~~$\Gamma' = \Gamma[d.\texttt{interpretationName} \mapsto (G, \sigma, N_1, N_2)]$ and $sem(\Gamma, d.\texttt{refinement}) = (G, \sigma, N_1, N_2)$~~ $\Gamma' = \Gamma[d.\texttt{interpretationNam}$

10.2.4.4 Semantics of interpretation types

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

If t is an `InterpretationType`,

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

where

- ~~$\mathbf{Name}(L_1) = \mathbf{Name}(t.\texttt{oms})$ and $\mathbf{Name}(L_2) = \mathbf{Name}(t.\texttt{oms2})$~~ $\mathbf{Name}(L_1) = \mathbf{Name}(t.\texttt{source})$ and $\mathbf{Name}(L_2) = \mathbf{Name}(t.\texttt{targ}$
- ~~$(\mathbf{Inst}(L_1), \mathsf{Sig}(L_1), \mathsf{Mod}(L_1), \mathsf{Th}(L_1)) = sem(\Gamma, t.\texttt{oms})$~~ $(\mathbf{Inst}(L_1), \mathsf{Sig}(L_1), \mathsf{Mod}(L_1), \mathsf{Th}(L_1)) = sem(\Gamma, t.\texttt{source})$,
- ~~$(\mathbf{Inst}(L_2), \mathsf{Sig}(L_2), \mathsf{Mod}(L_2), \mathsf{Th}(L_2)) = sem(\Gamma, t.\texttt{oms2})$~~ $(\mathbf{Inst}(L_2), \mathsf{Sig}(L_2), \mathsf{Mod}(L_2), \mathsf{Th}(L_2)) = sem(\Gamma, t.\texttt{target})$,

10.2.4.5 Semantics of refinements

$$sem(\Gamma, \texttt{Refinement}) = (((G_1, G_2), \sigma, \mathcal{M}) : (OMSGraph, OMSGraph, GraphMorphism, ModelClass)$$

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 ~~functor~~ graph homomorphism $\sigma^G : Shape(G_1) \to Shape(G_2)$, ~~together with~~ where given a network G, its shape $Shape(G)$ 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 model is a class $\mathcal{M}$ of pairs of families of models compatible with the two networks. Given a network morphism $\sigma : G_1 \to G_2$ and a G_2 model F, $F|_\sigma$ is defined as the family of models $\{M_i\}_{i \in Nodes(G_1)}$ such that $M_i = F_{\sigma^G(i)}|_{\sigma^M_i}$ for each $i \in Nodes(G_1)$.

Thus, the semantics of a `Refinement` consists of

- a refinement signature (G_1, G_2),
- a network morphism σ and
- a refinement model $\mathcal{M}$.

If r is `RefinementOMS`,

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

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 ~~L~~ N are given by $sem(\Gamma, r.\texttt{oms})$,
- σ is the identity morphism on $\mathsf{Sig}(r.\texttt{oms})$,
- $\mathcal{M} = \{((M), (M)) \mid M \in \mathsf{Mod}(r.\texttt{oms})\}$, where (M) is the singleton family consisting of M.

If r is `RefinementNetwork`,

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

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

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

~~where $sem(\Gamma, r.\texttt{refinement}) = ((G_1, G_1'), \sigma_1, \mathcal{M}_1)$, $sem(\Gamma, r.\texttt{refinement2}) = ((G_2, G_2'), \sigma_2, \mathcal{M}_2)$ such that $G_1' = G_2$, $\sigma = \sigma_1; \sigma_2$ is a network morphism from G_1 to G_2', and $\mathcal{M} = \{(F_1, F_3) \mid \exists F_2$ such that $(F_1, F_2) \in \mathcal{M}_1$ and $(F_2, F_3) \in \mathcal{M}_2\}$~~

If r is `SimpleOMSRefinement`,

$$sem(\Gamma, r) = ((G_1, G'_2), \sigma', \mathcal{M})$$

where
~~$sem^M(\Gamma, r.\texttt{oms}) = (\mathcal{I}_1, \Sigma_1, \mathcal{M}_1, \Delta_1)$~~$sem(\Gamma, r.\texttt{refinement}) = ((G_1, G_1'), \sigma_1, \mathcal{M}_1)$,
~~$sem(\Gamma, r.\texttt{refinement}) = ((G_1, G_2), (\rho_2, \sigma_2), \mathcal{M}')$ such that G_1 consists of an isolated node labeled with $(\mathcal{I}_2, \Sigma_2, \mathcal{M}_2, \Delta_2)$~~$sem(\Gamma, r.\texttt{refin}$
~~$sem(\Gamma, (\mathcal{I}_1, \Sigma_1), (\mathcal{I}_2, \Sigma_2), r.\texttt{omsRefinementMap}) = (\rho_1 = (\Phi, \alpha, \beta) : \mathcal{I}_1 \to \mathcal{I}_2, \sigma_1 : \Phi(\Sigma_1) \to \Sigma_2)$, for each $(M_1, M_2) \in \mathcal{M}', \beta_{\Sigma_1}(M_1|_{\sigma_1}) \in$~~
and G_2 are both graphs with one isolated node, labeled $(name1, \mathcal{I}_1, \Sigma_1, \mathcal{M}_1, \Delta_1)$ and respectively $(name2, \mathcal{I}_2, \Sigma_2, \mathcal{M}_2, \Delta_2)$,
$sem(\Gamma, \Sigma_1, \Sigma_2, r.\texttt{omsRefinementMap}) = (\rho = (\Phi, \alpha, \beta) : \mathcal{I}_1 \to \mathcal{I}_2, \sigma : \Phi(\Sigma_1) \to \Sigma_2)$,
~~G consists of an isolated node labeled with $sem^M(\Gamma, r.\texttt{oms})$~~$\sigma'$ maps the node n of G_1 to the node of G_3 and $(\sigma')^M_n = (\sigma_1)^M_n; (\rho; \sigma); (\sigma_2)^M_{\sigma^G_1(n)}$
~~$\sigma = (\rho_1, \sigma_1); (\rho_2, \sigma_2)$ and $\mathcal{M} = \{(\beta_{\Sigma_1}(M_1|_{\sigma_1}), M_2) \mid (M_1, M_2) \in \mathcal{M}'\}$~~and $\mathcal{M} = \{(M_1, M_3) \mid \exists M_2 such that (M_2, M_3) \in \mathcal{M}_2 and (M_1, M_2|_{(\rho,}$
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
~~$sem^M(\Gamma, r.\texttt{network}) = G_1, sem(\Gamma, r.\texttt{refinement}) = ((G_1', G_2), \sigma_2, \mathcal{M}')$~~$sem(\Gamma, r.\texttt{refinement}) = ((G_1, G_1'), \sigma_1, \mathcal{M}_1)$,
~~$sem(\Gamma, G_1, G_2, r.\texttt{networkRefinementMap}) = \sigma_1 : G_1 \to G_1'$~~$sem(\Gamma, r.\texttt{refinement2}) = ((G_2, G_2'), \sigma_2, \mathcal{M}_2)$,
~~$\sigma = \sigma_1; \sigma_2$ is a network morphism and $\mathcal{M} = \{(F_2|_\sigma, F_2) \mid (F_1, F_2) \in \mathcal{M}'\}$.~~

~~If m is an~~ $sem(\Gamma, G_1', G_2, r.\texttt{networkRefinementMap}) = \sigma : G_1' \to G_2$,

$$sem(\Gamma, (I_1, \Sigma_1), (I_2, \Sigma_2), m) = ((\Phi, \alpha, \beta), \sigma)$$

~~where~~
~~$sem(\Gamma, m.\texttt{omsLanguageTranslation}) = (\Phi, \alpha, \beta) : \mathcal{I}_1' \to \mathcal{I}_2'$ such that $\mathcal{I}_1' = \mathcal{I}_1$ and $\mathcal{I}_2' = \mathcal{I}_2$~~ $\sigma' = \sigma_1; \sigma; \sigma_2$
and ~~$sem(\Gamma.current = (lang', logic', ser'), \Phi(\Sigma_1), \Sigma_2, m.\texttt{symbolMap}) = \sigma : \Phi(\Sigma_1) \to \Sigma_2$ where $\Gamma.current = (lang, logic, ser)$, $logic'$ is the target logic of (Φ, α, β), and $lang'$ and ser' are the default language and serializations for $logic'$~~ $\mathcal{M} = \{(F_1, F_3) \mid \exists F_2$ such tha
The refinement is correct only if for each $(F_2, F_3) \in \mathcal{M}_2$, there is a $(F_1, F_2|_\sigma) \in \mathcal{M}_1$.

~~If m is a ,~~

$$sem(\Gamma, G_1, G_2, m) = sem(\Gamma, G_1, G_2, m.nodeMap)$$

10.2.4.6 Semantics of a set of refinements

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

If ~~$m_1, \ldots, m_n$ are all s,~~ $r_1, \ldots, r_n$ are all `Refinements`,

$$sem(\Gamma, G_1, G_2, Set\{m(r_1, \ldots, mr_n\})) = \sigma:$$

where
~~$sem(\Gamma, G_1, G_2, m_1) = (name_1^1, name_2^1, \rho_1, \sigma_1), \ldots$~~ $sem(\Gamma, r_1) = ((G_1^1, G_2^1), \sigma_1, \mathcal{M}_1), \ldots,$
~~$sem(\Gamma, G_1, G_2, m_n) = (name_1^n, name_2^n, \rho_n, \sigma_n)$ and~~ $sem(\Gamma, r_n) = ((G_1^n, G_2^n), \sigma_n, \mathcal{M}_n)$
~~$\sigma^G(name_1^i) = name_2^i$ and $\sigma^M_{name_1^i} = (\rho_i, \sigma_i)$ for each $i = 1, \ldots, n$. The map is required to~~ 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.

10.2.4.7 Semantics of refinement maps

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

If m is ~~a ,~~ an `OMSRefinementMap`,

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

where
~~$(\mathcal{I}_1, \Sigma_1, \mathcal{M}_1)$ is the label of $m.\texttt{omsName}$ in~~ G_1 ~~,~~must be a graph with just one isolated node labeled $(name_1, \mathcal{I}_1, \Sigma_1, \mathcal{M}_1, \Delta_1)$

~~$(\mathcal{I}_2, \Sigma_2, \mathcal{M}_2)$ is the label of $m.\texttt{omsName2}$ in~~ G_2 must be a graph with just one isolated node labeled $(name_2, \mathcal{I}_2, \Sigma_2, \mathcal{M}_2, \Delta_2)$,
~~$sem(\Gamma, m.\texttt{omsLanguageTranslation}) = \rho = (\Phi, \alpha, \beta) : \mathcal{I}_1 \to \mathcal{I}_2$,~~ $sem(\Gamma, m.\texttt{translation}) = \rho = (\Phi, \alpha, \beta) : \mathcal{I}_1 \to \mathcal{I}_2$, or

if m.translation is missing, the default comorphism between $\mathcal{I}_1$ and $\mathcal{I}_2$,
~~$sem(\Gamma.current = (lang', logic', ser'), \Phi(\Sigma_1), \Sigma_2, m.\texttt{symbolMap}) = \sigma : \Phi(\Sigma_1) \to \Sigma_2$,~~ $sem(\Gamma', \Phi(\Sigma_1), \Sigma_2, m.\texttt{symbolMap}) = \sigma : \Phi(\Sigma_1) \to$
where $\Gamma.current = (lang, logic, ser)$, $logic'$ is the target logic of (Φ, α, β)~~and~~, $lang'$ and ser' are the default language and serialization for $logic'$ 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 $sem(\Gamma, G_1, G_2, m.refinements) = \sigma$.

10.2.4.8 Semantics of entailment definitions

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

If e is an EntailmentDefinition,

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

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

10.2.4.9 Semantics of entailment types

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

If t is an OMSOMSEntailment,

$$sem(\Gamma, t) = L_2 \overset{\text{id}}{\to} L_1$$

where ~~$\mathbf{Name}(L_1) = \mathbf{Name}(t.\texttt{oms}), \mathbf{Name}(L_2) = \mathbf{Name}(t.\texttt{oms2})$~~$\mathbf{Name}(L_1) = \mathbf{Name}(t.\texttt{premise}), \mathbf{Name}(L_2) = \mathbf{Name}(t.\texttt{conclusi}$
~~$(\mathbf{Inst}(L_1), \mathsf{Sig}(L_1), \mathsf{Mod}(L_1), \mathsf{Th}(L_1)) = sem(\Gamma, t.\texttt{oms}), (\mathbf{Inst}(L_2), \mathsf{Sig}(L_2), \mathsf{Mod}(L_2), \mathsf{Th}(L_2)) = sem(\Gamma, t.\texttt{oms2})$~~ $(\mathbf{Inst}(L_1), \mathsf{Sig}(L_1), \mathsf{Mod}$
$(\mathbf{Inst}(L_2), \mathsf{Sig}(L_2), \mathsf{Mod}(L_2), \mathsf{Th}(L_2)) = sem(\Gamma, t.\texttt{conclusion})$ such that $\mathsf{Sig}(L_1) = \mathsf{Sig}(L_2)$ and $\mathsf{Mod}(L_1) \subseteq \mathsf{Mod}(L_2)$ and id is the identity morphism on $\mathsf{Sig}(L_1)$.

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

If t is a NetworkNetworkEntailment,

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

where ~~$sem(\Gamma, t.\texttt{network}) = G_1, sem(\Gamma, t.\texttt{network2}) = G_2$~~$sem(\Gamma, t.\texttt{premise}) = G_1, sem(\Gamma, t.\texttt{conclusion}) = G_2$, 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 models obtained by projecting each family of models compatible with G_1 to the component i is included in the class of models obtained by projecting each family of models compatible with G_2 to the component i. Then G extends the union of G_1 and G_2 for each pair of nodes (i_1, i_2), where i_1 and i_2 identify the 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 $\mathsf{Sig}(i_1)$.

10.2.4.10 Semantics of equivalence definitions

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

If d is an EquivalenceDefinition,

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

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

10.2.4.11 Semantics of OMS equivalences

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

If t is an `OMSEquivalence`,

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

where $O_1 = t.\texttt{oms}$, $O_2 = t.\texttt{oms2}$, $\cancel{O_3 = t.\texttt{oms3}, sem^{(\mathsf{Sig}(O_1)\cup\mathsf{Sig}(O_2),\emptyset)}_{\Gamma.lang,\Gamma.logic,\Gamma.ser}(O_3) = (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)}$ $O_3 = t.\texttt{mediatingOMS}$,
$sem(\Gamma, (\mathcal{I}, \mathsf{Sig}(O_1) \cup \mathsf{Sig}(O_2), \mathsf{Mod}^{\mathcal{I}}(\mathsf{Sig}(O_1) \cup \mathsf{Sig}(O_2)), \emptyset), O_3) = (\Gamma', (\mathcal{I}, \Sigma, \mathcal{M}, \Delta))$
G is the graph $N_1 \xrightarrow{\iota_1} N_3 \xleftarrow{\iota_2} N_3$ where

1. N_1 is labeled with $(\mathbf{Name}(O_1), \mathbf{Inst}(O_1), \mathsf{Sig}(O_1), \mathsf{Mod}(O_1), \mathsf{Th}(O_1))$,
2. N_2 is labeled with $(\mathbf{Name}(O_2), \mathbf{Inst}(O_2), \mathsf{Sig}(O_2), \mathsf{Mod}(O_2), \mathsf{Th}(O_2))$ 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. $\cancel{\mathbf{Inst}(O_1) = \mathbf{Inst}(O_2) = \mathbf{Inst}(O_3)}$ $\mathcal{I} = \mathbf{Inst}(O_1) = \mathbf{Inst}(O_2)$ and
3. for each $i = 1, 2$ and each model $M_i \in \mathsf{Mod}(O_i)$ there exists a unique model $M \in \mathcal{M}$ such that $M|_{\mathsf{Sig}(O_i)} = M_i$.

10.2.4.12 Semantics of network equivalences

$$\begin{aligned} sem(\Gamma, \texttt{NetworkEquivalence}) &= (G_1, G_2, G_3) \\ &: (OMSGraph, OMSGraph, OMSGraph) \end{aligned}$$

If t is a `NetworkEquivalence`,

$$sem(\Gamma, t) = (G_1, G_2, G_3)$$

where $n_1 = t.\texttt{network}$, $n_2 = t.\texttt{network2}$, $\cancel{n_3 = t.\texttt{network3}}$ $n_3 = t.\texttt{mediatingNetwork}$, $sem(\Gamma, n_1) = G_1$, $sem(\Gamma, n_2) = G_2$, $sem(\Gamma, n_3) = G_3$ such that G_1 and G_2 are subgraphs of G_3 and for each $i = 1, 2$ and each family of models $\mathcal{M}_i$ compatible with G_i there is a unique family of models $\mathcal{M}$ compatible with G_3 such that the projection of $\mathcal{M}$ to the nodes in G_i is $\mathcal{M}_i$.

10.2.4.13 Semantics of conservative extension definitions

$$\begin{aligned} sem(\Gamma, \texttt{ConservativeExtensionDefinition}) &= \Gamma' \\ &: LogicalEnvironment \end{aligned}$$

If d is a `ConservativeExtensionDefinition`,

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

where $\cancel{O_1 = d.\texttt{oms}, O_2 = d.\texttt{oms2}}$ $O_1 = d.\texttt{moduleType.module}$, $O_2 = d.\texttt{moduleType.whole}$, $c = d.\texttt{conservativityType}$, $\Sigma = sem(\Gamma, d.\texttt{interfaceSignature})$, $\Gamma' = \Gamma[d.\texttt{moduleName} \mapsto (G, \iota, N_2, N_1)]$ and G is the graph $N_1 \xrightarrow{\iota} N_2$ where N_1 is labeled with $(O_1, \mathbf{Inst}(O_1), \mathsf{Sig}(O_1), \mathsf{Mod}(O_1), \mathsf{Th}(O_1))$, N_2 with $(O_2, \mathbf{Inst}(O_2), \mathsf{Sig}(O_2), \mathsf{Mod}(O_2), \mathsf{Th}(O_2))$, and ι is an inclusion, when $\Sigma \subseteq \mathsf{Sig}(O_2) \subseteq \mathsf{Sig}(O_1)$ and if $\cancel{c = \textbf{mcons}}$ $c = \texttt{model-conservative}$ and for each $M \in \mathsf{Mod}(O_2)$ there is a model $M' \in \mathsf{Mod}(O_1)$ such that $M'|_\Sigma = M|_\Sigma$, or if $\cancel{c = \textbf{ccons}}$ $c = \texttt{consequence-conservative}$ and for each $\varphi \in \mathsf{Sen}(\Sigma)$, $O_1 \models \varphi$ implies $O_2 \models \varphi$.

10.2.4.14 Semantics of alignment definitions

$$\begin{aligned} sem(\Gamma, \texttt{AlignmentDefinition}) &= \Gamma' \\ &: LogicalEnvironment \end{aligned}$$

If d is an `AlignmentDefinition`,

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

where $\cancel{sem(\Gamma, d.\texttt{alignmentType}) = (L_1, L_2) \text{ and } \Gamma' = \Gamma[d.\texttt{AlignmentName} \mapsto (G, id, L_1, L_2)]}$ $sem(\Gamma, d.\texttt{alignmentType}) = (\Gamma_0, L_1, L$
and $\Gamma' = \Gamma_0[d.\texttt{AlignmentName} \mapsto (G, L'_1, L'_2)]$,
where $(L'_1, L'_2) = sem(\Gamma, L_1, L_2, d.\texttt{alignmentSemantics})$ ~~and~~,
$\cancel{G = sem(\Gamma, L'_1, L'_2, d.\texttt{alignmentCardinalityPair}, d.\texttt{alignmentSemantics}, d.\texttt{correspondence})}$ $card = d.\texttt{alignmentCardinal}$
or, when this is missing, $card = ('1', '1')$,
$aSem = d.\texttt{alignmentSemantics}$ or, when this is missing, $aSem = \texttt{single-domain}$,
and $G = sem(\Gamma_0, L'_1, L'_2, card, aSem, d.\texttt{correspondence})$.

10.2.4.15 Semantics of alignment types

$$\begin{array}{ll} sem(\Gamma, \texttt{AlignmentType}) & = (\Gamma', L_1, L_2) \\ & : (LogicalEnvironment, NodeLabel, NodeLabel) \end{array}$$

If t is an `AlignmentType`

$$sem(\Gamma, t) = (\Gamma'', L_1, L_2)$$

where $sem(\Gamma, t.\texttt{source}) = (\Gamma', (\mathcal{I}_1, \Sigma_1, \mathcal{M}_1, \Delta_1))$, $sem(\Gamma', t.\texttt{target}) = (\Gamma'', (\mathcal{I}_2, \Sigma_2, \mathcal{M}_2, \Delta_2))$, L_1 ~~is a node label whose name is **Name**(t.oms) and whose other components are given by $sem(\Gamma, t.\texttt{oms})$ and similarly,~~ and L_2 ~~is a node label whose name is **Name**(t.oms2) and whose other components are given by $sem(\Gamma, t.\texttt{oms2})$~~are the labels of the nodes of $t.\texttt{source}$ and $t.\texttt{target}$ in $\Gamma'.imports$.

10.2.4.16 Semantics of alignments

$$\begin{array}{ll} sem(\Gamma, L_1, L_2, (\texttt{AlignmentCardinality}, \texttt{AlignmentCardinality}), \texttt{AlignmentSemantics}, Set(\texttt{Correspondence})) & = G \\ & : OMSC \end{array}$$

If ~~$card$ is a set of s, sem~~ $card_1, card_2$ are `AlignmentCardinality`, $aSem$ is an `AlignmentSemantics` and $C = Set\{c_1, \ldots, c_n\}$ a set of `Correspondences`,

$$sem(\Gamma, L_1, L_2, (card_1, card_2), \cancel{sem}aSem, C) = G$$

where $sem(\Gamma, \mathsf{Sig}(L_1), \mathsf{Sig}(L_2), aSem, C) = (\Sigma_s, \Sigma_t, (\Sigma, \Delta), \phi_s : \Sigma_s \to \Sigma, \phi_t : \Sigma_t \to \Sigma, smap, cvalues)$,
~~if~~ where the semantics of the alignment is not defined in the following cases:

- if $cvalues = True$ and then at least one of the correspondences ~~$c_1, \ldots, c_n$~~ 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 ~~the semantics of the alignment is not defined, and the alignment is ill-formed if the alignment mapping does not have the arities given by $card$, otherwise~~ $smap$ 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

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

where ~~L_B, L_s and L_t are built in a logic-specific way from the correspondences $C_1, \ldots, C_n$ taking into account sem. [?] illustrates how this construction works in the case of OWL, in a way that can be generalized to other logics~~$L_s = (alignName ++ "\_so$
$L_t = (alignName ++ "\_target", \mathbf{Inst}(L_2), \Sigma_t, \mathsf{Mod}(\Sigma_t), \emptyset)$ and $L_B = (alignName ++ "\_bridge", \mathcal{I}, \Sigma, \mathsf{Mod}((\Sigma, \Delta)), \Delta)$.

10.2.4.17 Semantics of sets of correspondences

$$\begin{array}{ll} sem(\Gamma, \Sigma_1, \Sigma_2, \texttt{AlignmentSemantics}, Set(\texttt{Correspondence})) & = (\Sigma_s, \Sigma_t, (\mathcal{I}, \Sigma, \Delta), \phi_s : \Sigma_s \to \Sigma, \phi_t : \Sigma_t \to \Sigma, smap, cvalues) \\ & : (Signature, Signature, (Institution, Signature, Sentences), Sig \end{array}$$

If $c_1, \ldots, c_n$ are all `Correspondences` 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 $i = 1, \ldots, n$,
$cvalues = \bigvee_{i=1,\ldots,n} cvalues_i$,
$smap = \{s_i^1 \mapsto s_i^2\}$,
$(\mathcal{I}, \Sigma, \Delta, \phi_s : \Sigma_s \to \Sigma, \phi_t : \Sigma_t \to \Sigma) = theoryOfCorrespondences_{\Gamma.logic}(aSem, \Sigma_1, \Sigma_2, clist_1 ++ \ldots ++ clist_n)$.

10.2.4.18 Semantics of correspondences

$$sem(\Gamma, \Sigma_1, \Sigma_2, (defaultConf, defaultRel), \texttt{Correspondence}) = (clist, cvalues) : (Sequence((Relation, Symbol, Symbol)), Bool)$$

If c is a `DefaultCorrespondence`,

$$sem(\Gamma, \Sigma_1, \Sigma_2, (defaultConf, defaultRel), c) = (clist, cvalues)$$

where $cvalues = True$ if $defaultConf$ is different than 1 and $False$ otherwise,
$Sequence((sym_1^1, sym_1^2), \dots, (sym_k^1, sym_k^2)) = sameName_{\Gamma.logic}(\Sigma_1, \Sigma_2)$,
$clist = Sequence((defaultRel, sym_i^1, sym_i^2)\}_{i=1,\dots,k})$.

If c is a `SingleCorrespondence`,

$$sem(\Gamma, \Sigma_1, \Sigma_2, (defaultConf, defaultRel), c) = (clist, cvalues)$$

where $conf = \begin{cases} defaultConf & c.\texttt{confidence} \text{ is missing,} \\ c.\texttt{confidence} & \text{otherwise} \end{cases}$

$rel = \begin{cases} defaultRel & c.\texttt{relation} \text{ is missing,} \\ c.\texttt{relation} & \text{otherwise} \end{cases}$

$cvalues = \begin{cases} False & conf = 1 \\ True & \text{otherwise} \end{cases}$

$sem(\Gamma, \Sigma_1, c.generalizedTerm) = sym1$, $sem(\Gamma, \Sigma_2, c.symbolRef) = sym2$, $clist = Sequence((rel, sym1, sym2))$.

If c is a `CorrespondenceBlock`,

$$sem(\Gamma, \Sigma_1, \Sigma_2, (defaultConf, defaultRel), c) = (clist, cvalues)$$

where if $c.\texttt{relation}$ is missing, $rel = defaultRel$, else $rel = c.\texttt{relation}$
if $c.\texttt{confidence}$ is missing, $conf = defaultConf$, else $conf = c.\texttt{confidence}$
for all correspondences $c_1, \dots, c_n$ in $c.\texttt{correspondence}$, $(clist_i, cvalues_i) = sem(\Gamma, \Sigma_1, \Sigma_2, (conf, rel), c_i)$,
$clist = clist_1 ++ \dots ++ clist_n$ and $cvalues = \vee_{i=1,\dots,n} cvalues_i$.

$$sem(\Gamma, L_1, L_2, \texttt{AlignmentSemantics}) = ((Name_1, \mathcal{I}_1', \Sigma_1', \mathcal{M}_1', \Delta_1'), (Name_2, c\mathcal{I}_2', \Sigma_2', \mathcal{M}_2', \Delta_2')) : (NodeLabel, NodeLabel)$$

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

$$rel(L) = \begin{cases} (L_1, L_2) & \text{if } s = \texttt{single-domain} \\ relativize_{logic(\Gamma.current)}(Lname_1, \mathcal{I}_1, \Sigma_1', \mathcal{M}_1', \Delta_1'), (name_2, \mathcal{I}_2, \Sigma_2', \mathcal{M}_2', \Delta_2') & \text{otherwise} \end{cases}$$

~~where the relativization procedure is logic-specific. An example of this is done for OWL can be found in [?]~~ $relativize_{\mathcal{I}_1}(\Sigma_1, \Delta_1) = (\Sigma_1', \Delta$
$\mathcal{M}_1'$ is the class of Δ_1'-models, $name_1 =' relativized' ++ aName$,
$relativize_{\mathcal{I}_2}(\Sigma_2, \Delta_2) = (\Sigma_2', \Delta_2')$, $\mathcal{M}_2'$ is the class of Δ_2'-models and $name_2 =' relativized' ++ aName'$.

Annex

A Annex: DOL Registry

(Normative)

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.

B Annex: DOL Ontology

(Informative)

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.

~~This annex specifies the Ontology, an RDF vocabulary that implements the terms and definitions from clause 4. Part of the background and design considerations of the Ontology can be found in [?] .~~

B.1 Namespace Definitions

The namespaces and prefixes corresponding to external elements required for use by the DOL ontology are provided below. Table ?? lists the prefixes and namespaces on which DOL depends.

Prefix	Namespace
~~The Ontology is currently implemented in OWL 2 (). The normative snapshots are encoded in RDF/XML using the OWL 2 mapping to RDF graphs ().~~ 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/`

Table B.2: Prefix and Namespaces for referenced/external vocabularies

The ~~ontology makes use of the following standard ontologies and vocabularies:~~ namespace approach taken for DOL is based on OMG guidelines and is constructed as follows:

- ~~DCMI Metadata Terms ()~~ A standard prefix `http://www.omg.org/spec/`
- ~~OMG Specification Metadata (SM) Vocabulary ()~~ The abbreviation for the specification: in this case DOL
- ~~SKOS ()~~ The ontology name

~~The sources of the ontology are being maintained in OWL Manchester syntax [?] at .~~

~~It is intended to implement future versions of the ontology as a DOL document in~~ Note that the URI/IRI strategy for the ontology takes a "slash" rather than "hash" approach, in order to accommodate server-side applications. Table **??** provides the namespace definition for the DOL ontology. While the prefixes given in Tables **??** and **??** are informative, their use is required in any extension, including API4KP.

~~Applications of the Ontology include modeling statements about OMS in RDF, e.g., when annotating OMS, or when describing new conforming logics, OMS languages, serializations, translations, etc., in the of -conforming languages and translations detailed below in clause **??**.~~

~~The classes in the Ontology (and their annotations) correspond to the terms (and their definitions) in clause 4. Classes that are reifications of relations also have been introduced as object properties. All classes and object properties are assumed to be in the Ontology namespace unless stated otherwise. The Ontology additionally contains some top-level abstract classes as follows:~~

~~This reflects central issues in the structure of : while , as a language, is a linguistic entity, it is related to mathematical entities like logics, signatures and models through its semantics. That is, semantic entities provide the bridge bewteen linguistic and mathematical entities. Moreover, processes (like theorem proving) provide algorithmic procedeures for manipulating DOL librares, and tools implement these in software.~~

~~The top level object properties are structured in a similar way:~~

Prefix	**Namespace**
dol	`http://www.omg.org/spec/DOL/DOL-terms/`

Table B.4: Prefix and Namespaces for the DOL ontology

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.

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

~~There will be Maintenance Authority (MA) or, depending on advisability, a Registration Authorityestablished to maintain the registry as an informative resource governed by the standard. The registry contents itself will not be normative; however, it is expected to become the basis for normative activities.~~

C Annex: Conformance of OWL 2 DL With DOL

(Informative)

The semantic conformance of OWL 2 DL (as specified in **NR2**) with DOL is established in [**?**].

C.1 Abstract Syntax Conformance of OWL 2 With DOL

The metaclass `OWL Ontology` is a subclass (in the sense of SMOF **NR26** multiple classification) of `NativeDocument`. The metaclass `OWL Universe` is a subclass (in the sense of SMOF **NR26** multiple classification) of `BasicOMS`.

C.2 Conformance of the OWL Serializations With DOL

C.2.1 Text Conformance of the OWL 2 Manchester Syntax With DOL

The OWL 2 Manchester syntax satisfies the criteria for text conformance established in clause 2.2 in a straightforward way thanks to its line-based comment syntax (comments starting with #) and its flexible handling of line breaks.

C.2.2 Conformance of the XML and RDF Serializations of OWL With DOL

C.2.2.1 General Issues

With minor modifications detailed below, the OWL/XML serialization [**?**] 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.2.2.2 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 **??**).

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.2.2.3 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[1]. 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[2].

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

[1] **NR20**, section 3
[2] See the last sentence of section 3.2.5 of **NR20**

```
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.3 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. Models are first-order structures $I = (\Delta^I, .^I)$ with universe Δ^I that interpret concepts as unary and roles as binary predicates (using $.^I$). $I_1 \leq I_2$ if $\Delta^{I_1} = \Delta^{I_2}$ and all concepts and roles of I_1 are subconcepts and subroles of those in I_2. Sentences are subsumption relations $C_1 \sqsubseteq C_2$ between concepts, where concepts follow the grammar*

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

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

The logic $\mathcal{SROIQ}$ [?], which is the logical core of the Web Ontology Language OWL 2 DL[3], 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 [?].

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 [?]:

Definition 16 *A datatype map, formalizing datatype maps from the OWL 2 Specification [?], 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 $(LV, DT)^{LS} \in (DT)^{DT}$.*

[3]See also `http://www.w3.org/TR/owl2-overview/`

- *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.* □

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 [?], 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 [?], 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 [?], 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 [?].*
- 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*

$$I = (\Delta_I, \Delta_D, \cdot^C, \cdot^{OP}, \cdot^{DP}, \cdot^I, \cdot^{DT}, \cdot^{LT}, \cdot^{FA}, \mathit{NAMED})$$

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* $(a)^I \in \mathit{NAMED}$ *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 $N_{DT} \cup \{rdfs{:}Literal\}$,*
  - *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 [?]. Sentence translation is substitution of symbols.*
- *(D, V)-models are interpretations for D and V. (D, V)-model morphisms are maps between the domains Δ_I preserving membership in classes and properties, where Δ_D is mapped identically. Model reducts are built by first translating along the signature morphism and then looking up the interpretation in the model to be reduced.*
- *The satisfaction relation is defined as in direct model-theoretic semantics of OWL [?].*

□

Remark: strictly speaking, the institution defined above is *OWL 2 DL without restrictions* in the sense of [?]. 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.3.1 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'-model, we get that M is indeed a (C, R, I)-model and moreover $M \models \Delta$.*

Note If $T = ((C, R, I), \Delta)$ is an OWL theory, M' is a $\tilde{T}$-model and e is a (C, R, I)-sentence, we have that $M' \models \alpha(e)$ if and only if $\beta(M') \models e$.

C.3.2 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 $\top_t$.[4]
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}

     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`

[4] 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.

Note that we must express equivalences where role compositions are involved. This is only possible in OWL2 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 OWLin OWL2 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.

D Annex: Conformance of Common Logic with DOL

(Informative)

D.1 Abstract Syntax Conformance of Common Logic With DOL

The metaclass `Text` is a subclass (in the sense of SMOF **NR26** multiple classification) of `NativeDocument`. The metaclass `Sentence` 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 [?].

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-model $I = (UR, UD, rel, fun, int, seq)$ consists of a set UR, the universe of reference, with a non-empty subset $UD \subseteq UR$, the universe of discourse, and four mappings:*

- *rel from UR to subsets of* ~~$UD^* = \{< x_1, \ldots, x_n > | x_1, \ldots, x_n \in UD\}$~~ $UD^* = \{\langle x_1, \ldots, x_n \rangle \mid x_1, \ldots, x_n \in UD\}$ *(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.

Model reducts are defined in the following way: Given a signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ and a Σ_2-model $I_2 = (UR, UD, rel, fun, int, seq)$, $I|_\sigma = (UR, UD, rel, fun, int \circ \sigma, seq \circ \sigma)$.

Given two CL models $I_1 = (UR_1, UD_1, rel_1, fun_1, int_1, seq_1)$ and $I_2 = (UR_2, UD_2, rel_2, fun_2, int_2, seq_2)$, a homomorphism $h : I_1 \to I_2$ is a function $h : UR_1 \to UR_2$ such that

- *h restricts to $k : UD_1 \to UD_2$,*
- *for each $x \in UR_1$ and $s \in UD_1^*$, if $s \in rel_1(x)$, then $k^*(s) \in rel_2(h(x))$[1],*
- *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.

[1] k^* is the extension of h to sequences.

E Annex: Conformance of RDF and RDF Schema with DOL

(Informative)

~~The semantic conformance of RDF Schema (as specified in) with is established in [?]~~

E.1 Abstract Syntax Conformance of RDF and RDF Schema With DOL

The metaclass `Document` is a subclass (in the sense of SMOF **NR26** multiple classification) of `NativeDocument`. The metaclass `Triple` 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 DOLis established in [?] .

Definition 23 (RDF **and** RDF **Schema**) *The institutions for the Resource Description Framework (*RDF*) and* RDF~~*Schema*~~ *Schema (also known as* RDFS*), respectively, are defined ~~following~~ in the following [?]. 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* $\mathbf{R_s}$*-model* $M = \langle R_m, P_m, S_m, EXT_m \rangle$ *consists of a set* R_m *of resources, a set* $P_m \subseteq R_m$ *of predicates, a mapping function* $S_m : \mathbf{R_s} \to R_m$*, 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 models.*

Actually, the standard vocabulary is given by a certain theory. In case of RDF*, it contains e.g. resources* `rdf:type` *and* `rdf:Property` *and* `rdf:subject`*, and sentences like, e.g.*

$$(\mathtt{rdf:type}, \mathtt{rdf:type}, \mathtt{rdf:Property}),$$

~~*and*~~

$$(\mathtt{rdf:subject}, \mathtt{rdf:type}, \mathtt{rdf:Property}).$$

$$(\mathtt{rdf:type}, \mathtt{rdf:type}, \mathtt{rdf:Property}) \text{ , and}$$

$$(\mathtt{rdf:subject}, \mathtt{rdf:type}, \mathtt{rdf:Property}) \text{ .}$$

In the models, the standard vocabulary is interpreted with a fixed model. Moreover, for each RDF*-model* $\cancel{M = \langle R_m, P_m, S_m, EXT_m \rangle} M =$ *if* $p \in P_m$*, then it must hold* $\cancel{(p, S_m(\mathtt{rdf:Property})) \in EXT_m(\mathtt{rdf:type})} (p, S_m(\mathtt{rdf:Property})) \in EXT_m(\mathtt{rdf:type})$*. 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`~~ `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 *[?].* □

F Annex: Conformance of UML class and object models with DOL

(Informative)

This informative annex demonstrates conformance of a subset of UML class and object ~~diagrams~~ models with DOL by defining an institution for both. The subset is restricted to the static aspects of class ~~diagrams~~models; that is, change of state is ignored. This means that all operations are query operations.

F.1 Abstract Syntax Conformance of UML With DOL

The metaclass `OWL Model` is a subclass (in the sense of SMOF **NR26** multiple classification) of `NativeDocument`. The metaclass `OWL PackageableElement` is a subclass (in the sense of SMOF **NR26** multiple classification) of `BasicOMS`.

F.2 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.3 Semantic Conformance of UML With DOL

The institution of UML class and object ~~diagrams~~ models is defined using a translation of UML class ~~diagrams~~ models to Common Logic, following the fUML specification and [?].

F.3.1 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 ~~diagrams~~ 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
  // ~~fuml:sequence - membership~~  Membership  of an element in a sequence
  (forall (x s)
          (if (form:sequence-member x s)
              (form:Sequence s)))
      (forall (x s)
          (iff (form:sequence-member x s)
               (exists ( ~~pt~~  p )
                       (and (form:in-sequence s  ~~pt~~  p )
```

```
                                        (form:in-position  ~~pt~~   p  x)))))

  //  ~~selection~~   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~~   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~~   Insertion   of elements
  (forall (x s1 s2)
          ~~// inserting an element means...~~  (if (= (form:sequence-insert x s1) s2)
              (and (form:Sequence s1) (form:Sequence s2)
                   //  ~~the~~   The   new element is at the first position   ...   (form:in-position-c
                   //  ..   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~~   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   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   No   element can be inserted twice
                     (  forall (x s) (  if (from:ordered-set-member x   s) (= (form:ordered-set-insert
x s) s)    s1) (form:same-ordered-set s1 s2  ))
                     //   inserting   Inserting   a new element
                     (  forall (x s) (  if (not (from:ordered-set-member x s1))
                         (  exists (s2) (and (= (form:ordered-set-insert x s1) s2)    and   //
the    The   new element is at the first position   ...   (form:in-ordered-set-position-count s2 form
                               //   ...   and all other elements are shifted by one
                               (forall (n1 n2 y)
                                       (if (form:add-one n1 n2)
                                           (iff (form:in-ordered-set-position-count s1 n1 y)
                                                (form:in-ordered-set-position-count s2 n2 y)))))))
))  end

oms sets =
  // An empty set has no members.
  (forall (s)
          (if (form:empty-set s)
              (form:Set s)))
  (forall (s)
          (if (form:Set s)
              (iff (form:empty-set s)
                   (not (exists (x)
                                (form:set-member x s))))))
  // Size of sets
  (forall (s n)
          (if (form:set-size s n)
              (and (form:Set s)
                   (buml:UnlimitedNatural n))))
  (= (form:set-size form:empty-set) form:0)
  (forall (x s)
          (if (not (form:set-member x s))
              (exists (n)
                      (and (form:add-one (form:set-size s) n)
                           (= (form:set-size (form:set-insert x s)) n)))))

  // The same-set relation is true for sets that have the same members.
    // but:  why not replace same-set with = ?  (forall (s1 s2)
          (if (form:same-set s1 s2)
              (and (form:Set s1)
```

```
                    (form:Set s2))))
  (forall (s1 s2)
          (iff (form:same-set s1 s2)
               (forall (x)
                       (iff (form:set-member x s1)
                            (form:set-member x s2)))))
  // Insertion of elements into sets and set membership
  (forall (x s)
          (if (form:Set s)
              (form:Set (form:set-insert x s))))
  (forall (x y s)
          (iff (form:set-member x (form:set-insert y s))
               (or (= x y)
                   (form:set-member x s))))
end

oms bags =
  // An empty bag has no members.
  (forall (s)
          (if (form:empty-bag s)
              (form:Bag s)))
  (forall (s)
          (if (form:Bag s)
              (iff (form:empty-bag s)
                   (not (exists (x)
                        (form:bag-member x s))))))
  // Size of bags
  (forall (s n)
          (if (form:bag-size s n)
              (and (form:Bag s)
                   (buml:UnlimitedNatural n))))
  (= (form:bag-size form:empty-bag) form:0)
  (forall (x s)
          (exists (n)
                  (and (form:add-one (form:bag-size s) n)
                       (= (form:bag-size (form:bag-insert x s)) n))))

  // The same-bag relation is true for bags that have the same members.
  (forall (s1 s2)
          (if (form:same-bag s1 s2)
              (and (form:Bag s1)
                   (form:Bag s2))))
  (forall (s1 s2)
          (iff (form:same-bag s1 s2)
               (forall (x)
                       (iff (form:bag-member-count x s1)
                            (form:bag-member-count x s2)))))
  // Insertion of elements into bags and bag membership
  (forall (x s)
          (if (form:Bag s)
              (form:Bag (form:bag-insert x s))))
  (forall (x y s)
          (iff (form:bag-member x (form:bag-insert y s))
               (or (= x y)
                   (form:bag-member x s))))
  // Member count
  (forall (x s)
          (if (form:Bag s)
              (buml:UnlimitedNatural (form:bag-member-count x s))))
  (= (form:bag-member-count form:empty-bag) form:0)
  (forall (x s)
          (exists (n)
```

```
                    (and (form:add-one (form:bag-member-count x s) n)
                         (= (form:bag-member-count x (form:bag-insert x s)) n))))
  (forall (x y s)
          (if (not (= x y))
              (= (form:bag-member-count x (form:bag-insert y s))
                 (form:bag-member-count x s))))
end

oms collection-types =
  sequences-insert and ordered-sets and sets and bags
then
  //  bag  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  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  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  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  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)
          (if (buml:leq x y)
               (or (= x y)
```

```
                          (buml:less-than x y))))
end

oms uml-cd-preliminaries =
  collection-types and pairs
end
```

F.3.2 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:[1]

	ordered	not ordered
unique	OrderedSet	Set
not unique	Sequence	Bag

The default is "not ordered" and "unique".[2]

For a class/data type $c \in C$ of a class/data type-hierarchy $(C, \leq_C)$ and a collection type constructor

$$\tau \in \{\mathsf{OrderedSet}, \mathsf{Set}, \mathsf{Sequence}, \mathsf{Bag}\},$$

$$\tau \in \{\mathsf{OrderedSet}, \mathsf{Set}, \mathsf{Sequence}, \mathsf{Bag}\},$$

the expression $\tau[c]$ denotes the induced *collection type*.

Let $(C, \leq_C)$ be a class/data type hierarchy.

- An *attribute declaration*[3] over $(C, \leq_C)$ is of the form $c.p : \tau[c']$ with $c, c' \in C$, τ a collection type constructor, and p an *attribute name*. ~~(~~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 $c.q(x_1 : \tau_1[c_1], \ldots, x_r : \tau_r[c_r]) : \tau[c']$ with $c, c_1, \ldots, c_r, c' \in C$, τ a collection type constructor, o an *operation name*, and $x_1, \ldots, x_r$ *parameter names*.
- An *association declaration* over $(C, \leq_C)$ is of the form $a(p_1 : \tau_1[c_1], \ldots, p_r : \tau_r[c_r])$ with $r \geq 2$, $c_1, \ldots, c_r \in C$, $\tau_1, \ldots, \tau_r$ classifier annotations, a an *association name*, and $p_1, \ldots, p_r$ *member end names*.[4] An association declaration $\mathbf{a} = a(p_1 : \tau_1[c_1], \ldots, p_r : \tau_r[c_r])$ *yields* the *property declarations* $\mathbf{a}.p_i : \tau_i[c_i]$ for $1 \leq i \leq r$. An association declaration is *binary* if $r = 2$.[5]

  ~~A *composition declaration* over $(C, \leq_C)$ is of the form $m(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2])$ with $c_1, c_2 \in C$, τ_2 a collection type constructor, *m a composition name*, and p_1, p_2 *member end names*. A composition declaration $\mathbf{m} = m(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2])$ *yields* the property declarations $\mathbf{m}.p_1 : \mathsf{Set}[c_1]$~~ For a binary association with $\tau_1 = \mathsf{Set}$, the second member end may be *composite*, and ~~$\mathbf{m}.p_2 : \tau_2[c_2]$.~~ we write $a(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2])$ if this fact plays a rôle.[6]

~~In UML, each may have . However, such an aggregation kind has no semantic meaning when the property is not a member end of an association: the UML Superstructure Specification 2.4.1 does not mention the aggregation kind in the description of the semantics of , and UML 2.5 explains the use of aggregations for as "to model circumstances in which *one instance* is used to group together a set of instances" (p. 112, our emphasis). Moreover, composite properties, i.e., properties with~~

[1] ~~Cf. UML Superstructure Specification 2.4.1, p. 128; UML 2.5, p~~[?, p. 34] . ~~27.~~

[2] ~~UML Superstructure Specification 2.4.1, p. 96~~[?, p. 96] ; there does not seem to be default in UML 2.5.

[3] We separate attributes from association member ends due to their different uses. In UML, both are of class Property ([?, p. 109]).

[4] The member ends are ordered~~according to the UML Superstructure Specification 2.4.1, p. 29; UML 2.5, p. 206;~~ [?, p. 197] hence they are represented in a tuple-like notation.

[5] Only binary association may show member ends that are properties not owned by the association~~(UML Superstructure Specification 2.4.1, p. 37; UML 2.5, p. 228)~~[?, p. 218] . The property declarations induced by a more than binary association result in a query operation.

[6] Composite properties, i.e., properties with aggregation kind composite can only be member ends of binary associations [?, p. 218] and their multiplicity must not exceed one [?, p. 150] .

~~aggregation kind can only be member ends of binary associations (UML Superstructure Specification 2.4.1, p. 37; UML 2.5, p. 228) and their multiplicity must not exceed one (UML Superstructure Specification 2.4.1, p. 126; UML 2.5, p. 155). Thus, composition declarations are distinguished from general association declarations.~~

Class/data type nets (Signatures). A *class/data type net* ~~$\Sigma = ((C, \leq_C), P, O, A, M)$~~ $\Sigma = ((C, \leq_C), P, O, A)$ 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 $(C, \leq_C)$, ~~and a set M of composition 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 ~~and composition~~ names are unique: if d_1 and d_2 are the names of two different association ~~or composition declarations in $M \cup A$~~ 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~~or a composition declaration in M~~, then $p_i \neq p_j$ for $1 \leq i \neq j \leq r$;[7]
- the type of a member end[8] owned by a class/data type coincides with its declarations as attribute: We say that a property declaration $\mathbf{a}.p_i : \tau_i[c_i]$ yielded by a binary association $\mathbf{a} = a(p_1 : \tau_1[c_1], p_2 : \tau_2[c_2])$ is *owned by* $c_0 \in C$, if $c_{3-i} \leq_C c_0$ and there is an attribute declaration $c_0.p_i : \tau_i[c_i] \in P$~~; and similarly for property declarations yielded by composition declarations~~, 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 ~~generalization~~ generalisation hierarchy only a single attribute with name p_i may exist.)

A *class/data type net morphism* ~~$\sigma = (\gamma, \varphi, \alpha, \mu) : \Sigma = ((C, \leq_C), P, A, M) \to \mathrm{T} = ((D, \leq_D), Q, B, N)$~~ $\sigma = (\gamma, \varphi, \alpha) : \Sigma = ((C, \leq_C), P, A$ 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.
- a query operation declaration map $\rho : O \to R$ such that if $\rho(c.q(x_1 : \tau_1[c_1], \dots, x_r : \tau_r[c_r]) : \tau[c']) = d.r(x_1 : \tau'_1[d_1], \dots, x_r : \tau'_r[d_r]) : \tau[d'] \in R$, then $d = \gamma(c)$, $d_i = \gamma(c_i)$, $d' = \gamma(c')$, $\tau'_i = \tau_i$ and $\tau = \tau'$;
- an association declaration map $\alpha : A \to B$ such that if $\alpha(a(p_1 : \tau_1[c_1], \dots, p_r : \tau_r[c_r])) = b(q_1 : \tau'_1[d_1], \dots, q_s : \tau'_s[d_s]) \in B$, then $r = s$ and $d_i = \gamma(c_i)$ and $\tau_i = \tau'_i$ for $1 \leq i \leq r$, and member ends owned by the association are mapped into owned member ends~~;~~
~~a composition declaration map $\mu : M \to N$ such that if $\mu(m(p_1 : \mathsf{Set}[c_1], \blacklozenge p_2 : \tau_2[c_2])) = n(q_1 : \mathsf{Set}[d_1], \blacklozenge q_2 : \tau'_2[d_2]) \in N$, then $d_1 = \gamma(c_1)$, $d_2 = \gamma(c_2)$, and $\tau_2 = \tau'_2$, and member ends owned by the composition are mapped into owned member ends~~.

Class/data type nets as objects and class/data type net morphisms as morphisms form the category of *class/data type nets*, denoted by Cl.

For the example in Fig. **??** the class/data type net is

Classes/data types: Net, Station, Line, Connector, Unit, Track, Point, Linear,
Boolean, UnlimitedNatural, Integer, Real, String

Generalizations: Point $\leq$ Unit, Linear $\leq$ Unit

Properties: Line.linear : Set[Boolean], Track.linear : Set[Boolean],
~~Net.station : Set[Station]~~ Net◆station : Set[Station], ~~Net.line : Set[Line]~~ Net◆line : Set[Line],
Station.net : Set[Net], ~~Station.unit : Set[Unit]~~ Station◆unit : Set[Unit], ~~Station.track : Set[Track]~~ Station◆track : Set[Track],
Line.net : Set[Net], Line.linear : Set[Linear],
Connector.unit : Set[Unit],
Unit.station : Set[Station], Unit.connector : Set[Connector],
Track.station : Set[Station], Track.linear : Set[Linear],
Linear.track : Set[Track], Linear.line : Set[Line]

Associations: ~~l2l(line : Set[Line], linear : Set[Linear])~~ L2L(line : Set[Line], linear : Set[Linear]),
~~l2t(linear : Set[Linear], track : Set[Track])~~ L2T(linear : Set[Linear], track : Set[Track]),

[7] In UML, member end names need not be unique. However, for (1) a simpler handling of selecting a particular member end in the sentences and avoiding the use of number selectors, and (2) making the notion of member ends "owned" by a class/data type, 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 ~~$1, \dots, r$~~ $1, \dots, r$.

[8] All member ends are instances of Property~~; UML Superstructure Specification 2.4.1, p. 36; UML 2.5, p~~[?, p. 206]~~. 206.~~

c2u(connector : Set[Connector], unit : Set[Unit])Compositions: n2s(net : Set[Net], ◆station : Set[Station])C2U(connecto

N2S(net : Set[Net], ◆station : Set[Station]),

n2l(net : Set[Net], ◆line : Set[Line]) N2L(net : Set[Net], ◆line : Set[Line]),

s2u(station : Set[Station], ◆unit : Set[Unit]) S2U(station : Set[Station], ◆unit : Set[Unit]),

s2t(station : Set[Station], ◆track : Set[Track]) S2T(station : Set[Station], ◆track : Set[Track])

Here all member ends are owned by class/data types.

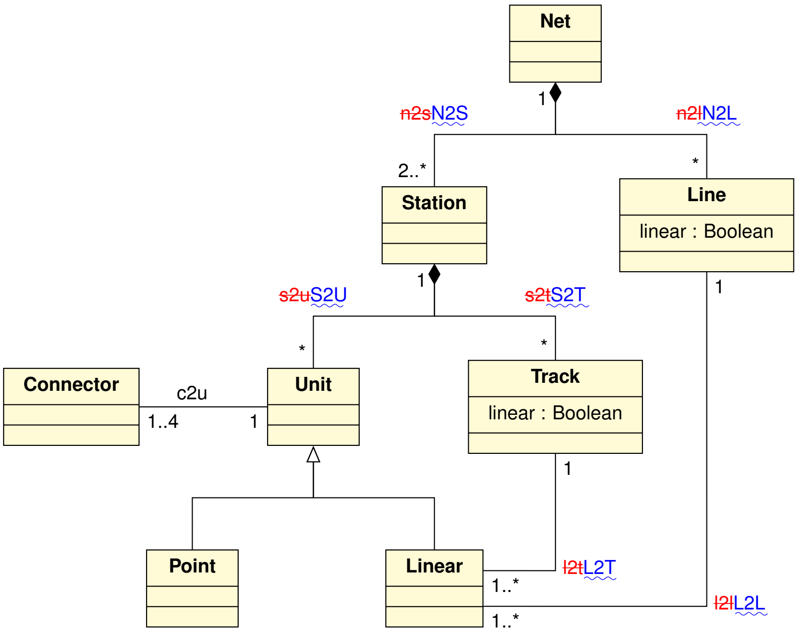


Figure F.1: Sample UML class ~~diagram.~~model

F.3.3 Models

As stated above, models (in the sense of the term model defined in clause 4) of UML class ~~diagrams~~ models are obtained via a translation to Common Logic.

For a classifier net ~~$\Sigma = ((C, \leq_C), K, P, M, A)$~~$\Sigma = ((C, \leq_C), P, O, A)$, a Common Logic theory ~~$CL(\Sigma)$~~ $\mathsf{CL}(\Sigma)$ is defined consisting of:

- for $c \in C$, a predicate[9] $\mathsf{CL}(c)$, such that
  - $\mathsf{CL}(\mathsf{Boolean}) =$ `buml:Boolean`,
  - $\mathsf{CL}(\mathsf{String}) =$ `buml:String`,
  - $\mathsf{CL}(\mathsf{Integer}) =$ `buml:Integer`,
  - $\mathsf{CL}(\mathsf{UnlimitedNatural}) =$ `form:NaturalNumber`,
  - $\mathsf{CL}(\mathsf{Real}) =$ `buml:Real`,
  - ~~$\mathsf{CL}(c) = c$~~$\mathsf{CL}(c) = c$, if c is an enumeration type with values ~~$k_1, \dots, k_n$~~$k_1, \dots, k_n$. In this case, additionally, the Common Logic theory is augmented by ~~`(not (=` $k_i \cdots k_j$`))` for $i \neq j$ and `(forall (x) (if (c x) (or (= x` k_1`)` $\cdots$ `(= x` k_n`)))),`~~

    * `(not (=` k_i k_j`))` for $i \neq j$@(forall (x) (if ($\mathsf{CL}(c)$ x) (or (= x k_1) $\cdots$ (= x k_n)))) @*
  - ~~$\mathsf{CL}(\mathsf{List}[c]) =$ `form:Sequence,`~~ $\mathsf{CL}(\mathsf{List}[c]) = \mathsf{CL}(\mathsf{List}) =$ `form:Sequence,`
  - ~~$\mathsf{CL}(\mathsf{Set}[c]) =$ `form:Set,`~~ $\mathsf{CL}(\mathsf{Set}[c]) = \mathsf{CL}(\mathsf{Set}) =$ `form:Set,`
  - ~~$\mathsf{CL}(\mathsf{OrderedSet}[c]) =$ `form:OrderedSet,`~~ $\mathsf{CL}(\mathsf{OrderedSet}[c]) = \mathsf{CL}(\mathsf{OrderedSet}) =$ `form:OrderedSet,`
  - ~~$\mathsf{CL}(\mathsf{Bag}[c]) =$ `form:Bag,`~~ $\mathsf{CL}(\mathsf{Bag}[c]) = \mathsf{CL}(\mathsf{Bag}) =$ `form:Bag,`

[9]Strictly speaking, this is just a name.