1 Scope

1.1 General

This OMG Specification specifies the Distributed Ontology, Modeling and Specification Language (DOL). DOL is designed to achieve integration and interoperability of ontologies, specifications and 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 [22].

1.2 Background Information

Logical languages are used in several fields of computing for the development of formal, machine-processable texts that carry a formal semantics. Among those fields are 1) Ontologies formalizing domain knowledge, 2) (formal) Models of systems, and 3) the formal Specification 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).

1.3 Features Within Scope

The following are within the scope of this OMG Specification:

1) homogeneous OMS as well as heterogeneous OMS (OMS that consist of parts written in different languages);
2) mappings between OMS (which map OMS symbols to OMS symbols);
3) OMS networks (involving several OMS and mappings between them);
4) translations between different OMS languages conforming with DOL (translating a whole OMS to another language);
5) structuring constructs for modeling non-monotonic behavior;
6) annotation and documentation of OMS, mappings between OMS, symbols, and sentences;
7) recommendations of vocabularies for annotating and documenting OMS;
8) a syntax for embedding the constructs mentioned under (1)–(6) as annotations into existing OMS;
9) a syntax for expressing (1)–(6) as standoff markup that points into existing OMS;
4 Terms and Definitions

4.1 Distributed Ontology, Modeling and Specification Language

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

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

4.2 Native OMS, OMS, and OMS Languages

native OMS collection of expressions (like non-logical symbols, sentences and structuring elements) from a given OMS language.

Example A UML class model, an ontology written in OWL 2 EL, and a specification written in CASL are three different native OMS.

Note An OMS can be written in different OMS language serializations.

native document document containing a native OMS.

DOL document document containing a DOL library.

OMS language language equipped with a formal, declarative, logic-based semantics, plus non-logical annotations.

Example OMS languages include OWL 2 DL, Common Logic, F-logic, UML class models, RDF Schema, and OBO.

Note An OMS language is used for the formal specification of native OMS.

Note An OMS language has a logical language aspect, a structuring language aspect, and an annotation language aspect.

DOL structured OMS syntactically valid DOL expression denoting an OMS that is built from smaller OMS as building blocks.

Note DOL structured OMS, typically, use basic OMS as building blocks for defining other structured OMS, OMS mappings or OMS networks.

Note All DOL structured OMS are structured OMS.

ontology logical theory that is used as a shard conceptualization

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). In this document, we use the term realization for models in the sense of model theory in logic.

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) OMS (ontology, specification or 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).
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 occurring 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** set (or otherwise structured collection) of non-logical symbols of an OMS.

**Note** The signature of a term is the set of all non-logical symbols occurring in the term. The notion of signature depends on the OMS language or logic.

**Note** The signature of an OMS is usually unequivocally determinable.

**realization** semantic interpretation of all non-logical symbols of a signature.

**Note** A model realization of an OMS is a model realization of the signature of the OMS that also satisfies all the additional constraints expressed by the OMS. In case of flattenable OMS, these constraints are expressed by the axioms of the OMS.

**Note** This term refers to model in the sense of model theory (a branch of logic). It is not to be confused with in the sense of modeling (i.e., the “M” in OMS). The notion of model depends on the OMS language or logic.

**Note** In logical model theory, a realization is called “model”. However, we have reserved the term “model” for models in the sense of model-driven engineering.

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

**term** syntactic expression either consisting of a single non-logical symbol or recursively composed of other terms (a.k.a. its subterms).

**Note** A term belongs to the logical language aspect of an OMS language.

**sentence** term that is either true or false in a given realization, i.e. which is assigned a truth value in this realization.

**Note** In a realization, on the one hand, a sentence is always true or false. In an OMS, on the other hand, a sentence can
have several logical statuses. For example, a sentence can be: an axiom, if postulated to be true; a theorem, if proven from other axioms and theorems; or a conjecture, if expecting to be proven from other axioms and theorems.

**Note** A sentence can conform to one or more signatures (namely those signatures containing all non-logical symbols used in the sentence).

**Note** It is quite common that sentences are required to be closed (i.e. have no free variables). However, this depends on the OMS language at hand.

**Note** A sentence belongs to the logical language aspect of an OMS language.

**Note** The notion of sentence depends on the OMS language or logic.

**satisfaction relation** relation between realizations and sentences indicating which sentences hold true in the realization.

**Note** The satisfaction relation depends on the OMS language or logic.

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

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

**entailment**

**logical consequence**

**specialization** relation between two OMS (or an OMS and a sentence, or two OMS networks, or an OMS network and an OMS) expressing that the second item (the conclusion) is logically implied by the first one (the premise).

**Note** Entailment expresses that each model realization satisfying the premise also satisfies the conclusion.

**Note** The converse is generalization.

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

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

**tool** software for processing DOL libraries and OMS. **theorem proving** process of demonstrating 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 \( O_1 \).
An owl:import in OWL is an import.
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 environment, 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 superset of those present in the first OMS.

**Note** The second OMS is said to extend the first, and is an extension of the first OMS.

**conservative extension** extension that does not add new logical properties with respect to the signature of the extended OMS.

**Note** An extension is a consequence-theoretic or model-theoretic conservative extension. If used without qualification, the model-theoretic version is meant.

**consequence-theoretic conservative extension** extension that does not add new theorems (in terms of the unextended signature).

**Note** An extension $O_2$ of an OMS $O_1$ is a consequence-theoretic conservative extension, if all properties formulated in the signature of $O_1$ hold for $O_1$ whenever they hold for $O_2$.

**model-theoretic conservative extension** extension that does not lead to a restriction of class of models of an OMS.

**Note** An extension $O_2$ of an OMS $O_1$ is a model-theoretic conservative extension, if each model-realization of $O_1$ can be expanded to a model-realization of $O_2$.

**Note** Each model-theoretic conservative extension is also a consequence-theoretic one, but not vice versa.

**monomorphic extension** extension whose newly introduced non-logical symbols are interpreted in a way unique up to isomorphism.

**Note** An extension $O_2$ of an OMS $O_1$ is a monomorphic extension, if each model-realization of $O_1$ can be expanded to a model-realization of $O_2$ that is unique up to isomorphism.

**Note** Each monomorphic extension is also a model-theoretic conservative extension but not vice versa.

**definitional extension** extension whose newly introduced non-logical symbols are interpreted in a unique way.

**Note** An extension $O_2$ of an OMS $O_1$ is a definitional extension, if each model-realization of $O_1$ can be uniquely expanded to a model-realization of $O_2$.

**Note** $O_2$ being a definitional extension of $O_1$ implies a bijective correspondence between the classes of model-realizations of $O_2$ and $O_1$.

**Note** Each definitional extension is also a monomorphic extension but not vice versa.

**weak definitional extension** extension whose newly introduced non-logical symbols can be interpreted in at most one way.

**Note** An extension $O_2$ of an OMS $O_1$ is a weak definitional extension, if each model-realization of $O_1$ can be expanded to at most one model-realization of $O_2$.

**Note** An extension is definitional if and only if it is both weakly definitional and model-theoretically conservative.

**implied extension** model-theoretic conservative extension that does not introduce new non-logical symbols.
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 realizations of $O_2$ is the model class of realizations of $O_1$.

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.

The opposite is inconsistency.

In many (but not all) logics, consistency of an OMS equivalently can be defined as *false* not being a logical consequence of the OMS. However, this does not work for logics that e.g. do not feature a *false*. See [59] for a more detailed discussion.

**satisfiability** property of an OMS expressing that it is satisfied by at least one realization.

The opposite is unsatisfiability.

Any satisfiable OMS is consistent, but there are some logics where the converse does not hold.

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

**model finder** tool that implements model finding.

**module** structured OMS expressing a sub-OMS that conservatively extends to the whole OMS.

The conservative extension can be either model-theoretic or consequence-theoretic; without qualification, the model-theoretic version is used.

**module extraction** activity of obtaining from an OMS concrete modules to be used for a particular purpose (e.g. to contain a particular sub-signature of the original OMS).

Cited and slightly adapted from [71].

The goal of module extraction is “decomposing an OMS into smaller, more manageable modules with appropriate dependencies” [70].

**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.

Technically, a maximum approximant is a uniform interpolant, see [46].

**approximation** structured OMS that expresses a maximum approximant.

**filtering** structured OMS expressing the verbatim removal of symbols or axioms from an OMS.

If a symbol is removed, all axioms containing that symbol are removed, too.

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

**closure**

**circumscription** structured OMS expressing a variant of the closed world assumption by restricting the realizations to those that are minimal, maximal, free or cofree (with respect to the local environment).

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.

DOL supports four different forms of closure: minimization and maximization as well as freeness and cofreeness (explained below).

See [50], [43].

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

**maximization** form of closure that restricts the model realizations to those that are maximal (with respect to the local environment).
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 XML 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.
7.5 Use Case Onto-4: Interoperability Between Closed-World Data and Open-World Metaddata

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 [5].

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

DOL provides the framework for expressing such bridge rules in a systematic way, maintaining these, and building tools for them.
goes through the two ports bankCom and atmCom linked by a connector. The communication protocol on this connector is captured with a protocol state machine, see Fig. 7.4(c). The protocol state machine fixes in which order the messages verify, verified, reenterPIN, and markInvalid between atm and bank may occur. Figure 7.4(d) provides structural information in form of interfaces specifying what is provided and required at the userCom port and the bankCom port of the atm instance. An interface is a set of operations that other MDE model elements have to implement. In our case, the interfaces are described in a class model. Its component type ATM is further enriched with the OCL constraint trialsNum <= 3, which refines its semantics requiring that trialsNum must not exceed three.

Finally, the dynamic behavior of the atm object is specified by the behavioral state machine shown in Fig. 7.4(e). The machine consists of five states including Idle, CardEntered, etc. Beginning in the initial Idle state, the user can trigger a state change by entering the card. This has the effect that the parameter c from the card event is assigned to the cardId in the atm object (parameter names are not shown on triggers). Entering a PIN triggers another transition to PINEntered. Then the ATM requests verification from the bank using its bankCom port. The transition to Verifying uses a completion event: No explicit trigger is declared and the machine autonomously creates such an event whenever a state is completed, i.e., all internal activities of the state are finished (in our example there are no such activities). If the interaction with the bank results in reenterPIN, and the guard trialsNum < 3 is true, the user can again enter a PIN.

The ATM example in Fig. 7.4 consists of five different UML models, which naturally form a network. Coherence of this network is expressed as its consistency. It is assumed that XMI representations of the relevant UML models have been stored at http://www.example.org/uml/ in a single xmi-file http://www.example.org/uml/atm.xmi that contains a uml:Model element for each UML model whose xmi:id has a prefix xxx followed by an underscore. xxx is determined as follows:

<table>
<thead>
<tr>
<th>Figure</th>
<th>xxx</th>
<th>diagram type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 7.4(a)</td>
<td>sd</td>
<td>sequence diagram</td>
</tr>
<tr>
<td>Fig. 7.4(b)</td>
<td>cmp</td>
<td>composite structure diagram</td>
</tr>
<tr>
<td>Fig. 7.4(c)</td>
<td>psm</td>
<td>protocol state machine</td>
</tr>
<tr>
<td>Fig. 7.4(d)</td>
<td>cd</td>
<td>class diagram</td>
</tr>
<tr>
<td>Fig. 7.4(e)</td>
<td>stm</td>
<td>state machine</td>
</tr>
</tbody>
</table>


library ATM

view cd2stm = cd refined to ( atm hide along atm2cd) 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 = ... end
refinement r = ATM_network refined to Some_refined_ATM_network
entailment e = Some_refined_ATM_network entails ATM_network

Here, abstract_to_concrete_atm is defined in the next section, and atm2cd and psm2cd are suitable logic projections extracting the classes, attributes and operations from a (protocol) state machine, delivering a class model.

7.12 Use Case Model-2: Refinements Between UML Models of Different Types, and Their Reuse

A problem is a lack of reusability of refinements: Consider a controller for an elevator, which is specified with a UML protocol state machine, enriched with UML sequence models and OCL constraints. Assume further that this UML model is not directly implemented, but first refined to a UML behavior state machine (which then can be automatically or semi-automatically transformed into some implementation using standard UML tools). However, there is no standardized language to express, document and maintain the refinement relation itself (UML only allows very simple refinements, namely between
8.6 Semantic Foundations of DOL

A large variety of OMS languages in use can be captured at an abstract level using the concept of institutions [18]. This allows the development of DOL independently of the particularities of a logical system and to use the notions of institution and logical language interchangeably. The main idea is to collect the non-logical symbols of the language in signatures and to assign to each signature the set of sentences that can be formed with its symbols. For each signature, DOL provides means for extracting the symbols it consists of, together with their kind. Institutions also provide a model theory, which introduces semantics for the language and gives a satisfaction relation between the realizations and the sentences of a signature.

It is also possible to complement an institution with a proof theory, introducing a derivability relation between sentences, formalized as an entailment system [51]. In particular, this can be done for all logics that have so far been in use in DOL.

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

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

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

A heterogeneous OMS can import several OMS expressed in different conforming logics, for which suitable translations have been defined in the logic graph provided in annex I or in an extension to it that has been provided when establishing the conformance of some other logic with DOL. Determining the semantics of the heterogeneous OMS requires a translation into a common target language to be applied (cf. clause 10). This translation is determined via a lookup in the transitive closure of the logic graph. Depending on the reasoners available in the given application setting, it can, however, be necessary to employ a different translation. Authors can express which one to employ. However, DOL provides default translations, which are applied unless the user specifies a translation that deviates from the default. Both default and non-default translations may be combined to multi-step translations.
Moreover, annex L informatively introduces Applications, which apply a substitution to an OMS.

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

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

The DOL metamodel for extension OMS is shown in Fig. 9.6. ExtendingOMS is a subclass of OMS, containing those OMS that may be used to extend a given OMS within an ExtensionOMS. An ExtendingOMS can be one of the following:

- a basic OMS BasicOMS written inline, in a conforming serialization of a conforming OMS language (which is defined outside this standard; practically every example uses basic OMS

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

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

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

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

Furthermore, OMS can be constructed using

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

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

\(^{16}\)In this place, any OMS in a conforming serialization of a conforming OMS language is permitted. However, DOL’s module sublanguage should be used instead of the module sublanguage of the respective conforming OMS language; e.g. DOL’s OMS reference and extension construct should be preferred over OWL’s import construct.
equivalence (EquivalenceDefinition, see Annex M.3 for an example), a named declaration of the relation between a module of an OMS and the whole OMS (ConservativeExtensionDefinition, see use case 7.4 for an example), or a named alignment (AlignmentDefinition, see use case 7.3 and Annex M.7 for examples).

The DOL metamodel for interpretations and refinements is shown in Fig. 9.8. Both interpretations and refinements specify a logical entailment or specialization relation between OMS.

An InterpretationDefinition specifies source and target OMS (forming the InterpretationType), as well as a SymbolMap and/or an OMSLanguageTranslation. The SymbolMap in an interpretation always must lead to a signature morphism. A proof obligation expressing that the source OMS, when translated along the signature morphism and/or the OMSLanguageTranslation, logically follows from the target OMS.

A symbol map in an interpretation is required to cover all non-logical symbols of the source OMS; the semantics specification in clause 10 makes this assumption. (Mapping a non-logical symbol twice is an error. Mapping two source non-logical symbols to the same target non-logical symbol is legal, this is a non-injective OMS mapping.)

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

The DOL metamodel for entailments and equivalences is shown in Fig. 9.9. 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 model realizations of the network, when restricted to a given node (given by an IRI), are model realizations of the OMS.

The DOL metamodel for alignments is shown in Fig. 9.10. Signature morphisms used in interpretations and refinements use a functional style of mapping symbols of OMS. In contrast to this style, an alignment provides a relational connection
— a category \[\text{Sig}\] of signatures and signature morphisms,
— a functor \(\text{Sen} : \text{Sig} \to \text{Set}\) giving, for each signature \(\Sigma\), the set of sentences \(\text{Sen}(\Sigma)\), and for each signature morphism \(\sigma : \Sigma \to \Sigma'\), the sentence translation map \(\text{Sen}(\sigma) : \text{Sen}(\Sigma) \to \text{Sen}(\Sigma')\), where often \(\text{Sen}(\sigma)(\phi)\) is written as \(\sigma(\phi)\),
— a functor \(\text{Mod} : \text{Sig}^{op} \to \text{Cat}\) giving, for each signature \(\Sigma\), the category of model-realization\(^{22}\) \(\text{Mod}(\Sigma)\), and for each signature morphism \(\sigma : \Sigma \to \Sigma'\), the reduct functor \(\text{Mod}(\sigma) : \text{Mod}(\Sigma') \to \text{Mod}(\Sigma)\), where often \(\text{Mod}(\sigma)(M)\) is written as \(M'\sigma\), and \(M'\sigma\) is called the \(\sigma\)-reduct of \(M\), while \(M'\) is called a \(\sigma\)-expansion of \(M'\sigma\),
— a satisfaction relation \(\models \subset \text{Mod}(\Sigma) \times \text{Sen}(\Sigma)\) for each \(\Sigma \in \text{Sig}\),

such that for each \(\sigma : \Sigma \to \Sigma'\) in \(\text{Sig}\) the following satisfaction condition holds:

\[
M' \models \sigma(\phi) \iff M'\sigma \models \varphi
\]

for each \(M' \in \text{Mod}(\Sigma')\) and \(\varphi \in \text{Sen}(\Sigma)\), expressing that truth is invariant under change of notation and context. \(\Box\)

**Definition 3 (Propositional Logic)** The signatures of propositional logic are sets \(\Sigma\) of propositional symbols, and signature morphisms are just functions \(\sigma : \Sigma_1 \to \Sigma_2\) between these sets. A \(\Sigma\)-model-realization is a function \(M : \Sigma \to \{\text{True}, \text{False}\}\), and the reduct of a \(\Sigma\)-model-realization \(M_2\) along a signature morphism \(\sigma : \Sigma_1 \to \Sigma_2\) is the \(\Sigma_1\)-model-realization given by the composition of \(\sigma\) with \(M_2\). \(\Sigma\)-sentences are built from the propositional symbols with the usual connectives, and sentence translation is replacing the propositional symbols in \(\Sigma\) 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. \(\Box\)

**Definition 4 (Common Logic — CL)** A common logic signature \(\Sigma\) (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 \(\Sigma\)-model-realization consists of a set \(\text{UR}\), the universe of reference, with a non-empty subset \(\text{UD} \subseteq \text{UR}\), the universe of discourse, and four mappings:

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

A \(\Sigma\)-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 \(\text{UD}^*\), 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 \([20]\).

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\(^{23}\) Model reducts, Reducts leave \(\text{UR}\), \(\text{UD}\), rel and fun untouched, while int and seq are composed with the appropriate signature morphism component. \(\Box\)

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

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

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

\(^{21}\) See \([13]\) for an introduction into category theory.

\(^{22}\) To avoid confusion with models in the sense of model-driven engineering, we use ‘realization’ instead of the commonly used term ‘model’ in logic.

\(^{23}\) That is, a name is a discourse name if and only if its image under the signature morphism is.
Given a theory $T = (\Sigma, \Delta)$, the class of $T$-models-realization is the class of all $\Sigma$-models-realizations $M$ such that $M \models \delta$, for each sentence $\delta \in \Delta$. A theory $(\Sigma, \Delta)$ is consistent if at least one $\Sigma$-model-$(\Sigma, \Delta)$-realization exists. Semantic entailment is defined as usual: for a theory $\Delta \subseteq \text{Sen}(\Sigma, \Delta)$ and $\varphi \in \text{Sen}(\Sigma)$, $\Delta$ entails $\varphi$, written $\Delta \models \varphi$, if all models-realizations satisfying all sentences in $\Delta$ also satisfy $\varphi$. For a theory $(\Sigma, \Delta)$, we write $\Delta^*$ for the set of all $\Sigma$-sentences $\varphi$ such that $\Delta \models \varphi$.

**Definition 6** (Theory morphism) A theory morphism $\phi: (\Sigma, \Delta) \to (\Sigma', \Delta')$ is a signature morphism $\phi: \Sigma \to \Sigma'$ such that $\Delta' \models \phi(\Delta)$.

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

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

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

$$
\begin{array}{ccc}
\text{Sen}'(\Sigma_1) & \xrightarrow{\alpha_{\Sigma_1}} & \text{Sen}'(\Phi(\Sigma_1)) \\
\text{Sen}'(\sigma) \downarrow & & \downarrow \text{Sen}'(\Phi(\sigma)) \\
\text{Sen}'(\Sigma_2) & \xrightarrow{\alpha_{\Sigma_2}} & \text{Sen}'(\Phi(\Sigma_2))
\end{array}
\quad
\begin{array}{ccc}
\text{Mod}'(\Phi(\Sigma_2)) & \xrightarrow{\beta_{\Sigma_2}} & \text{Mod}'(\Sigma_2) \\
\text{Mod}'(\Phi(\sigma)) \downarrow & & \downarrow \text{Mod}'(\sigma) \\
\text{Mod}'(\Phi(\Sigma_1)) & \xrightarrow{\beta_{\Sigma_1}} & \text{Mod}'(\Sigma_1)
\end{array}
$$

A comorphism is:

- **faithful** if logical consequence is preserved and reflected along the comorphism:
  $$\Gamma \models I \varphi \iff \alpha(\Gamma) \models I \alpha(\varphi)$$
- **model-expansive** if each $\beta_\Sigma$ is surjective;
- **(weakly) exact** if for each signature morphism $\sigma: \Sigma_1 \to \Sigma_2$, the naturality diagram
  $$
  \begin{array}{ccc}
  \text{Mod}'(\Phi(\Sigma_2)) & \xrightarrow{\beta_{\Sigma_2}} & \text{Mod}'(\Sigma_2) \\
  \text{Mod}'(\Phi(\sigma)) \downarrow & & \downarrow \text{Mod}'(\sigma) \\
  \text{Mod}'(\Phi(\Sigma_1)) & \xrightarrow{\beta_{\Sigma_1}} & \text{Mod}'(\Sigma_1)
  \end{array}
  $$
  admits (weak) amalgamation, i.e. any for any two model-realizations $M_2 \in |\text{Mod}'(\Sigma_2)|$ and $M'_2 \in |\text{Mod}'(\Phi(\Sigma_1))|$ with $M_2|_{\phi} = \beta_{\Sigma_2}(M'_2)$, there is a unique (not necessarily unique) $M'_2 \in |\text{Mod}'(\Phi(\Sigma_2))|$ with $\beta_{\Sigma_2}(M'_2) = M_2$ and $M'_2|_{\phi} = M'_2$;
- **a subinstitution comorphism** if $\Phi$ is an embedding, each $\alpha_\Sigma$ is injective and each $\beta_\Sigma$ is bijective [24];
- **an inclusion comorphism** if $\Phi$ and each $\alpha_\Sigma$ are inclusions, and each $\beta_\Sigma$ 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 $\alpha(\Gamma) \models I \alpha(\varphi)$. Subinstitution comorphism preserve the semantics of more advanced DOL structuring constructs such as OMS translation and OMS reduction.

---

24) An isomorphism if model-morphisms of realizations are taken into account.
Definition 8 Given an institution \( I = (\text{Sig}, \text{Mod}, \text{Sen}, |=) \), the institution of its theories, denoted \( 1^\text{th} \), can be defined as follows. The category of signatures of \( 1^\text{th} \) is the category of I-theories and I-theory morphisms, denoted \( \text{Th}^I \). For each theory \((\Sigma, \Delta)\), its sentences are just \( \Sigma \)-sentences in \( I \), and its \textit{models-realizations} are just \( \Sigma \)-models-realizations in \( I \) that satisfy the sentences in \( \Delta \), while the \((\Sigma, \Delta)\)-satisfaction is the \( \Sigma \)-satisfaction of sentences in \textit{models-realizations} of \( I \).

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

Definition 9 Let \( I = (\text{Sig}^I, \text{Mod}^I, \text{Sen}^I, |=^I) \) and \( J = (\text{Sig}^J, \text{Mod}^J, \text{Sen}^J, |=^J) \) be two institutions. A \textit{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 \textit{institution morphism} from an institution \( I = (\text{Sig}^I, \text{Mod}^I, \text{Sen}^I, |=^I) \) to an institution \( J = (\text{Sig}^J, \text{Mod}^J, \text{Sen}^J, |=^J) \) consists of a functor \( \Phi : \text{Sig}^I \rightarrow \text{Sig}^J \), and two natural transformations \( \alpha : \text{Mod}^I \circ \Phi \Rightarrow \text{Mod}^J \) and \( \beta : \text{Sen}^I \circ \Phi \Rightarrow \text{Sen}^J \), such that for each \( I \)-signature \( \Sigma \), each sentence \( \varphi \in \text{Sen}^J(\Phi(\Sigma)) \) and each \textit{model-realization} \( M \in \text{Mod}^I(\Sigma) \)

\[
M |=^I_{\Sigma} \alpha(\varphi) \iff \beta(M) |=^J_{\Phi(\Sigma)} \varphi.
\]

holds, called the satisfaction condition.

Colimits are a categorical concept providing ways 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 \[13\]. For a formal mathematical definition, see \[P.1.1\]

A major property of colimits of specifications is amalgamation (also related to ‘exactness’ \[14\]). It can be intuitively explained as stating that \textit{models-realizations} of given specifications can be combined to yield a uniquely determined \textit{model-realization} of a colimit specification, provided that the original \textit{models-realizations} coincide on common components. Amalgamation is a common technical assumption in the study of specification semantics \[67\].

In the following, fix an arbitrary institution \( I = (\text{Sig}, \text{Sen}, \text{Mod}, |=) \).

Definition 11 Given a network \( D : J \rightarrow \text{Sig}^I \), a \textit{family of models-realizations} \( M = \{M_p\}_{p \in |J|} \) is consistent with \( D \) (or sometimes compatible with \( D \)) if for each node \( p \) of \( D \), \( M_p \in \text{Mod}(D(p)) \) and for each edge \( e : p \rightarrow q, M_p = M_q|_{D(e)} \). A \textit{cocone} \((\Sigma, (M_p)_{p \in |J|})\) over the network \( D : J \rightarrow \text{Sig}^I \) is called weakly amalgamable if it is mapped to a weak limit by \textit{Mod}. For \textit{models-realizations}, this means that for each \( D \)-compatible family of \textit{models-realizations} \( (M_p)_{p \in |J|} \), there is a \( \Sigma \)-\textit{model-realization} \( M \), called an amalgamation of \((M_p)_{p \in |J|}\), with \( M|_{\Sigma_j} = M_p \) \((j \in |J|)\), and similarly for \textit{model-morphisms of realizations}. If this \textit{model-realization} is unique, the cocone is called amalgamable. \( I \) (or \textit{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.

\[R\] 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 \textit{models-realizations} compatible with the network. This allows us to define a function \textit{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 \textit{models-realizations} compatible with a network and returning its selected amalgamation.

10.3 Semantics of DOL Language Constructs

The semantics of DOL is based on a fixed (but in principle arbitrary) heterogeneous logical environment. The semantic domains are based on this heterogeneous logical environment. A specific heterogeneous logical environment is given in the annexes.
A heterogeneous logical environment is given by a collection of OMS languages and OMS language translations\(^{25}\), a collection of institutions, institution morphisms and institution comorphisms (serving as logics, logic reductions and logic translations), and a collection of serializations. Moreover, some of the institution comorphisms are marked as default translations (but only at most one between a given source and target institution), and there is a binary relation \textit{supports} between OMS languages and institutions, and a binary relation \textit{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 \(n\) \(\)in the heterogeneous logical environment \(\Gamma\).

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

**Definition 12** An inclusive category \(^{26}\) is a category having a broad subcategory \(^{26}\) 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. \(\square\)

A category \textit{has pushouts which preserve inclusions} iff there exists a pushout

\[
\begin{array}{ccc}
A & \xrightarrow{\eta} & A' \\
\downarrow & & \downarrow \\
B' & \xrightarrow{\eta'} & B'
\end{array}
\]

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 \textit{weakly inclusive} if

- \(\text{Sig}\) is inclusive and has pushouts which preserve inclusions,
- \(\text{Sen}\) is inclusive, and
- each \textit{model category} \textit{category of realizations} has a broad subcategory of inclusions. \(\square\)

Let \(I\) be a weakly inclusive institution. \(I\) \textit{has differences}, if there is a binary operation \(\setminus\) on signatures, such that for each pair of signatures \(\Sigma_1, \Sigma_2\), the greatest signature \(\Sigma\) such that

1) \(\Sigma \subseteq \Sigma_1\)
2) \(\Sigma \cap \Sigma_2 = \emptyset\)

exists and is equal to \(\Sigma_1 \setminus \Sigma_2\).

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

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

**Definition 14** An inclusive category with symbols is an inclusive category \(C\) equipped with a faithful functor \(|\_|: C \rightarrow \text{Set}\)\(^{27}\) that preserves inclusions. \(\square\)

---

\(^{25}\)The terms \textit{OMS language} and \textit{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.

\(^{26}\)That is, with the same objects as the original category.

\(^{27}\)That is, \((C, |\_|)\) is a concrete category.
Moreover, if \( \sigma : \Sigma \rightarrow \Sigma' \) is a signature morphism, it uniquely determines a map \(|\sigma| : |\Sigma| \rightarrow |\Sigma'|\).

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 of realizations \( \mathcal{M}_\emptyset \) and such that there exists a unique signature morphism from \( \emptyset \) to any signature of the institution. Moreover, the existence of a partial union operation on institutions is required, denoted \( \bigcup : L_1 \bigcup L_2 = (L, \rho_1 : L_1 \rightarrow L, \rho_2 : L_2 \rightarrow L) \), when defined, where \( L \) is an institution and \( \rho_1 \) and \( \rho_2 \) 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 \( I \) 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
  \[
  \text{semBasic}_{\text{lang}, \text{logic}, \text{ser}}(\Sigma, O) = (\Sigma', \Delta')
  \]
  where \( O \) is a BasicOMS, \( \Sigma \) gives the context of previous declarations, \( \Sigma' \) is the resulting signature and \( \Delta' \) is the resulting set of sentences. It is required then that \( \Sigma \subseteq \Sigma' \).
- a function \( \text{makeMorphism}_2 \) that takes as arguments two signatures \( \Sigma_1 \) and \( \Sigma_2 \) of \( I \) and returns as result the list of all pairs of symbols \( (s^1_i, s^2_i) \) such that \( s^1_i \in |\Sigma_1| \) and \( s^2_i \in |\Sigma_2| \) and the symbols have the same name. The relation represented by \( \text{sameName}_2(\Sigma_1, \Sigma_2) \) must be an equivalence relation.
- a relativization function \( \text{relativize}_2 \) taking as argument a theory and giving as result a theory, and a function \( \text{theoryOfCorrespondences} \) for translating correspondences of alignments into sentences in the logic according to a given assumption about the semantics of the alignment, both needed in Section 10.3.3.

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 model-class-of-realizations over some signature of an institution. This is called model-level semantics. In some cases, but not in all, one can also define a theory-level semantics of an OMS as a set of sentences over some signature of an institution. The two semantics are related by the fact that, when both the model-level and the theory-level semantics of an OMS are defined, they are compatible in the sense that the class of model-class-of-realizations given by the model-level semantics is exactly the model-class-of-realizations of the theory given by the theory-level semantics.

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

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

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

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

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:

- \( \text{Name}(N) \), the name of the node
- \( \text{Inst}(N) \), the institution of the node
In this section the semantics of Documents constructs regarding documents and DOL libraries is defined.

\[
\text{sem}(\text{Document}) = \Gamma \\
: \text{LogicalEnvironment}
\]

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 \( \text{nativeDocument} \),

\[
\text{sem}(\text{nativeDocument}) = \Gamma''
\]
10.3.2 Semantics of Networks

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

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

The following auxiliary functions are used:

- $\text{insert}(G, \Gamma, \text{iri}, \text{id})$, where $G$ is a graph, $\Gamma$ is a global environment, $\text{iri}$ is an IRI and $\text{id}$ is an Id, defined as follows:
  - if $\text{iri}$ denotes an OMS in $\Gamma$, then a new node named $\text{iri}$ and labeled with $\Gamma(\text{iri})$ and with $\text{id}$ is added to $G$, unless a node named $\text{iri}$ already exists in $G$, and in this case $G$ is left unchanged,
  - if $\text{iri}$ denotes an OMS mapping or a network in $\Gamma$, the result is the union of $G$ with the graph of $\Gamma(\text{iri})$.
- $\text{removeElement}(\Gamma, G, \text{anIRI})$, where $G$ is a graph, $\Gamma$ is a global environment and $\text{anIRI}$ is an IRI, defined as follows:
  - if $\text{anIRI}$ denotes an OMS in $\Gamma$, then the node labeled with $\text{anIRI}$ and all its incoming and outgoing edges are removed from $G$.
  - if $\text{anIRI}$ denotes an OMS mapping in $\Gamma$, then $\Gamma(\text{anIRI})$ 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 $\text{anIRI}$ is a network in $\Gamma$, then all the nodes of its graph and all their incoming and outgoing edges are removed from $G$.
- $\text{removePaths}(\Gamma, G, \text{iri}_1, \text{iri}_2)$, where $G$ is a graph, $\Gamma$ is a global environment and $\text{iri}_1, \text{iri}_2$ are IRIs, whose result is that all paths of imports in $G$ between the nodes labeled with $\text{iri}_1$ and $\text{iri}_2$ are removed from $G$.

Finally, the operation $\text{addImports}(\Gamma, G, [\text{iri}_1, \ldots, \text{iri}_n])$ adds to $G$ all import edges in $\Gamma.imports$ between nodes which appear in the subgraph determined by $\Gamma(\text{iri}_1), \ldots, \Gamma(\text{iri}_n)$.

10.3.2.1 Semantics of network definitions

\[
\text{sem}(\Gamma, \text{NetworkDefinition}) = \Gamma' : \text{LogicalEnvironment}
\]

If $n$ is a NetworkDefinition, \[\text{sem}(\Gamma, n) = \Gamma'\]

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

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

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

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

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

### 10.3.2.2 Semantics of networks

\[
\text{sem}(\Gamma,\text{Network}) = G : \textit{OMSGraph}
\]

If \( n \) is a network, \( \text{sem}(\Gamma,n) = G' \)

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

### 10.3.2.3 Semantics of sets of network elements

\[
\text{sem}(\Gamma,\text{Set}(\text{NetworkElement})) = G : \textit{OMSGraph}
\]

If \( \text{elem}_1,\ldots,\text{elem}_n \) are all \( \text{NetworkElements} \),

\[
\text{sem}(\Gamma,\text{Set}\{\text{elem}_1,\ldots,\text{elem}_n\}) = G
\]

where

\[
\begin{align*}
G_1 &= \text{sem}(\Gamma,G_0,\text{elem}_1), \text{where } G_0 \text{ is the empty graph}, \\
G_2 &= \text{sem}(\Gamma,G_1,\text{elem}_2) \\
& \vdots \\
G_n &= \text{sem}(\Gamma,G_{n-1},\text{elem}_n), \\
G &= \text{addImports}(\Gamma,G_n,\{\text{elem}_1,\ldots,\text{elem}_n\}).
\end{align*}
\]

### 10.3.2.4 Semantics of network elements

\[
\text{sem}(\Gamma,G,\text{NetworkElement}) = G' : \textit{OMSGraph}
\]

If \( \text{networkElement} \) is a \( \text{NetworkElement} \),

\[
\text{sem}(\Gamma,G,\text{networkElement}) = \text{insert}(G,\Gamma,\text{networkElement.\elementRef.iri,\text{networkElement.id}})
\]

### 10.3.2.5 Semantics of sets of excluded elements

\[
\text{sem}(\Gamma,G,\text{Set}(\text{ExcludedElement})) = G' : \textit{OMSGraph}
\]
10.3.3.5 Semantics of ExtendingOMS

\[
\text{sem}(\Gamma, \text{ExtendingOMS}) = (\Gamma', (\mathcal{I}, \mathcal{M}, \Delta))
\]

: (LogicalEnvironment, (Institution, Signature, ModelClass, Sentences))

The semantics for ClosableOMS has been defined above.

If \( O \) is a RelativeClosureOMS, \( \text{O.closureType} = \text{minimize} \) and \( O' = \text{O.closableOMS} \), then

\[
\text{sem}(\Gamma, O) = (\Gamma', (\mathcal{I}, \mathcal{M}, \Delta))
\]

where

\begin{itemize}
  \item \( \mathcal{I} = \text{Inst}(O') \)
  \item \( \mathcal{M} = \{ M \in \text{Mod}(O') \mid M \text{ is minimal in } \text{Mod}(O') \} \) and “minimal” is interpreted in the pre-order defined by \( M_1 \leq M_2 \) if there is a model homomorphism \( \text{homomorphism of realizations} \) \( M_1 \to M_2 \).
  \item \( \Delta = \bot \)
  \item \( \Gamma' \) is obtained from \( \Gamma'' = \text{Env}(\Gamma, O') \) by adding to \( \Gamma'' \). \text{imports} a new node labeled with (Name\((O), \text{Inst}(O), \text{Sig}(O), \text{Mod}(O), \text{Th}(O)\)) and an edge from the node of \( O' \) to the node of \( O \) labeled with the identity morphism on \( \text{Sig}(O') \).
\end{itemize}

The semantics of \( O \) is defined similarly for the other three alternatives of \( \text{O.closureType} \), only the model-class \text{class of realizations} differs:

\begin{itemize}
  \item if \( \text{O.closureType} = \text{maximize} \), \( \mathcal{M} = \{ M \in \text{Mod}(O') \mid M \text{ is maximal in } \text{Mod}(O') \} \)
  \item if \( \text{O.closureType} = \text{free} \), \( \mathcal{M} = \{ M \in \text{Mod}(O') \mid M \text{ is initial in } \text{Mod}(O') \} \)
  \item if \( \text{O.closureType} = \text{cofree} \), \( \mathcal{M} = \{ M \in \text{Mod}(O') \mid M \text{ is terminal in } \text{Mod}(O') \} \)
\end{itemize}

Here, initial and terminal \text{model-realizations} are defined as in category theory, see [P.1.1]

10.3.3.6 Semantics of ExtendingOMS in a local environment

\[
\text{sem}(\Gamma, (\mathcal{I}, \mathcal{M}, \Delta), \text{ExtendingOMS})) = (\Gamma', (\mathcal{I}', \mathcal{M}', \Delta'))
\]

: (LogicalEnvironment, (Institution, Signature, ModelClass, Sentences))

The semantics for ClosableOMS has been defined above.

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

Formally, if \( O \) is a RelativeClosureOMS, \( \text{O.closureType} = \text{minimize} \) and \( O' = \text{O.closableOMS} \), and

\[
\text{sem}(\Gamma, (\mathcal{I}, \mathcal{M}, \Delta), O') = (\Gamma', (\mathcal{I}', \mathcal{M}', \Delta'))
\]

then

\[
\text{sem}(\Gamma, (\mathcal{I}, \mathcal{M}, \Delta), O)) = (\Gamma'', (\mathcal{I}'', \mathcal{M}'', \Delta''))
\]

where

\begin{itemize}
  \item \( \mathcal{I}'' = \mathcal{I}' \)
  \item \( \mathcal{M}'' = \mathcal{M}' \)
  \item \( \Delta'' = \bot \)
  \item \( \mathcal{M}'' = \{ M \in \mathcal{M} \mid M \text{ is minimal in } \{ M' \in \mathcal{M} \mid M'_{\sigma} = M_{|\Sigma} \} \} \) and “minimal” is interpreted in the pre-order defined by \( M_1 \leq M_2 \) if there is a model homomorphism \( \text{homomorphism of realizations} \) \( M_1 \to M_2 \).
\end{itemize}
— $\Gamma''$ is obtained from $\Gamma' = \text{Env}(\Gamma, O.\text{oms})$ by extending $\Gamma'.\text{imports}$ with a new node for $O$ labeled as in the items above and with a new edge from the node of $O.\text{oms}$ to the node of $O$ labeled with $((\Phi, \alpha, \beta), \sigma)$.

The semantics of a ReductionOMS $O$ is

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

where

— $\mathcal{I} = J, \Sigma = \Sigma', \Delta = \Delta'$
— $\mathcal{M}$ is the class of $\Delta$-models—realizations
— $\Gamma''$ is obtained from $\Gamma' = \text{Env}(\Gamma, O.\text{oms})$ by extending $\Gamma'.\text{imports}$ 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 $O.\text{oms}$ labeled with $((\Phi, \alpha, \beta), \sigma)$.

The semantics of an ExtractionOMS $O$ is

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

where

— $\mathcal{I} = \text{Inst}(O.\text{oms})$
— $\Sigma = \Sigma'$
— $\Delta = \Delta'$
— $\mathcal{M}$ is the class of $\Delta$-models—realizations
— $\Gamma''$ is obtained from $\Gamma' = \text{Env}(\Gamma, O.\text{oms})$ by extending $\Gamma'.\text{imports}$ 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 $O.\text{oms}$ labeled with the inclusion of $\Sigma'$ in $\text{Sig}(O.\text{oms})$

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

The semantics of an ApproximationOMS $O$ is

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

where

— $\mathcal{I}' = \mathcal{I}'$
— $\Sigma' = \Phi(\Sigma)$
— $\Delta = \alpha_{\text{Sig}(O.\text{oms})}^{-1}(\Theta(\Sigma)) \cap \text{Sen}_{\mathcal{I}'}(\text{Sig}(O.\text{oms}))$. i.e. that part of $\text{Th}(O.\text{oms})$ that can be expressed in the smaller signature and logic. In practice, one looks for a finite subset that still is logically equivalent to this set.
— $\mathcal{M}$ is the class of $\Delta$-models—realizations
— $\Gamma''$ is obtained from $\Gamma' = \text{Env}(\Gamma, O.\text{oms})$ by extending $\Gamma'.\text{imports}$ with a new node for $O$ labeled as in the items above and with a new edge from the node of $O.\text{oms}$ to the node of $O$ labeled with $(\rho, \iota: \Phi(\Sigma) \rightarrow \text{Sig}(O.\text{oms}))$

where $(\rho = (\Phi, \alpha, \beta): \mathcal{I} \rightarrow \mathcal{I}', \Sigma) = \text{sem}(\Gamma, (\text{Inst}(O.\text{oms}), \text{Sig}(O.\text{oms})), O.\text{approximation})$.

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

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

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

where

— $\mathcal{I}'' = \mathcal{I}'$
— $\Sigma''$ is the smallest signature with $\Sigma' \subseteq \Sigma''$ and $\Delta' \subseteq \text{Sen}(\Sigma'')$. (If this smallest signature does not exist, the semantics is undefined.)
— $\Delta'' = (\Delta \cap \text{Sen}(\Sigma'')) \cup \Delta'$
— $\text{Mod}(O)$ is the class of all $\Delta$-models—realizations.
— $\Gamma''$ is obtained from $\Gamma'$ by extending $\Gamma'.\text{imports}$ 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 $O.\text{oms}$ labeled with the inclusion of $\Sigma''$ in $\Sigma$.

If $c = \text{remove}$, the semantics of $O$ is

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

where

— $cI'' = I'$
— $\Sigma'' = \Sigma \setminus \Sigma'$
— $\Delta'' = \Delta \cap \text{Sen}(\Sigma'') \setminus \Delta'$
— $\mathcal{M}''$ is the class of all $\text{Th}(O)$-models—realizations.
— $\Gamma''$ is obtained from $\Gamma'$ by extending $\Gamma'.\text{imports}$ 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 $O.\text{oms}$ labeled with the inclusion of $\Sigma''$ in $\Sigma$.

The semantics of an $\text{UnionOMS}$ $O$ is

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

where

— $cI' = I$ where $\text{Inst}(O_1) \cup \text{Inst}(O_2) = (\mathcal{I}, (\Phi_1, \alpha_1, \beta_1) : \text{Inst}(O_1) \rightarrow \mathcal{I}, (\Phi_2, \alpha_2, \beta_2) : \text{Inst}(O_2) \rightarrow \mathcal{I})$
— $\Sigma' = \Phi_1(\text{Sig}(O_1)) \cup \Phi_2(\text{Sig}(O_2))$
— $\mathcal{M}' = \{ M \in \text{Mod}(\Sigma') \mid \beta_i \cdot (M |_{\text{Sig}(O_i)}) \in \text{Mod}(O_i), \text{ for } i = 1, 2 \}$
— $\Delta' = \alpha_1(\text{Th}(O_1)) \cup \alpha_2(\text{Th}(O_2))$.
— $\Gamma''$ is obtained from $\Gamma' = \text{Env}(\text{Env}(\Gamma, O_1), O_2)$ by extending $\Gamma'.\text{imports}$ with a new node for $O$ labeled as in the items above and with edges from the nodes of $O_1$ and $O_2$, respectively, to the node of $O$, labeled for each $i = 1, 2$ with $(\Phi_i, \alpha_i, \beta_i, \iota_i : \Phi_i(O_i) \rightarrow \Sigma')$.

where $O_1 = O.\text{oms}$ and $O_2 = O.\text{oms}2$.

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

The semantics of an $\text{ExtensionOMS}$ $O$ is

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

where

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

The semantics of a $\text{QualifiedOMS}$ $O$ in the context $\Gamma$ is the same as the semantics of $O.\text{oms}$ in the context $\Gamma'$ given by the semantics of $O.\text{qualification}$ in the context $\Gamma$. The change of context is local to $O.\text{oms}$, which means that if the qualification appears as a term in a larger expression, after its analysis the context will be $\Gamma$ and not $\Gamma'$. Formally,

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

Distributed Ontology, Model, and Specification Language (DOL), v1.0 Beta
10.3.3.16 Semantics of approximations

\[
sem(\Gamma, (I, \Sigma), \text{Approximation}) = (\rho: I \rightarrow I', \Sigma')
\]

: (Morphism, Signature)

If \(a\) is an Approximation, 
\[
sem(\Gamma, (I, \Sigma), a) = (\rho, \Sigma')
\]

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

10.3.3.17 Semantics of filtering

\[
sem(\Gamma, (I, \Sigma, M, \Delta), \text{Filtering}) = (c, I', \Sigma', \Delta')
\]

: ('keep'|'remove', Institution, Signature, Sentences)

If \(f\) is a Filtering such that \(f.\text{removalKind} = \text{keep}\), 
\[
sem(\Gamma, (I, \Sigma, M, \Delta), f) = (\text{keep}, I', \Sigma', \Delta')
\]

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

If \(f\) is a Filtering such that \(f.\text{removalKind} = \text{remove}\), 
\[
sem(\Gamma, (I, \Sigma, M, \Delta), f) = (\text{remove}, I', \Sigma', \Delta')
\]

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

\[
sem(\Gamma, (I, \Sigma, M, \Delta), \text{BasicOMSOrSymbolList}) = (I', \Sigma', M', \Delta')
\]

: (Institution, Signature, Sentences)

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

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

10.3.3.18 Semantics of extension

\[
sem(\Gamma, (I, \Sigma, M, \Delta), \text{Extension}) = (\Gamma', (I, \Sigma', M', \Delta'))
\]

: (LogicalEnvironment, (Institution, Signature, ModelClass, Sentences))

If \(e\) is an Extension, 
\[
sem(\Gamma, (I, \Sigma, M, \Delta), e) = (\Gamma', (I, \Sigma', M', \Delta'))
\]

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

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

If \(e.\text{extensionName}\) is present, the inclusion link is labeled with this name.
10.3.3.19 Semantics of interface signatures

\[
\text{sem}(\Gamma, \Sigma, \text{RemovalKind}, \text{InterfaceSignature}) = \Sigma'
\]

If \( r \) is a RemovalKind and \( s \) is an InterfaceSignature,

\[
\text{sem}(\Gamma, \Sigma, r, s) = \Sigma'
\]

where

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

10.3.3.20 Semantics of OMS definitions

\[
\text{sem}(\Gamma, \text{OMSDefinition}) = \Gamma'
\]

An OMSDefinition \( O \) extends the global environment.

\[
\text{sem}(\Gamma, O) = \Gamma''
\]

where for each of the institutions \( I_1, \ldots, I_n \) supported by \( \Gamma.\text{lang} \), we have

\[
\Gamma_1 = \Gamma_1[\text{postfixLogicIRI}(O.\text{omsName}, I_1) \mapsto (I_1, \Sigma_1, M_1, \Delta_1)]
\]

where \( \text{sem}(\Gamma_1.\text{logic} = I_1, O.\text{oms}) = (\Gamma_1', (I_1, \Sigma_1, M_1, \Delta_1)) \),

\[
\Gamma_2 = \Gamma_2[\text{postfixLogicIRI}(O.\text{omsName}, I_2) \mapsto (I_2, \Sigma_2, M_2, \Delta_2)]
\]

where \( \text{sem}(\Gamma_2.\text{logic} = I_2, O.\text{oms}) = (\Gamma_2', (I_2, \Sigma_2, M_2, \Delta_2)) \), \ldots,

\[
\Gamma'' = \Gamma''_n[\text{postfixLogicIRI}(O.\text{omsName}, I_n) \mapsto (I_n, \Sigma_n, M_n, \Delta_n)]
\]

where \( \text{sem}(\Gamma_{n-1}.\text{logic} = I_n, O.\text{oms}) = (\Gamma''_n, (I_n, \Sigma_n, M_n, \Delta_n)) \)

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

If \( O.\text{conservativityStrength} \) is model-conservative, the semantics is only defined if \( M \neq \emptyset \).

If \( O.\text{conservativityStrength} \) is consequence-conservative, the semantics is only defined if \( \Delta \) has only tautologies\(^{28}\) as signature-free\(^{29}\) logical consequences.

If \( O.\text{conservativityStrength} \) is monomorphic, the semantics is only defined if \( M \) consist of exactly one isomorphism class of model\(\text{-}realizations\).\(^{20}\)

If \( O.\text{conservativityStrength} \) is weak-definitional, the semantics is only defined if \( M \) is empty or a singleton.

If \( O.\text{conservativityStrength} \) is definitional, the semantics is only defined if \( M \) is a singleton.

10.3.3.21 Semantics of OMS references

\[
\text{sem}(\Gamma, \text{OMSReference}) = (\Gamma', (I, \Sigma, M, \Delta))
\]

: \((\text{LogicalEnvironment}, (\text{Institution, Signature, ModelClass, Sentences}))\)

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

\(^{28}\)A tautology is a sentence holding in every model.

\(^{29}\)A signature-free sentence is one over the empty signature.
10.3.4 Semantics of OMS Mappings

10.3.4.1 Semantics of mapping definitions

\[ \text{sem}(\Gamma, \text{MappingDefinition}) = \Gamma' : \text{LogicalEnvironment} \]

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

10.3.4.2 Semantics of interpretation definitions

\[ \text{sem}(\Gamma, \text{InterpretationDefinition}) = \Gamma' : \text{LogicalEnvironment} \]

If \( d \) is an InterpretationDefinition, \( \text{sem}(\Gamma, d) = \Gamma' \)

where \( \Gamma' = \Gamma[\text{d.interpretationName} \rightarrow \text{sem}(\Gamma, \text{d.interpretationType})] \)

and \( G \) is the graph \( L_1 \xrightarrow{(\rho, \sigma)} L_2 \) where

- \( (L_1, L_2) = \text{sem}(\Gamma, \text{d.interpretationType}) \)
- \( \rho = (\Phi, \alpha, \beta) : \text{Inst}(L_1) \to \text{Inst}(L_2) \) is the comorphism given by \( \text{sem}(\Gamma, \text{d.omsLanguageTranslation}) \).
  - If \( \text{d.omsLanguageTranslation} \) is missing, the default translations between the logics is selected.
- \( \sigma = \text{sem}(\Gamma, \{ \text{current} = (\text{lang, logic', ser}) \}, \Phi(\text{Sig}(L_1)), \text{Sig}(L_2), \text{d.symbolMap}) \), where \( \Gamma \).current = (lang, logic, ser) and logic' is the target logic of \( \rho \), or, if \( \text{d.symbolMap} \) is missing, \( \sigma \) is the identity signature morphism on \( \Phi(\text{Sig}(L_1)) \) which must be equal with \( \text{Sig}(L_2) \).

The semantics is only defined if \( \beta_{\text{Sig}(L_1)}(M_2|\sigma) \in \text{Mod}(L_1) \) for each \( M_2 \in \text{Mod}(L_2) \).

If the optional argument \( \text{d.conservativityStrength} \) is

- model-conservative, for each model-realization \( M_1 \in |\text{Mod}(L_1)| \) there exist a model-realization \( M_2 \in |\text{Mod}(L_2)| \) such that \( \beta_{\text{Sig}(L_1)}(M_2|\sigma) = M_1 \).
- consequence-conservative, for each \( \text{Sig}(L_1) \)-sentence \( \varphi \), if \( M_2 \models \sigma(\alpha_{\text{Sig}(L_1)}(\varphi)) \) then \( M_1 \models \varphi \).
- not-model-conservative, there must exist a model-realization \( M_2 \in |\text{Mod}(L_1)| \) such that there is no model realization \( M_2 \in |\text{Mod}(L_2)| \) such that \( \beta_{\text{Sig}(L_1)}(M_2|\sigma) = M_1 \).
- not-consequence-conservative, there is a \( \text{Sig}(L_1) \)-sentence \( \varphi \), such that \( M_2 \models \sigma(\alpha_{\text{Sig}(L_1)}(\varphi)) \) and \( M_1 \not\models \varphi \).

10.3.4.3 Semantics of refinement definitions

\[ \text{sem}(\Gamma, \text{RefinementDefinition}) = \Gamma' : \text{LogicalEnvironment} \]

If \( d \) is a RefinementDefinition, \( \text{sem}(\Gamma, d) = \Gamma' \)

where \( \Gamma' = \Gamma[\text{d.interpretationName} \rightarrow \text{sem}(\Gamma, \text{d.refinement})] \).
10.3.4.4 Semantics of interpretation types

\[
\text{sem}(\Gamma, \text{InterpretationType}) = (((N_1, I_1, \Sigma_1, M_1, \Delta_1), (N_2, I_2, \Sigma_2, M_2, \Delta_2)) : (\text{NodeLabel}, \text{NodeLabel})
\]

If \( t \) is an InterpretationType, \( \text{sem}(\Gamma, t) = (L_1, L_2) \)
where

- \( \text{Name}(L_1) = \text{Name}(t.\text{source}) \) and \( \text{Name}(L_2) = \text{Name}(t.\text{target}) \),
- \( (\text{Inst}(L_1), \text{Sig}(L_1), \text{Mod}(L_1), \text{Th}(L_1)) = \text{sem}(\Gamma, t.\text{source}) \),
- \( (\text{Inst}(L_2), \text{Sig}(L_2), \text{Mod}(L_2), \text{Th}(L_2)) = \text{sem}(\Gamma, t.\text{target}) \).

10.3.4.5 Semantics of refinements

\[
\text{sem}(\Gamma, \text{Refinement}) = ((G_1, G_2, \sigma, \mathcal{M}) : (\text{OMSGraph}, \text{OMSGraph}, \text{GraphMorphism}, \text{ModelClass})
\]

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 \( \sigma : G_1 \rightarrow G_2 \) is

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

2) a natural transformation \( \sigma^M : G_1 \rightarrow \sigma^G; G_2 \)

such that

1) for each node \( N_1 \) in \( G_1 \) labeled with \( (I_1, \Sigma_1, M_1) \) such that \( \sigma^G(N_1) \) is a node \( N_2 \) labeled with \( (I_2, \Sigma_2, M_2) \) in \( G_2 \), there is a signature morphism \( (\rho^G_{N_1}, \sigma^G_{N_1}) : (I_1, \Sigma_1) \rightarrow (I_2, \Sigma_2) \), where

2) \( \rho^G_{N_1} = (\Phi, \alpha, \beta) : I_1 \rightarrow I_2 \) is an institution comorphism between the logics of the two nodes and \( \sigma^M_{N_1} : \Phi(\Sigma_1) \rightarrow \Sigma_2 \) is a signature morphism, such that \( \beta_{\Sigma_1}(M_2\vert_{N_1}) \in M_1 \) for each \( M_2 \in M_2 \).

A refinement \text{model-realization} is a class \( \mathcal{M} \) of pairs of families of \text{model-realizations} compatible with the two networks. Given a network morphism \( \sigma : G_1 \rightarrow G_2 \) and a \( G_2 \text{ model-realization} F, F|_\sigma \) is defined as the family of \text{model-realizations} \( \{M_i\}_{i \in \text{Nodes}(G_1)} \) such that \( M_i = F_{\sigma G_i}|_{\sigma i} \) for each \( i \in \text{Nodes}(G_1) \).

Thus, the semantics of a \text{Refinement} consists of

- a refinement signature \( (G_1, G_2) \),
- a network morphism \( \sigma \) and
- a refinement \text{model-realization} \( \mathcal{M} \).

If \( r \) is \text{RefinementOMS}, \( \text{sem}(\Gamma, r) = ((G, G), \sigma, \mathcal{M}) \)
where

- \( G \) is a graph with just one isolated node \( N \) such that \( \text{Name}(N) = \text{Name}(r.\text{oms}) \) and the other elements of the tuple labeling \( N \) are given by \( \text{sem}(\Gamma, r.\text{oms}) \),
— \( \sigma \) is the identity morphism on \( \text{Sig}(r\text{oms}) \),
— \( \mathcal{M} = \{(M, (M)) \mid M \in \text{Mod}(r\text{oms})\} \), where \( (M) \) is the singleton family consisting of \( M \).

If \( r \) is \text{RefinementNetwork},

\[
\text{sem}(\Gamma, r) = ((G, G), \sigma, \mathcal{M})
\]

where \( \text{sem}(\Gamma, \text{network}) = G, \sigma \) is the identity network morphism on \( G \) and \( \mathcal{M} = \{(F, F) \mid F \in \text{Mod}(G)\} \).

If \( r \) is \text{SimpleOMSRefinement},

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

where

\[
\text{sem}(\Gamma, \text{r.refinement}) = ((G_1, G_1'), \sigma_1, \mathcal{M}_1),
\]

\[
\text{sem}(\Gamma, \text{r.refinement}) = ((G_2, G_2), \sigma_2, \mathcal{M}_2),
\]

\( G_1' \) and \( G_2 \) are both graphs with one isolated node, labeled \( \text{name1} = \tau_1, \tau_2, \tau_1, \tau_2, \Delta_1 \) and respectively \( \text{name2} = \tau_2, \tau_2, \tau_2, \Delta_2 \),

\[
\text{sem}(\Gamma, \Sigma_1, \Sigma_2, \text{r.omsRefinementMap}) = (\rho = (\Phi, \alpha, \beta) : I_1 \to I_2, \sigma : \Phi(\Sigma_1) \to \Sigma_2),
\]

\( \sigma' \) maps the node \( n \) of \( G_1 \) to the node of \( G_2 \) and \( (\sigma'_n)^M = (\sigma_1)_n^M ; (\rho; \sigma); (\sigma_2)^{\delta'_{\rho(n)}} \),

and \( \mathcal{M} = \{(M_1, M_1) \mid \exists M_2 \text{ such that } (M_2, M_2) \in \mathcal{M}_1 \text{ and } (M_1, M_2|_{\rho(n)}) \in \mathcal{M}_2\} \).

The refinement is correct only if for each \( (M, N) \in \mathcal{M}_2 \), there exists \( (M_1, M_2|_{\rho(n)}) \in \mathcal{M}_1 \).

If \( r \) is \text{SimpleNetworkRefinement},

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

where

\[
\text{sem}(\Gamma, \text{r.refinement}) = ((G_1, G_1'), \sigma_1, \mathcal{M}_1),
\]

\[
\text{sem}(\Gamma, \text{r.refinement}) = ((G_2, G_2), \sigma_2, \mathcal{M}_2),
\]

\[
\text{sem}(\Gamma, G_1', G_2, \text{r.networkRefinementMap}) = \sigma : G_1' \to G_2,
\]

\( \sigma' = \sigma_1^{\delta_1} ; \sigma ; \sigma_2 \)

and \( \mathcal{M} = \{(F_1, F_2) \mid \exists F_3 \text{ such that } (F_1, F_2|_{\rho(n)}) \in \mathcal{M}_1 \text{ and } (F_3, F_3) \in \mathcal{M}_2\} \).

The refinement is correct only if for each \( (F_2, F_3) \in \mathcal{M}_2 \), there is a \( (F_1, F_2|_{\rho(n)}) \in \mathcal{M}_1 \).

10.3.4.6 Semantics of a set of refinements

\[
\text{sem}(\Gamma, G_1, G_2, \text{Set}(\text{Refinement})) = \sigma : \text{GraphMorphism}
\]

If \( r_1, \ldots, r_n \) are all \text{Refinements},

\[
\text{sem}(\Gamma, G_1, G_2, \text{Set}(r_1, \ldots, r_n)) = \sigma
\]

where

\[
\text{sem}(\Gamma, r_1) = ((G_1^1, G_1^2), \sigma_1, \mathcal{M}_1),
\]

\[
\text{sem}(\Gamma, r_n) = ((G_n^1, G_n^2), \sigma_n, \mathcal{M}_n)
\]

such that \( G_1 = \bigcup_{i=1}^{n} G_i^1 \) and no node of \( G_1 \) appears in two graphs \( G_i^1 \) and \( G_j^1 \) for some \( i \neq j \in 1, \ldots, n \),

\( G_2 = \bigcup_{i=1}^{n} G_2^2 \) and no node of \( G_2 \) appears in two graphs \( G_i^2 \) and \( G_j^2 \) for some \( i \neq j \in 1, \ldots, n \),

and \( \sigma : G_1 \to G_2 \) is defined by \( \sigma^G(n) = \sigma^G_i(n) \) if the node \( n \) comes from \( G_i^1 \) and similarly for such a node \( n \) of \( G \) coming from \( G_i^1 \), we have that \( \sigma^M_n = \sigma^M_i(n) \). Moreover, \( \sigma \) must be total on the nodes of \( G_1 \).

10.3.4.7 Semantics of refinement maps

\[
\text{sem}(\Gamma, G_1, G_2, \text{RefinementMap}) = \sigma : \text{GraphMorphism}
\]

If \( m \) is an \text{OMSRefinementMap},

\[
\text{sem}(\Gamma, G_1, G_2, m) = (\text{name1}, \text{name2}, \rho, \sigma)
\]

where \( G_1 \) must be a graph with just one isolated node labeled \( \text{name1}, \tau_1, \tau_1, \Delta_1 \)
10.3.4.8 Semantics of entailment definitions

$$\text{sem}(\Gamma, \text{EntailmentDefinition}) = \Gamma'$$

where \(\Gamma'\) is a \text{LogicalEnvironment}.

If \(e\) is an \text{EntailmentDefinition},

$$\text{sem}(\Gamma, e) = \Gamma'$$

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

10.3.4.9 Semantics of entailment types

$$\text{sem}(\Gamma, \text{EntailmentType}) = G$$

where \(G\) is an \text{OMSOMSEntailment}, \(\text{sem}(\Gamma, t) = L_2 \xrightarrow{id} L_1\)

If \(t\) is a \text{NetworkOMSEntailment}, \(\text{sem}(\Gamma, t) = G\)

where \(\text{Name}(L_1) = \text{Name}(t.\text{premise}), \text{Name}(L_2) = \text{Name}(t.\text{conclusion}),\)

\((\text{Inst}(L_1), \text{Sig}(L_1), \text{Mod}(L_1), \text{Th}(L_1)) = \text{sem}(\Gamma, t.\text{premise}), (\text{Inst}(L_2), \text{Sig}(L_2), \text{Mod}(L_2), \text{Th}(L_2)) = \text{sem}(\Gamma, t.\text{conclusion})\)

such that \(\text{Sig}(L_1) = \text{Sig}(L_2)\) and \(\text{Mod}(L_1) \subseteq \text{Mod}(L_2)\) and \(id\) is the identity morphism on \(\text{Sig}(L_1)\).

If \(t\) is a \text{NetworkNetworkEntailment}, \(\text{sem}(\Gamma, t) = G\)

where \(\text{sem}(\Gamma, t.\text{premise}) = G_1, \text{sem}(\Gamma, t.\text{conclusion}) = G_2,\) such that \(\text{Shape}(G_1) = \text{Shape}(G_2)\) and, for each node \(i \in \text{Shape}(G_1)\), its names in the networks \(G_1\) and \(G_2\) are the same, its signatures are the same and the class of \text{models} \text{realizations} obtained by projecting each family of \text{models} \text{realizations} compatible with \(G_1\) to the component \(i\) is included in the class of \text{models} \text{realizations} obtained by projecting each family of \text{models} \text{realizations} compatible with \(G_2\) to the component \(i\). Then \(G\) extends the union of \(G_1\) and \(G_2\) for each pair of nodes \((i_1, i_2)\), where \(i_1\) and \(i_2\) identify the occurrences of the same node \(i\) in \(G_1\) and \(G_2\) respectively, with a theorem link from \(i_1\) to \(i_2\) labeled with the identity on \(\text{Sig}(i_1)\).
10.3.4.10 Semantics of equivalence definitions

\[ \text{sem}(\Gamma, \text{EquivalenceDefinition}) = \Gamma' : \text{LogicalEnvironment} \]

If \( d \) is an EquivalenceDefinition, \( \text{sem}(\Gamma, d) = \Gamma' \)

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

10.3.4.11 Semantics of OMS equivalences

\[ \text{sem}(\Gamma, \text{OMSEquivalence}) = (G, N_1, N_2) : (\text{OMSGraph}, \text{Node}, \text{Node}) \]

If \( t \) is an OMSEquivalence, \( \text{sem}(\Gamma, t) = (G, N_1, N_2) \)

where \( O_1 = t.\text{oms}, O_2 = t.\text{oms2}, O_3 = t.\text{mediatingOMS} \),

\( \text{sem}(\Gamma, (\mathcal{I}, \Sigma, \mathcal{M}, \Delta)) \) is the graph \( N_1 \xrightarrow{\imath_1} N_3 \xleftarrow{\imath_2} N_3 \) where

1) \( N_1 \) is labeled with \([\text{Name}(O_1), \text{Inst}(O_1), \Sigma, \text{Mod}(O_1), \text{Th}(O_1)]\),
2) \( N_2 \) is labeled with \([\text{Name}(O_2), \text{Inst}(O_2), \Sigma, \text{Mod}(O_2), \text{Th}(O_2)]\) and
3) \( N_3 \) is labeled with \([\text{Name}(O_3), \mathcal{I}, \Sigma, \mathcal{M}, \Delta] \)

such that

1) \( \imath : \Sigma \xrightarrow{} \Sigma \) are signature inclusions,
2) \( \mathcal{I} = \text{Inst}(O_1) = \text{Inst}(O_2) \) and
3) for each \( i = 1, 2 \) and each \( \text{model realization} M_i \in \text{Mod}(O_i) \) there exists a unique \( \text{model realization} M \in \mathcal{M} \) such that \( M|_{\Sigma(O_i)} = M_i \).

10.3.4.12 Semantics of network equivalences

\[ \text{sem}(\Gamma, \text{NetworkEquivalence}) = (G_1, G_2, G_3) : (\text{OMSGraph}, \text{OMSGraph}, \text{OMSGraph}) \]

If \( t \) is a NetworkEquivalence, \( \text{sem}(\Gamma, t) = (G_1, G_2, G_3) \)

where \( n_1 = t.\text{network}, n_2 = t.\text{network2}, n_3 = t.\text{mediatingNetwork} \),

\( \text{sem}(\Gamma, n_1) = G_1, \text{sem}(\Gamma, n_2) = G_2, \text{sem}(\Gamma, n_3) = G_3 \) such that \( G_1 \) and \( G_2 \) are subgraphs of \( G_3 \) and for each \( i = 1, 2 \) and each family of \( \text{model realizations} M_i \) compatible with \( G_i \) there is a unique family of \( \text{model realizations} M \) compatible with \( G_3 \) such that the projection of \( M \) to the nodes in \( G_i \) is \( M_i \).

10.3.4.13 Semantics of conservative extension definitions

\[ \text{sem}(\Gamma, \text{ConservativeExtensionDefinition}) = \Gamma' : \text{LogicalEnvironment} \]
If \( d \) is a ConservativeExtensionDefinition, 
\[
sem(\Gamma, d) = \Gamma'
\]
where \( O_1 = d.d.moduleType.module, O_2 = d.d.moduleType.whole, c = d.d.conservativityType, \)
\( \Sigma = sem(\Gamma, d.d.interfaceSignature), \Gamma' = \Gamma[d.d.interfaceSignature -> (G_1, N_2, N_1)] \) and \( G \) is the graph \( N_1 \rightarrow N_2 \) where \( N_1 \) is labeled with \((O_1, \text{Inst}(O_1), \text{Sig}(O_1), \text{Mod}(O_1), \text{Th}(O_1))\), \( N_2 \) with \((O_2, \text{Inst}(O_2), \text{Sig}(O_2), \text{Mod}(O_2), \text{Th}(O_2))\), and \( \iota \) is an inclusion, when \( \Sigma \subseteq \text{Sig}(O_2) \subseteq \text{Sig}(O_1) \) and if \( c = \text{model-conservative} \) and for each \( M \in \text{Mod}(O_2) \) there is a modal realization \( M' \in \text{Mod}(O_1) \) such that \( M'|_\Sigma = M|_\Sigma \), or if \( c = \text{consequence-conservative} \) and for each \( \varphi \in \text{Sem}(\Sigma), O_1 \models \varphi \) implies \( O_2 \models \varphi \).

### 10.3.4.14 Semantics of alignment definitions

\[
sem(\Gamma, \text{AlignmentDefinition}) = \Gamma'
\]

: \( \text{LogicalEnvironment} \)

If \( d \) is an AlignmentDefinition,
\[
sem(\Gamma, d) = \Gamma'
\]
where \( sem(\Gamma, d.d.alignmentType) = (\Gamma_0, L_1, L_2) \) and \( \Gamma' = \Gamma_0[d.d.alignmentName -> (G, L_1', L_2')] \),
\( (L_1', L_2') = sem(\Gamma, L_1, L_2, d.d.alignmentSemantics) \),
\( \text{card} = d.d.alignmentCardinality or, when this is missing, \text{card} = (1', 1') \),
\( aSem = d.d.alignmentSemantics or, when this is missing, aSem = \text{single-domain} \),
and \( G = sem(\Gamma_0, L_1', L_2', \text{card}, aSem, d.d.correspondence) \).

### 10.3.4.15 Semantics of alignment types

\[
sem(\Gamma, \text{AlignmentType}) = (\Gamma', L_1, L_2)
\]

: \( \text{(LogicalEnvironment, NodeLabel, NodeLabel)} \)

If \( t \) is an AlignmentType
\[
sem(\Gamma, t) = (\Gamma'', L_1, L_2)
\]
where \( sem(\Gamma, t.d.source) = (\Gamma'', (I_1, \Sigma_1, M_1, \Delta_1)), sem(\Gamma, t.d.target) = (\Gamma'', (I_2, \Sigma_2, M_2, \Delta_2)) \), \( L_1 \) and \( L_2 \) are the labels of the nodes of \( t.d.source \) and \( t.d.target \) in \( \Gamma'' \).imports.

### 10.3.4.16 Semantics of alignments

\[
sem(\Gamma, L_1, L_2, \text{(AlignmentCardinality, AlignmentCardinality), AlignmentSemantics, Set(Correspondence))} = G
\]

: \( \text{OMSGraph} \)

If \( \text{card}_1, \text{card}_2 \) are AlignmentCardinality, \( aSem \) is an AlignmentSemantics and \( C = \text{Set}\{c_1, \ldots, c_n\} \) a set of Correspondences,
\[
sem(\Gamma, L_1, L_2, \text{(card}_1, \text{card}_2), aSem, C) = G
\]
where \( sem(\Gamma, \text{Sig}(L_1), \text{Sig}(L_2), aSem, C) = (\Sigma_1, \Sigma_2, \text{(\Sigma, \Delta)}, \phi_s : \Sigma_s \rightarrow \Sigma, \phi_t : \Sigma_t \rightarrow \Sigma, \text{smap}, \text{cvalues}) \),
\( \text{where the semantics of the alignment is not defined in the following cases:} \)

- if \( \text{cvalues} = \text{True} \) and then at least one of the correspondences in \( C \) has a confidence value different than 1 or
- if the alignment does not have the specified cardinality, i.e.
  - if \( \text{card}_1 = '1' \), then \( \text{smap} \) must be injective,
  - if \( \text{card}_1 = '+' \), then \( \text{smap} \) must be total on the symbols of \( \text{Sig}(L_1) \).
\[ cvalues = \begin{cases} \text{False} & \text{conf} = 1 \\ \text{True} & \text{otherwise} \end{cases} \]

\[ \text{sem}(\Gamma, \Sigma_1, c._\text{generalizedTerm}) = \text{sym}1, \text{sem}(\Gamma, \Sigma_2, c._\text{symbolRef}) = \text{sym}2, \text{clist} = \text{Sequence}(\text{rel}, \text{sym}1, \text{sym}2), \]

If \( c \) is a CorrespondenceBlock,

\[ \text{sem}(\Gamma, \Sigma_1, \Sigma_2, (\text{defaultConf}, \text{defaultRel}), c) = (\text{clist}, \text{cvalues}) \]

where if \( c._\text{relation} \) is missing, \( \text{rel} = \text{defaultRel} \), else \( \text{rel} = c._\text{relation} \)

if \( c._\text{confidence} \) is missing, \( \text{conf} = \text{defaultConf} \), else \( \text{conf} = c._\text{confidence} \)

for all correspondences \( c_1, \ldots, c_n \) in \( c._\text{correspondence} \),

\[ \text{clist} = \text{clist}_1 + \ldots + \text{clist}_n \]

and \( \text{cvalues} = \bigvee_{i=1,\ldots,n} \text{cvalues}_i \).

\[ \text{sem}(\Gamma, L_1, L_2, \text{AlignmentSemantics}) = ((\text{Name}_1, I'_1, \Sigma'_1, M'_1, \Delta'_1), (\text{Name}_2, cI'_2, \Sigma'_2, M'_2, \Delta'_2)) \]

: (NodeLabel, NodeLabel)

If \( s \) is an AlignmentSemantics, \( L_1 = (a\text{Name}, I_1, \Sigma_1, M_1, \Delta_1) \) and \( L_2 = (a\text{Name}', I_2, \Sigma_2, M_2, \Delta_2) \)

\[ \text{sem}(\Gamma, L_1, L_2, s) = (\text{rel}(L_1), \text{rel}(L_2)) \]

where

\[ \text{rel}(L) = \begin{cases} (L_1, L_2) & \text{if } s \text{ is single-domain} \\ ((\text{name}_1, I_1, \Sigma'_1, M'_1, \Delta'_1), (\text{name}_2, I_2, \Sigma'_2, M'_2, \Delta'_2)) & \text{otherwise} \end{cases} \]

\[ \text{relativize}_1(\Sigma_1, \Delta_1) = (\Sigma'_1, \Delta'_1), M'_1 \text{ is the class of } \Delta'_1-\text{models-realizations, } \text{name}_1 = '\text{ relativized} + a\text{Name}, \]

\[ \text{relativize}_2(\Sigma_2, \Delta_2) = (\Sigma'_2, \Delta'_2), M'_2 \text{ is the class of } \Delta'_2-\text{models-realizations and } \text{name}_2 = '\text{ relativized} + a\text{Name}'. \]
C.3.2.3 RDF Conformance of a Modified Serialization of OWL in RDF With DOL

The serialization of OWL in RDF (regardless of the concrete RDF serialization employed to serialize the RDF graph that represents the OWL ontology) does not satisfy requirement 3 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 O₁ importing an ontology O₂, as an RDF resource. However, only resources can have identifiers in RDF. RDF reification would allow for turning the statement O₁ owl:imports O₂ 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. 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.

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. i rdf:subject s.
2. i rdf:predicate owl:imports.
3. i rdf:object o.

and thus introduces a resource i to represent that the ontology s imports the ontology o, these three triples are removed from G. From an ontology serialized in this super-language of the serialization of OWL in RDF, one can obtain semantically equivalent ontologies (with regard to the semantics of OWL) by stripping all triples whose predicate is rdf:subject, rdf:predicate or rdf:object, or by adding triples that declare these three properties to be annotation properties.

C.4 Semantic Conformance of OWL 2 With DOL

The logic 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 ALC is discussed, afterward the approach is generalized to the more complex description logic SROIQ, which is underlying OWL 2 DL. Signatures of the description logic ALC consist of a set A of atomic concepts, a set R of roles and a set I of individual constants. Signature morphisms are tuples of functions, one for each signature component. Models Realizations are first-order structures I = (∆I, ⦵I) with universe ∆I that interpret concepts as unary and roles as binary predicates (using ' '). I₁ ⊆ I₂ if ∆I₁ = ∆I₂ and all concepts and roles of I₁ are subconcepts and subroles of those in I₂. Sentences are subsumption relations C₁ ⊆ C₂ between concepts, where concepts follow the grammar

\[ C ::= A \mid \top \mid C_1 \sqcup C_2 \mid C_1 \sqcap C_2 \mid \neg C \mid \forall R.C \mid \exists R.C \]

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 ∈ I) or pairs of individuals in roles (written R(a, b) for a, b ∈ I, R ∈ R). Satisfaction is the standard satisfaction of description logics.

The logic SROIQ, which is the logical core of the Web Ontology Language OWL 2 DL, extends ALC with the following constructs: (i) complex role inclusions such as R ⊑ S ⊆ S as well as simple role hierarchies such as R ⊆ S, assertions for symmetric, transitive, reflexive, asymmetric and disjoint roles (called RBox sentences, denoted by SR), as well as the construct ∃R.Self (collecting the set of ‘R-reflexive points’); (ii) nominals, i.e. concepts of the form {a}, where a ∈ I (denoted by O); (iii) inverse roles (denoted by I); qualified and unqualified number restrictions (Q). For details on the rather complex grammatical restrictions of 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).

30) NR20 section 3
31) See the last sentence of section 3.2.5 of NR20
32) See also http://www.w3.org/TR/owl2-overview/
Definition 18  Given a datatype map $D$ and a vocabulary $V$ over $D$, an interpretation

$$I = (\Delta_I, \Delta_D, C, OP, DP, I, DT, LT, FA, NAMED)$$

for $D$ and $V$ is a 10-tuple with the following structure:

- $\Delta_I$ is a nonempty set called the object domain.
- $\Delta_D$ is a nonempty set disjoint with $\Delta_I$ called the data domain such that $(DT)^{DT} \subseteq \Delta_D$ for each datatype $DT \in V_DT$.
- $C$ is the class interpretation function that assigns to each class $C \in V_C$ a subset $(C)^C \subseteq \Delta_I$ such that
  - $\text{owl:Thing}^C = \Delta_I$ and
  - $(\text{owl:Nothing})^C = \emptyset$.
- $OP$ 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
  - $\text{owl:topObjectProperty}^OP = \Delta_I \times \Delta_I$ and
  - $\text{owl:bottomObjectProperty}^OP = \emptyset$.
- $DP$ 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
  - $\text{owl:topDataProperty}^DP = \Delta_I \times \Delta_D$ and
  - $\text{owl:bottomDataProperty}^DP = \emptyset$.
- $I$ is the individual interpretation function that assigns to each individual $a \in V_I$ an element $(a)^I \in \Delta_I$.
- $DT$ is the datatype interpretation function that assigns to each datatype $DT \in V_DT$ a subset $(DT)^{DT} \subseteq \Delta_D$ such that
  - $\text{owl:topDataProperty}^DT = \Delta_D$.
- $LT$ 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 $lt$.
- $FA$ 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 $\Delta_I$ such that $(a)^I \in NAMED$ for each named individual $a \in V_I$.

The institution $\text{SROIQ}(D)$ underlying OWL is now defined as follows:

Definition 19  — An $\text{SROIQ}(D)$ signature is a pair $(D, V)$, where $D$ is a datatype map and $V$ a vocabulary over $D$.  

Given $\text{SROIQ}(D)$ signatures $(D, V)$ and $(D', V')$, a $\text{SROIQ}(D)$ signature morphism $\sigma: (D, V) \to (D', V')$ only exists if $D \subseteq D'$. In this case, such a signature morphism consists of

- a map $\sigma_C: V_C \to V'_C$,
- a map $\sigma_{OP}: V_{OP} \to V'_{OP}$,
- a map $\sigma_{DP}: V_{DP} \to V'_{DP}$,
- a map $\sigma_I: V_I \to V'_I$,
- a map $\sigma_{DT}: V_{DT} \to V'_{DT}$ that is the identity on $V_{DT} \cup \{\text{rdfs:Literal}\}$,
- a map $\sigma_{LT}: V_{LT} \to V'_{LT}$

The sentences for a signature are defined as in the direct model-theoretic semantics of OWL [61]. Sentence translation is substitution of symbols.

$(D, V)$-models—realizations are interpretations for $D$ and $V$. Morphisms of $(D, V)$-model morphisms—models are maps between the domains $\Delta_I$ preserving membership in classes and properties, where $\Delta_D$ is mapped identically. Model reducts Reducts of realizations are built by first translating along the signature morphism and then looking up the interpretation in the model realization to be reduced.
— The satisfaction relation is defined as in direct model-theoretic semantics of OWL \cite{67}.

\[\square\]

Remark: strictly speaking, the institution defined above is OWL 2 DL without restrictions in the sense of \cite{68}. 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.

\subsection{C.4.1 Relativization in OWL}

\textbf{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

\begin{itemize}
  \item \(C' = C \cup \{\top\}\)
  \item \(\Delta'\) contains axioms stating that:
    \begin{itemize}
      \item each concept in \(C\) is subsumed by \(\top\),
      \item each individual in \(I\) is an instance of \(\top\),
      \item each role \(r\) has its domain and range intersected with \(\top\), if they are present in \(\Delta\), otherwise they are \(\top\), and, for each sentence \(e \in \Delta\), the sentence \(\alpha(e)\), obtained by replacing the concepts in \(e\) as follows: are made:
        \begin{itemize}
          \item each occurrence of \(\top\) is replaced with \(\top\),
          \item each occurrence of \(\neg C\) is replaced with \(\top \cap \neg C\),
          \item each occurrence of \(\forall r \bullet C\) is replaced with \(\top \cap \forall r \bullet C\).
        \end{itemize}
    \end{itemize}
\end{itemize}

\textbf{Definition 21} Given an OWL theory \(T = ((C, R, I), \Delta)\), we define \(\beta : \text{Mod}^{\text{OWL}}(\tilde{T}) \rightarrow \text{Mod}^{\text{OWL}}(T)\) as follows: if \(M' \in \text{Mod}^{\text{OWL}}(\tilde{T})\), then \(M = \beta(M')\) has as universe \(\Delta'\) the set \((\top)\setminus\Delta'\) and each concept, role and individual are interpreted in \(M\) in the same way as in \(M'\). Since \(M'\) is a \(\Delta'\)-model-realization, we get that \(M\) is indeed a \((C, R, I)\)-model-realization and moreover \(M \models \Delta\).

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

\subsection{C.4.2 Translating correspondences to a bridge theory in OWL}

We define the function \(\text{theoryOfCorrespondences}_{\text{OML}}\) 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 \((\text{relation}, \text{sourceSymbol}, \text{targetSymbol})\).

The result of the function is a co-span of theories: \(\xymatrix{\Sigma_s \ar[r]^s & \Sigma & \Sigma_t \ar[l]_{\Sigma} }\). Intuitively, \(\Sigma_s\) and \(\Sigma_t\) gather the symbols of the aligned ontologies that appear in the list of correspondences passed as an argument, while \((\Sigma, \Delta)\) contains an OWL sentence representing each correspondence.

We distinguish three cases.

1) Single domain:
   - no other symbols occur in the signatures \(\Sigma_s\) and \(\Sigma_t\) 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.
Conformance of Common Logic with DOL

D.1 Abstract Syntax Conformance of Common Logic With DOL

The metaclass Text is a subclass (in the sense of SMOF) of NativeDocument. The metaclass Sentence is a subclass (in the sense of SMOF) of BasicOMS.

D.2 Serialization Conformance of Common Logic With DOL

The semantic conformance of Common Logic (as specified in Annex D with DOL is established in [56]. The XCF dialect of Common Logic has a serialization that satisfies the criteria for XML conformance. The CLIF dialect of Common Logic has a serialization that satisfies the criteria for text conformance.

D.3 Semantic Conformance of Common Logic With DOL

Common Logic can be defined as an institution as follows:

Definition 22 Common Logic. A common logic signature \( \Sigma \) (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 \( \Sigma \)-\textit{realization} \( I = (UR, UD, rel, fun, int, seq) \) consists of a set \( UR \), the universe of reference, with a non-empty subset \( UD \subseteq UR \), the universe of discourse, and four mappings:

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

A \( \Sigma \)-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.

\begin{itemize}
  \item \textbf{Model reducts: Reducts of realizations} are defined in the following way: Given a signature morphism \( \sigma : \Sigma_1 \to \Sigma_2 \) and a \( \Sigma_2 \)-\textit{model–realization} \( I_2 = (UR, UD, rel, fun, int, seq) \) and \( I_1 = (UR, UD, rel_1, fun_1, int_1, seq_1) \), then \( \sigma(I_1) \). A homomorphism of realizations \( h : I_1 \to I_2 \) is a function \( h : UR_1 \to UR_2 \) such that
    \begin{itemize}
      \item for each \( x \in UR_1 \) and \( s \in UD_1 \), then \( h(s) \in rel_2(h(x)) \quad [34]
      \item for each \( x \in UR_1 \), \( k \circ fun_1(x) = fun_2(h(x)) \circ k^* \),
    \end{itemize}
\end{itemize}

\[ [34] k^* \] is the extension of \( h \) to sequences.
Annex E
(informative)
Conformance of RDF and RDF Schema with DOL

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

The metaclass 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 DOL is established in [56].

Definition 23 (RDF and RDF Schema) The institutions for the Resource Description Framework (RDF) and RDF Schema (also known as RDFS), respectively, are defined in the following [44]. Both RDF and RDFS are based on a logic called bare RDF (SimpleRDF), which consists of triples only (without any predefined resources).

A signature Rs in SimpleRDF is a set of resource references. For sub,pred,obj ∈ Rs, 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 RDF-model-realization M = (Rm, Pm, Sm, EXTm) consists of a set Rm of resources, a set Pm ⊆ Rm of predicates, a mapping function Sm : Rs → Rm, and an extension function EXTm : Pm → P(Rm × Rm) mapping every predicate to a set of pairs of resources. Satisfaction is defined as follows:

∀M |= Rs (sub, pred, obj) ⇔ (Sm(sub), (Sm(obj)) ∈ EXTm(Sm(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 model-realizations.

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

(rdf:type, rdf:type, rdf:Property), and
(rdf:subject, rdf:type, rdf:Property).

In the model-realizations, the standard vocabulary is interpreted with a fixed model-realization. Moreover, for each RDF-model-realization M = (Rm, Pm, Sm, EXTm), if p ∈ Pm, then it must hold (p, Sm(rdf:Property)) ∈ EXTm(rdf:type). For RDFS, similar conditions are formulated (here, for example also the subclass relation is fixed).

In the case of RDFS, the standard vocabulary contains more elements, like rdfs:domain, rdfs:range, rdfs:Resource, rdfs:Literal, rdfs:Datatype, rdfs:Class, rdfs:subClassOf, rdfs:subPropertyOf, rdfs:member, rdfs:Container, rdfs:ContainerMembershipProperty.

There is also OWL Full, an extension of RDFS with resources such as owl:Thing and owl:oneOf, tailored towards the representation of OWL [24].
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.linear : Set[Line],
Station.net : Set[Net], Station.unit : Set[Unit], Station.track : Set[Track],
Line.net : Set[Net], Line.linear : Set[Linear],
Connector.unit : Set[Unit],
Unit.station : Set[Station], Unit.connector : Set[Connector],
Track.station : Set[Station], Track.linear : Set[Linear],
Line.track : Set[Track], Linear.line : Set[Line]

Associations: L2L(line : Set[Line], linear : Set[Linear]),
L2T(linear : Set[Linear], track : Set[Track]),
C2U(connector : Set[Connector], unit : Set[Unit])
N2S(net : Set[Net], *station : Set[Station]),
N2L(net : Set[Net], *line : Set[Line]),
S2U(station : Set[Station], *unit : Set[Unit]),
S2T(station : Set[Station], *track : Set[Track])

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

Figure F.1 – Sample UML class model

F.4.3 Realizations

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

For a classifier net \( \Sigma = ((C, \leq C), P, O, A) \), a Common Logic theory \( CL(\Sigma) \) is defined consisting of:

- for \( c \in C \), a predicate \( CL(c) \), such that

\[ 43 \] Strictly speaking, this is just a name.
\[
\{ \exists \; (x_1 \ldots x_r) \; (\text{and} \; (\text{CL}(c_1) \; x_1) \ldots (\text{CL}(c_r) \; x_r) \\
(= \; \tau \; (\text{form:sequence-insert} \; x_1 \ldots (\text{form:sequence-insert} \; x_r \; \text{form:empty-sequence})))) )\}
\]

In case that all the \( \tau_i \) are omitted (or, equivalently, equal to \( \text{Set} \)), the representation is simplified to an \( r \)-ary predicate:

\[
\{ \forall \; (x_1 \; x_2 \ldots x_r) \; \text{if} \; (\text{CL}(a) \; x_1 \; x_2 \ldots x_r) \; (\text{and} \; (\text{CL}(c_1) \; x_1) \ldots (\text{CL}(c_r) \; x_r)) \}
\]

- the interpretation of a member end of a binary association declaration owned by a class/data type coincides with the interpretation of the attribute: if for \( i \in \{1, 2\} \), \( a.p_i : \tau_i[c_i] \) for \( a = a(p_1 : \tau_1[c_1], p_2 : \tau_2[c_2]) \in A \) is owned by \( c \in C \) with \( c.p_i : \tau_i[c_i] \in P \), then

\[
\{ \forall \; (\circ \; s) \; (\text{if} \; (\text{CL}(c.p) \; \circ \; s) \; (= \; s \; (\text{form:seq2CL}(\tau_i) \; (\text{form:selecti} \; \circ \; \text{CL}(a)))) )\}
\]

If \( a \) is represented in simplified form, then instead the following is used

\[
\{ \forall \; (\circ \; s) \; (\text{if} \; (\text{CL}(c.p) \; \circ \; s) \; (\forall \; x \; (\text{iff} \; (\text{member} \; x \; s) \; (\text{CL}(a) \; \circ \; x)))) )\}
\]

- For the compositions, let \( c^1.p^1 : \tau[c^1], \ldots, c^k.p^k : \tau[c^k] \) be all the composite attributes in \( P \) and \( a^1 = a^1(p_1 : \text{Set}[c_1], p_2 : \tau^1_2[c_1]), \ldots, a^r = a^r(p_1 : \text{Set}[c_1], p_2 : \tau^r_2[c_1]) \) all the composite binary associations in \( A \). Abbreviate

\[
\{ \text{or} \; (\text{CL}(c^1.p^1) \; \circ \; x) \ldots (\text{CL}(c^k.p^k) \; \circ \; x) \; (\text{form:sequence-member} \; (\text{form:pair} \; \circ \; x) \; (\text{CL}(a^1)) \ldots (\text{form:sequence-member} \; (\text{form:pair} \; \circ \; x) \; (\text{CL}(a^r))) )\}
\]

by \( (\text{owner} \; \circ \; x) \), where, for each binary association \( a^i \) represented in the simplified way, \( (\text{CL}(a^i) \; \circ \; i) \) replaces \( (\text{form:sequence-member} \; (\text{form:pair} \; \circ \; x) \; (\text{CL}(a^i))) \). Then

\[
\{ \forall \; (o_1 \; o_2 \; x) \; (\text{if} \; (\text{and} \; (\text{owner} \; o_1 \; x) \; (\text{owner} \; o_2 \; x)) \; (= \; o_1 \; o_2)) \}
\]

It is straightforward to extend \( \text{CL} \) from signatures to signature morphisms.

**Models—Realizations.** A \( \Sigma \)-model—realization of the UML class model institution is just a \( \text{CL}(\Sigma) \)-model—realization in Common Logic. That is, the UML class model institution inherits model realizations from Common Logic. Moreover, model-reduct—reducts of realizations are inherited as well, using the action of \( \text{CL} \) on signature morphisms.
I.5.2 QL → OWL and DL-Lite \(\rightarrow SROIQ(D)\)

QL → OWL is the sublanguage inclusion obtained by the syntactic restriction according to the definition of QL, see [NR6]. Since by definition, DL-Lite \(\rightarrow SROIQ(D)\) is the corresponding sublogic inclusion.

I.5.3 RL → OWL and RL → SROIQ(D)

RL → OWL is the sublanguage inclusion obtained by the syntactic restriction according to the definition of RL, see [NR6]. Since by definition, RL → SROIQ(D) is the corresponding sublogic inclusion.

I.5.4 SimpleRDF → RDF

SimpleRDF → RDF is an obvious inclusion, except that SimpleRDF resources need to be renamed if they happen to have a predefined meaning in RDF. The model translation needs to forget the fixed parts of RDF realizatons. Since this part can always reconstructed in a unique way, the result is an isomorphic model translation.

I.5.5 RDF → RDFS

This is entirely analogous to SimpleRDF → RDF.

I.5.6 SimpleRDF → SROIQ(D)

A SimpleRDF signature is translated to SROIQ(D) by providing a class \(P\) and three roles sub, pred and obj (these reify the extension relation), and one individual per SimpleRDF resource. A SimpleRDF triple \((s, p, o)\) is translated to the SROIQ(D) sentence

\[
\top \sqsubseteq \exists U. (\exists \text{sub.}\{s\} \cap \exists \text{pred.}\{p\} \cap \exists \text{obj.}\{o\}).
\]

From an SROIQ(D) model realization \(I\), obtain a SimpleRDF model realization by inheriting the universe and the interpretation of individuals (then turned into resources). The interpretation \(P^I\) of \(P\) gives \(P_m\), and \(\text{EXT}_m\) is obtained by de-reifying, i.e.

\[
\text{EXT}_m(x) := \{(y, z) \mid \exists u. (u, x) \in \text{pred}^T, (u, y) \in \text{sub}^T, (u, z) \in \text{obj}^T\}.
\]

RDF → SROIQ(D) is defined similarly. The theory of RDF built-ins is (after translation to SROIQ(D)) added to any signature translation. This ensures that the model translation can add the built-ins.

I.5.7 OWL → FOL

I.5.7.1 Translation of signatures

\(\Phi((C, R, I)) = (F, P)\) with

- function symbols: \(F = \{a^{(1)} | a \in I\}\)
- predicate symbols \(P = \{A^{(1)} | A \in C\} \cup \{R^{(2)} | R \in R\}\)

I.5.7.2 Translation of sentences

Concepts are translated as follows:
I.5.8  \textit{FOL} \rightarrow \textit{CL}

This comorphism maps classical first-order logic (FOL) to Common Logic.

A FOL signature is translated to CL by turning all constants into discourse names, and all other function symbols and all predicate symbols into non-discourse names. A FOL sentence is translated to CL by a straightforward recursion, the base being translations of predications:

$$\alpha_\Sigma(P(t_1,\ldots,t_n)) = (P \alpha_\Sigma(t_1) \ldots \alpha_\Sigma(t_n))$$

Within terms, function applications are translated similarly:

$$\alpha_\Sigma(f(t_1,\ldots,t_n)) = (f \alpha_\Sigma(t_1) \ldots \alpha_\Sigma(t_n))$$

A CL model realization is translated to a FOL model realization by using the universe of discourse as FOL universe. The interpretation of constants is directly given by the interpretation of the corresponding names in CL. The interpretation of a predicate symbol $P$ is given by using $rel^M(int^M(P))$ and restricting to the arity of $P$; similarly for function symbols (using $fun^M$). Both the satisfaction condition and model-expansiveness of the comorphism are straightforward.

I.5.9  \textit{OWL} \rightarrow \textit{CL}

This comorphism is the composition of the comorphisms described in the previous two sections.

I.5.10  UML class models \rightarrow \textit{CL}

This translation has been described in annex \[76\]. Translation of signatures is detailed in section \[76\]. Translation of sentences in section \[76\]. Model Realizations are translated identically.

I.5.11  \textit{FOL} \rightarrow \textit{Casl}

This is an obvious sublogic.

I.5.12  UML class model to \textit{OWL}

Let $\Sigma = ((C, \leq_C), P, O, A, M)$ be a class/data type net representing a UML class model as described in annex \[76\]. This net can be translated to OWL2 using the approach described in \[76\]. The ontology is extended by translating parts of this net and its multiplicity constraints $Mult(\Sigma)$:

- For each class $c \in C$ with superclasses $c_1, c_2, \ldots, c_n \in C$ (i.e. $c \leq_C c_i$ for $i = 1, \ldots, n$):
  - \textbf{Class}: c
    - \textbf{SubClassOf}: $c_1$
    - \text{...}
    - \textbf{SubClassOf}: $c_n$

- For each attribute declaration $c.p : c'$ in $P$
  - \textbf{ObjectProperty}: p
    - \textbf{Domain}: c
    - \textbf{Range}: $c'$

- For each attribute multiplicity $n \leq c.p : \tau[c']$ in $Mult(\Sigma)$ extend the description of class $c$ by:
  - \textbf{SubClassOf}: p $\min n$ $c'$

Distributed Ontology, Model, and Specification Language (DOL), v1.0 Beta 135
Annex P
(informative)
Introduction to Category Theory

P.1 Categories

Definition 25 A category C consists of

— a class of objects, denoted |C|,
— for each two objects a and b, a class of morphisms (or arrows), denoted C(a,b),
— for each three objects a, b and c, a composition operation, denoted ; : C(a,b) × C(b,c) → C(a,c) such that the following axioms hold:
  — if f ∈ C(a,b), g ∈ C(b,c) and h ∈ C(c,d) for four objects a, b, c, d, then f;(g;h) = (f;g);h
  — for each object a there is a morphism id_a ∈ C(a,a) such that for every f ∈ C(a,b) and every g ∈ C(b,a) for some object b we have that id_a;f = f and g;id_a = g.

Example Set is the category whose class of objects is the class of all sets, Set(A,B) is the set of all functions from A to B for any sets A and B, id_A is the identity function on a set A and the composition is the usual composition of functions.

Example Rel is the category whose class of objects is the class of all sets, Rel(A,B) is the class of all relations R ⊆ A × B, for any sets A and B, id_A is the diagonal relation {(a,a) | a ∈ A} for a set A and the composition of R ∈ Rel(A,B) with S ∈ Rel(B,C) for three sets A, B, C is defined as {(a,c) | exists b ∈ B such that (a,b) ∈ R and (b,c) ∈ S}.

Example The category of unsorted first-order signatures has as objects tuples of the form (σ, ::) of function symbols of arity i, for each natural number i. Given two objects F and G, a morphism σ : F → G is a family of functions (σ_i : F_i → G_i)_{i∈N}, which means that the arities of function symbols are preserved by morphisms. The identity morphism for an object F is the family of identity functions (id_F_i)_{i∈N} and the composition is defined component-wise: if σ : F → G and τ : G → H are signature morphisms between the signatures F,G and H, then σ;τ = (σ_i;τ_i)_{i∈N}.

Example Given an unsorted first-order signature F, a model realization M of F consists of an universe M_U together with an interpretation of each function symbol f ∈ F_i as a function M_f taking i arguments in M_U with result in M_U. Given two such model-realizations M and N, a model homomorphism homomorphism of realizations m : M → N is a function m : M_U → N_U such that for each i ∈ N and each f ∈ F_i we have that m(M_f(x_1,...,x_n)) = N_f(m(x_1),...,m(x_n)) for every x_1,...,x_n in M_U. The identity function on M_U is a model homomorphism homomorphism of realizations on M and the composition is the usual composition of functions. This gives us the category of first-order model-realizations of F.

Definition 26 Let C be a category. Its dual or opposite category, denoted C^{op}

— has the same objects as C: |C^{op}| = |C|,
— for two objects a, b ∈ |C|, C^{op}(a,b) = C(b,a),
— ;^{op} : C^{op}(a,b) × C^{op}(b,c) → C^{op}(a,c) is defined as f;^{op} g = g; f for any f ∈ C^{op}(a,b) = C(b,a) and g ∈ C^{op}(b,c) = C(c,b). The result g; f is a morphism in C(c,a) = C^{op}(a,c),
— for each object a, id_a ∈ C^{op}(a,a) = C(a,a) is the identity w.r.t. the composition ;^{op}.

Definition 27 An object A is called an initial object in a category C if for each object B of C there is exactly one morphism from A to B.

Definition 28 An object A is called a terminal object in a category C if for each object B of C there is exactly one morphism from B to A.

Example In Set, the empty set is the initial object and each singleton set is a terminal object.
P.1.1 Limits and colimits

**Definition 29** A network in a category $C$ is a functor $D : G \to C$, where $G$ is a small category and can be thought of as the shape of the graph of interconnections between the objects of $C$ selected by the functor $D$.

**Definition 30** A cocone of a network $D : G \to C$ consists of an object $c$ of $C$ and a family of morphisms $\alpha_i : D(i) \to c$, for each object $i$ of $G$, such that for each edge of the network, $e : i \to i'$ it holds that $D(e) \alpha_i = \alpha_{i'}$.

**Definition 31** A colimiting cocone (or colimit) $(c, \{\alpha_i\}_{i \in |G|})$ has the property that for any cocone $(d, \{\beta_i\}_{i \in |G|})$ there exists a unique morphism $\gamma : c \to d$ such that $\alpha_i \gamma = \beta_i$.

By dropping the uniqueness condition and requiring only that a morphism $\gamma$ should exist, a weak colimit is obtained.

When $G$ is the category $\bullet \leftarrow \bullet$, $G$-colimits are called pushouts. When $G$ is a discrete category (i.e. no arrows between objects other than identities), $G$-limits are called coproducts.

**Definition 32** A cone of a network $D : G \to C$ consists of an object $c$ of $C$ and a family of morphisms $\alpha_i : c \to D(i)$, for each object $i$ of $G$, such that for each edge of the network, $e : i \to i'$ it holds that $\alpha_i e = \alpha_{i'} D(e)$.

**Definition 33** A limiting cone (or limit) $(c, \{\alpha_i\}_{i \in |G|})$ has the property that for any cone $(d, \{\beta_i\}_{i \in |G|})$ there exists a unique morphism $\gamma : c \to d$ such that $\gamma \alpha_i = \beta_i$.

When $G$ is the category $\bullet \leftarrow \bullet$, $G$-limits are called pullbacks. When $G$ is a discrete category, $G$-limits are called products.

P.2 Functors

**Definition 34** Let $C$ and $D$ be two categories. A functor $F : C \to D$ is a mapping that

- assigns to each object $c$ of $C$ an object $F(c)$ in $D$,
- assigns to each morphism $f \in C(c,d)$ a morphism $F(f) \in D(F(c),F(d))$ such that
  - $F(id_c) = id_{F(c)}$ for each $c \in |C|$,
  - $F(f;g) = F(f);F(g)$ for each $f \in C(a,b)$, $g \in C(b,c)$ and $a, b, c \in |C|$.

**Example** For each category $C$, the identity functor $id_C : C \to C$ takes each object and each morphism to itself.

**Example** The forgetful functor $F$ from the category of unsorted first-order model-realizations of a signature $F$ to $\text{Set}$ takes each model-realization $M$ to the set $M_U$ and each model morphism morphism of realizations $m : M \to N$ to its underlying function $m : M_U \to N_U$.

**Example** The covariant powerset functor $\mathcal{P} : \text{Set} \to \text{Set}$ maps each set $A$ to the set of all subsets of $A$ and each function $f : A \to B$ to the function that takes a subset $X$ of $A$ to the set $\{f(x) \mid x \in X\}$, which is a subset of $B$.

**Example** The covariant finite powerset functor $\mathcal{P}^f : \text{Set} \to \text{Set}$ maps each set $A$ to the set of all finite subsets of $A$ and each function $f : A \to B$ to the function that takes a subset $X$ of $A$ to the set $\{f(x) \mid x \in X\}$, which is a subset of $B$.

---

51) A network is called a diagram in category theory texts. This terminology is introduced to disambiguate OMS networks from UML diagrams.

52) That is, it has a set of objects and sets of morphisms between them instead of classes.