The Distributed Ontology, Modeling and Specification Language – DOL

Till Mossakowski, Mihai Codescu, Fabian Neuhaus and Oliver Kutz

Abstract. There is a diversity of ontology languages in use, among them OWL, RDF, OBO, Common Logic, and F-logic. Related languages such as UML class diagrams, entity-relationship diagrams and object role modelling provide bridges from ontology modelling to applications, e.g. in software engineering and databases. Also in model-driven engineering, there is a diversity of diagrams: UML consists of 15 different diagram types, and SysML provides further types. Finally, in software and hardware specification, a variety of formalisms are in use, like Z, VDM, first-order logic, temporal logic etc.

Another diversity appears at the level of ontology, model and specification modularity and relations among ontologies, specifications and models. There is ontology matching and alignment, module extraction, interpolation, ontologies linked by bridges, interpretation and refinement, and combination of ontologies, models and specifications.

The Distributed Ontology, Modeling and Specification Language (DOL) aims at providing a unified metalanguage for handling this diversity. In particular, DOL provides constructs for (1) "as-is" use of ontologies, models and specifications (OMS) formulated in a specific ontology, modelling or specification language, (2) OMS formalised in heterogeneous logics, (3) modular OMS, (4) mappings between OMS, and (5) networks of OMS. This paper sketches the design of the DOL language. DOL has been submitted as a proposal within the OntoIOp (Ontology, Model, Specification Integration and Interoperability) standardisation activity of the Object Management Group (OMG).

Mathematics Subject Classification (2000). Primary 68T30; Secondary 03C95.

Keywords. heterogeneous ontologies, modularity, interoperability, institutions.

1. Introduction

Logical languages are used in several fields of computing for the development of formal, machine-processable texts that carry a formal semantics. Among those fields

are 1) **O**ntologies formalising domain knowledge, 2) (formal) **M**odels of systems, and 3) the formal **S**pecification of systems. Ontologies, models and specifications will (for the purpose of this paper) henceforth be abbreviated as **OMS**.

An OMS provides formal descriptions which range in scope from domain knowledge and activities (ontologies, models) to properties and behaviours of hardware and software systems (models, specifications). While the use of OMS varies considerably, there are two recurring challenges: *reusability* and *interoperability*.

Reusability is an issue because the development of OMS is typically done manually by experts and, thus, an expensive process. Hence, it is desirable to be able to reuse existing OMS during the development of new OMS. This presupposes a framework that allows to build *structured OMS* by identifying modules and their relationships to each other. For example, it requires the ability to combine two existing OMS in a way that handles the namespaces of the OMS in an appropriate way. Further, the reuse of an existing OMS often requires that the OMS is *adapted* for its new purpose. For example, the adaption may require the extension of the OMS by new axioms, or the extraction of a subset of the OMS, or the change of its semantics from open world to closed world.

The interoperability challenge is closely related to the reusability challenge. Since the development of OMS is not an exact science and is usually driven by project specific requirements, two OMS that have been developed independently will represent the same domain in different and, often, conflicting ways. They may differ, for example, with respect to the terminology, or with respect to the definitions of the underlying concepts, or with respect to the perspective from which they represent their domain. Thus, in a situation where two independently developed OMS are supposed to be reused as modules of a larger OMS, the differences between these OMS will typically prevent them from working together properly. Overcoming this lack of interoperability may require an alignment or even an integration of these OMS. This typically involves the identification of synonyms, homonyms, and the development of bridge axioms, which connect the two OMS appropriately.

Both the reusability and the interoperability challenges are amplified by the diversity of OMS languages that are in use. For ontologies these include OWL, RDF, OBO, Common Logic, and F-logic. Related languages such as UML class diagrams, entity-relationship diagrams and object role modelling provide bridges from ontology modelling to applications, e.g., in software engineering and databases. Also in model-driven engineering, there is a diversity of diagrams: UML consists of 15 different diagram types, and SysML provides further types. Finally, in software and hardware specification, a variety of formalisms are in use, like Z, VDM, first-order logic, temporal logic etc. These languages do not just differ with respect to their syntax, but with respect to their semantics and to their levels of expressiveness.

To address both challenges we propose the Distributed Ontology, Modeling and Specification Language (DOL). DOL is a metalanguage that enables the reuse, integration, and alignment of existing OMS – even if they are written in different

formalisms. The underlying methodological stance is that it would be futile to attempt to develop yet another OMS language that would subsume all the others; instead we have to accept the diversity of OMS languages and the diversity of perspectives that are represented by different OMS. DOL provides a sound and formal semantic basis for specifying structured OMS, which may reuse as modules several existing OMS (possibly written in different languages) without requiring any changes to these modules. Further, DOL allows to specify mappings between different OMS (e.g., alignments and logical entailments).

In particular, DOL enjoys the following distinctive features:

- modular OMS and OMS networks are specially supported,
- OMS can not only be aligned (as in BioPortal [40] and NeON [17]), but also combined along alignments,
- mappings between OMS (interpretation of theories, conservative extensions etc.) are supported,
- it supports a variety of OMS languages (OWL, RDF, Common Logic, first-order logic, CASL; planned: UML, relational database schema, F-logic, distributed description logics, and more),
- OMS can be translated to other OMS languages, and compared with OMS in other languages,
- heterogeneous OMS (i.e., structured OMS with modules written in different languages) can be built,
- OMS languages and OMS language translations are first-class citizens and are available on the Web as linked data.

The paper is organised as follows: we first discuss the theoretical foundations of DOL in Section 2, followed by a sketch of the DOL language itself in Section 3. Section 4 briefly discusses the DOL-enabled, web-based OMS repository engine Ontohub, and Section 5 concludes.

2. Foundations of the Distributed Ontology, Modeling and Specification Language (DOL)

The Distributed Ontology, Modeling and Specification Language (DOL)[1] aims at providing a unified framework for (1) "as-is" use of OMS formulated in a specific OMS language, (2) modular OMS, (3) mappings between OMS, (4) OMS networks, and (5) OMS formalised in heterogeneous logics. Historically, the design of DOL has inherited many ideas and features (1) discussed in the Workshop on Modular Ontologies series [16, 15, 43, 24, 28, 45], (2) from the Alignment API [10], (3) from CLEAR, ASL and specifications in an arbitrary institution [5, 47, 41, 42], and (4) from the CASL (Common Algebraic Specification Language)

[1] DOL has formerly been standardised within ISO/TC 37/SC 3. The OntoIOp (Ontology, Modelling and Specification Integration and Interoperability) activity is now being continued at OMG, see the project page at `http://ontoiop.org`.

and HetCASL (CASL's heterogeneous extension) languages, standardised in IFIP WG 1.3[2] (Foundations of System Specification) [2, 30, 35, 25].

A library in DOL consists of modules formalised in *basic OMS languages*, such as OWL (based on description logic) or Common Logic (based on first-order logic with some second-order features). These modules are serialised in the existing syntaxes of these languages in order to facilitate reuse of existing OMS. DOL adds a meta-level on top, which allows for expressing heterogeneous OMS and mappings between OMS.[3] Such mappings include (heterogeneous) *imports* and *alignments*, *conservative extensions* (important for studying OMS modules), and *theory interpretations* (important for reusing proofs). Thus, DOL gives OMS interoperability a formal grounding and makes heterogeneous OMS and services based on them amenable to automated verification. The basic syntax and semantics of DOL has been introduced in [38, 37], and the general theory of heterogeneous specifications for OMS in [27]. DOL uses internationalised resource identifiers (IRIs, the Unicode-aware superset of URIs) for all entities of OMS libraries to make them referenceable on the Web.

2.1. Foundations

The large variety of logical languages in use can be captured at an abstract level using the concept of *institutions* [12]. This allows us to develop results independently of the particularities of a logical system and to use the notions of institution and logical language interchangeably throughout the rest of the paper.

The main idea is to collect the non-logical symbols of the language in signatures and to assign to each signature the set of sentences that can be formed with its symbols. For each signature, we provide means for extracting the symbols it consists of, together with their kind. Signature morphisms are mappings between signatures. We do not assume any details except that signature morphisms can be composed and that there are identity morphisms; this amounts to a category of signatures. Readers unfamiliar with category theory may replace this with a partial order (signature morphisms are then just inclusions). See [37] for details of this simplified foundation.

Institutions also provide a model theory, which introduces semantics for the language and gives a satisfaction relation between the models and the sentences of a signature. The main restriction imposed is the satisfaction condition, which captures the idea that truth is invariant under change of notation (and enlargement of context) along signature morphisms. This relies on two further components of institutions: the translation of sentences along signature morphisms, and the reduction of models against signature morphisms (generalising the notion of model reduct known from logic).

---

[2] See `http://ifipwg13.informatik.uni-bremen.de`

[3] The languages that we call "basic" OMS languages here are usually limited to one logic and do not provide meta-theoretical constructs.

Definition 2.1. An **institution** [12] is a quadruple $I = (\mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \models)$ consisting of the following:

- a category **Sign** of *signatures* and *signature morphisms*,
- a functor $\mathbf{Sen} : \mathbf{Sign} \to \mathbb{S}et$[4] giving, for each signature Σ, the set of *sentences* $\mathbf{Sen}(\Sigma)$, and for each signature morphism $\sigma : \Sigma \to \Sigma'$, the *sentence translation map* $\mathbf{Sen}(\sigma) : \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$, where often $\mathbf{Sen}(\sigma)(\varphi)$ is written as $\sigma(\varphi)$,
- a functor $\mathbf{Mod} : \mathbf{Sign}^{op} \to \mathbb{C}at$[5] giving, for each signature Σ, the category of *models* $\mathbf{Mod}(\Sigma)$, and for each signature morphism $\sigma\colon \Sigma \longrightarrow \Sigma'$, the *reduct functor* $\mathbf{Mod}(\sigma) : \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$, where often $\mathbf{Mod}(\sigma)(M')$ is written as $M'|_\sigma$, and $M'|_\sigma$ is called the *σ-reduct* of M', while M' is called a *σ-expansion* of $M'|_\sigma$,
- a satisfaction relation $|_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$ for each $\Sigma \in |\mathbf{Sign}|$,

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

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

for each $M' \in |\mathbf{Mod}(\Sigma')|$ and $\varphi \in \mathbf{Sen}(\Sigma)$. □

It is also possible to complement an institution with a proof theory, introducing a derivability relation between sentences, formalised as an *entailment system* [33]. In particular, this can be done for all logics that have so far been in use in DOL.

Example. OWL signatures consist of sets of atomic classes, individuals, object and data properties. OWL signature morphisms map classes to classes, individuals to individuals, object properties to object properties and data properties to data properties. For an OWL signature Σ, sentences are subsumption relations between classes or properties, membership assertions of individuals in classes and pairs of individuals in properties, complex role inclusions, and some more. Sentence translation along a signature morphism simply replaces non-logical symbols with their image along the morphism. The kinds of symbols are class, individual, object property and data property, respectively, and the set of symbols of a signature is the union of its sets of classes, individuals and properties. Models are (unsorted) first-order structures that interpret concepts as unary and properties as binary predicates, and individuals as elements of the universe of the structure, and satisfaction is the standard satisfaction of description logics. This gives us an institution for OWL.

Strictly speaking, this institution captures *OWL 2 DL without restrictions* in the sense of [44]. 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 modelled *inside* this institution, as a constraint on OMS. This constraint is

[4] $\mathbb{S}et$ is the category having all sets as objects and functions as arrows.

[5] $\mathbb{C}at$ is the category of categories and functors. Strictly speaking, $\mathbb{C}at$ is not a category but only a so-called quasicategory, which is a category that lives in a higher set-theoretic universe.

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.

In this framework, a basic OMS O over an institution I is a pair (Σ, E) where Σ is a signature and E is a set of Σ-sentences. Given a basic OMS O, we denote by $\mathsf{Sig}(O)$ the signature of the OMS. An OMS morphism $\sigma : (\Sigma_1, E_1) \to (\Sigma_2, E_2)$ is a signature morphism $\sigma : \Sigma_1 \to \Sigma_2$ such that $\sigma(E_1)$ is a logical consequence of E_2. Here, in an arbitrary institution, notions like logical consequence, satisfiability etc. can be defined in the standard way.

In the following we will need to assume existence of inclusions between signatures and of their unions. These concepts can be captured in a categorical setting using *inclusion systems* [11]. However, inclusion systems are too strong for our purposes and therefore we will work under weaker assumptions.

Definition 2.2. An *inclusive category* [14] is a category having a broad subcategory[6] which is a partially ordered class with finite products and coproducts, called intersection (denoted $\cap$) and union (denoted $\cup$) such that for each pair of objects A, B, $A \cup B$ is a pushout of $A \cap B$ in the category.

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

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

for each span where one arrow is an inclusion.

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

Definition 2.3. An institution is *weakly inclusive* if

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

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

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

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

2.2. Translations between Institutions

Several notions of *translations* between institutions can be introduced. The most frequently used variant are *institution comorphisms* [13]. A comorphism from institution L_1 to institution L_2 maps L_1-signatures to L_2-signatures along a functor Φ and Σ-sentences in L_1 to $\Phi(\Sigma)$-sentences in L_2, for each L_1-signature Σ, while $\Phi(\Sigma)$-models are mapped to Σ-models. Again, a satisfaction condition has to be fulfilled. For *institution morphisms* [13], the directions of the translation of sentences and models are reversed.

Definition 2.4. An **institution comorphism** from an institution $I = (\mathbf{Sign}^I, \mathbf{Mod}^I, \mathbf{Sen}^I, \models^I)$ to an institution $J = (\mathbf{Sign}^J, \mathbf{Mod}^J, \mathbf{Sen}^J, \models^J)$ consists of a functor $\Phi : \mathbf{Sign}^I \longrightarrow \mathbf{Sign}^J$, and two natural transformations $\beta : \mathbf{Mod}^J \circ \Phi \Longrightarrow \mathbf{Mod}^I$ and $\alpha : \mathbf{Sen}^I \Longrightarrow \mathbf{Sen}^J \circ \Phi$, such that

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

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

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

$$\begin{array}{ccc} Sen^I(\Sigma) & \xrightarrow{\alpha_\Sigma} & Sen^J(\Phi(\Sigma)) \\ {\scriptstyle Sen^I(\sigma)}\downarrow & & \downarrow{\scriptstyle Sen^J(\Phi(\sigma))} \\ Sen^I(\Sigma') & \xrightarrow[\alpha_{\Sigma'}]{} & Sen^J(\Phi(\Sigma')) \end{array} \qquad \begin{array}{ccc} Mod^J(\Phi(\Sigma')) & \xrightarrow{\beta_{\Sigma'}} & Mod^I(\Sigma') \\ \downarrow{\scriptstyle Mod^J(\Phi(\sigma))} & & \downarrow{\scriptstyle Mod^I(\sigma)} \\ Mod^J(\Phi(\Sigma)) & \xrightarrow[\beta_\Sigma]{} & Mod^I(\Sigma) \end{array}$$

Definition 2.5. An **institution morphism** from an institution $I = (\mathbf{Sign}^I, \mathbf{Mod}^I, \mathbf{Sen}^I, \models^I)$ to an institution $J = (\mathbf{Sign}^J, \mathbf{Mod}^J, \mathbf{Sen}^J, \models^J)$ consists of a functor $\Phi : \mathbf{Sign}^I \longrightarrow \mathbf{Sign}^J$, and two natural transformations $\beta : \mathbf{Mod}^I \Longrightarrow \mathbf{Mod}^J \circ \Phi$ and $\alpha : \mathbf{Sen}^J \circ \Phi \Longrightarrow \mathbf{Sen}^I$, such that

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

holds, called the **satisfaction condition**.

Mappings of institutions are split along the following dichotomies:

- *translation* versus *projection*: a translation embeds or encodes a logic into another one, while a projection is a forgetful operation (e.g. the projection from first-order logic to propositional logic forgets predicates with arity greater than zero). It is an interesting informal observation that translations can be formalised as institution comorphisms, and projections as institution morphisms.

- *plain mapping* versus *simple theoroidal mapping* [13]: while a plain mapping needs to map signatures to signatures, a (simple) theoroidal mapping maps signatures to theories. The latter therefore allows for using "infrastructure axioms": e.g. when mapping OWL to Common Logic, it is convenient to rely on a first-order axiomatisation of a transitivity predicate for properties.

Mappings can also be classified according to their accuracy; see [36] for details. *Sublogics* are the most accurate mappings: they are syntactic subsets. *Embeddings* come close to sublogics, like injective functions come close to subsets. A mapping can be *faithful* in the sense that logical consequence (or logical deduction) is preserved and reflected, that is, inference systems and reasoning engines for the target logic can be reused for the source logic (along the mapping). *(Weak) exactness* is a technical property that guarantees this faithfulness even in the presences of OMS structuring operations [4].

2.3. A Graph of Logic Translations

Figure 1 is a revised and extended version of the graph of logics and translations introduced in [36]. New nodes include UML class diagrams, OWL-Full (i.e. OWL with an RDF semantics instead of description logic semantics), and Common Logic without second-order features (CL^-). We have defined the translations between most of these logics in earlier publications [38, 36]. The definitions of the DOL conformance of some central standard OMS languages and translations among them will be given as annexes to the standard and published in an open registry, which is also the place where the remaining definitions will be maintained.

3. The Language DOL

3.1. DOL Syntax and Semantics

The DOL language is not "yet another OMS language", but a *metalanguage* for expressing relations between OMS. Therefore, any OMS written in any conforming OMS language also is a DOL OMS. Therefore, when working with DOL users can reuse OMS as they are, no changes are required.

DOL provides abstract syntax categories for:

1. OMS (ontologies, models and specifications). *Basic OMS* are OMS that are written in some OMS language (e.g., OWL or CASL). A *modular* or *structured* OMS is written in a modular way, with the help of DOL structuring operations. A heterogeneous OMS is a modular OMS that involves modules, which are written in different OMS languages. The semantics of OMS is given by a signature and a class of models. In some cases, we can additionally provide a theory-level semantics of OMS, as a signature and a class of sentences that, if it exists, agrees with the model-level semantics (that is, the model class is equal to the class of models satisfying the theory). We call an OMS *flattenable* if it has a theory-level semantics and *elusive* if it only admits a

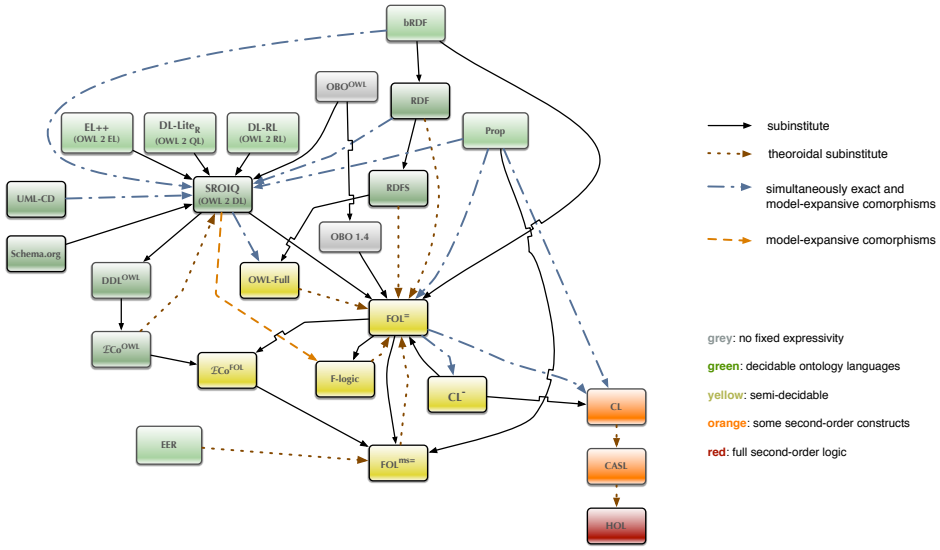

FIGURE 1. The current logic translation graph for DOL-conforming languages

model-level semantics. Whether an OMS is flattenable can be decided based on the structuring operations on OMS, as follows:

- **: Flattenable OMS:** basic OMS are flattenable; if all their components OMS are flattenable, then the following operations on OMS yield flattenable OMS: extension, union, translation, interpolate/forget, extract, reference, qualification, combination.
- **: Elusive OMS:** the reduction, minimisation, or maximisation of an OMS is elusive; further, any OMS containing an elusive OMS is elusive.[7]

For detailed definitions of these types of OMS, see Section 3.2.

2. OMS *mappings*. They denote relations between two OMS or OMS networks, typically along a signature morphism. Some mappings may also involve other OMS or other signatures. Examples of OMS mappings are interpretations (specifying a logical consequence relationship between OMS), equivalences of OMS (specifying that their model classes are in bijective correspondence), conservative extensions (between OMS and their modules), OMS alignment. They are presented in Section 3.3.
3. *Networks* of OMS. Networks are graphs with nodes labelled with OMS and edges labelled with OMS mappings. The edges show how two OMS are interlinked. The rationale behind networks is that they provide a way to specify

[7]Note that extension, union, translation, reference, qualification and combination are defined for flattenable and elusive OMS, while interpolate/forget and extract are only defined for flattenable OMS.

or model complex distributed systems (or domains), where a single OMS would become too complex (this especially can be the case if the OMS are formulated in different OMS languages). Instead, the different OMS of the network provide different viewpoints on the system, while their compatibility is ensured via mappings. Networks are discussed in Section 3.4.

4. *Libraries* of OMS. OMS, mappings and networks are organised in libraries. A library consists of a list of declarations involving (possibly modular and/or heterogeneous) OMS. These declarations can be definitions (assigning a name to an OMS, OMS mapping or network of OMS) and qualifications of the current language, logic and/or serialisation. This is detailed in Section 3.5.

The semantics of DOL is based on a fixed (but in principle arbitrary) logic graph. A logic graph is given by a collection of institutions, institution morphisms and institution comorphisms (serving as logics, logic reductions and logic translations). Moreover, some of the institution comorphisms are marked as default translations and some of the institution morphisms are marked as default projection (but only at most one between a given source and target institution).

We assume that for each institution in the logic graph there is a trivial signature $\emptyset$ with model class $\mathcal{M}_\emptyset$ and such that there exists a unique signature morphism from $\emptyset$ to any signature of the institution. Moreover we assume the existence of a designated error logic in the graph, and a partial union operation on logics, denoted $\bigcup$: $L_1 \bigcup L_2 = (L, \rho_1 : L_1 \to L, \rho_2 : L_2 \to L)$, when defined.

3.2. Modular and Heterogeneous OMS

Modular and heterogeneous OMS are generated by the following grammar, where Σ is a signature, Δ is a set of sentences over Σ, σ a signature morphism, I an institution, ρ an institution comorphism and μ an institution morphism[8]:

```
OMS ::= ⟨I, Σ, Δ⟩
      | IRI
      | OMS and OMS | OMS then OMS
      | OMS with σ | OMS with translation ρ
      | OMS reveal Σ | OMS hide Σ | OMS hide along μ
      | OMS keep Σ [keep I] | OMS keep I | OMS forget Σ [keep I]
      | OMS extract Σ | OMS remove Σ
      | OMS select ⟨Σ, Δ⟩ | OMS reject ⟨Σ, Δ⟩
      | minimize OMS | maximize OMS
      | combine Network
```

The semantics of an OMS O has four components:

- the institution of O, denoted $\mathbf{Inst}(O)$,
- the signature of O, denoted $\mathbf{Sign}(O)$ (which is a signature in $\mathbf{Inst}(O)$),
- the models of O, denoted $\mathbf{Mod}(O)$ (which is a class of models over $\mathbf{Sign}(O)$),

[8]This is a mathematically abstracted version of DOL. In reality, signatures are represented by symbol sets, and signature morphisms by symbol maps. The details of passing from symbol sets (resp. maps) to signatures (resp. signature morphisms) are left out here. Also, we have left out OMS bridges, since their design is still being discussed.

- the axioms of O, denoted $\mathbf{Ax}(O)$ (which is a set of sentences over $\mathbf{Sign}(O)$).[9]

For elusive OMS, $\mathbf{Ax}(O)$ is undefined. For flattenable OMS, $\mathbf{Mod}(O)$ can be obtained as $\mathbf{Mod}(O) = \{M \in \mathbf{Mod}(\mathbf{Sign}(O)) \mid M \models \mathbf{Ax}(O)\}$.

In the following we discuss the different kinds of (possibly modular and/or heterogeneous) OMS.

3.2.1. Basic OMS

A *basic OMS* O written inline, in a conforming OMS language and serialisation. The semantics is inherited from the OMS language I[10] and results in a theory $\langle \Sigma, \Delta \rangle$ (therefore, for simplicity, in the syntax above, we have identified the basic OMS with $\langle I, \Sigma, \Delta \rangle$). O can also be an OMS fragment, which means that some of the symbols or axioms may refer to symbols declared outside O (i.e. in an imported OMS). This is mainly used for extensions and equivalences. Here are two sample ontologies in OWL (using Manchester syntax) and Common Logic (using CLIF):

```
Class: Woman EquivalentTo: Person and Female
ObjectProperty: hasParent
```

```
(cl-module PreOrder
  (forall (x) (le x x))
  (forall (x y z) (if (and (le x y) (le y z)) (le x z))))
```

Formally,

- $\mathbf{Inst}(I, \Sigma, \Delta) = I$
- $\mathbf{Sign}(I, \Sigma, \Delta) = \Sigma$
- $\mathbf{Mod}(I, \Sigma, \Delta) = \{M \in \mathbf{Mod}(\Sigma) \mid M \models \Delta\}$
- $\mathbf{Ax}(I, \Sigma, \Delta) = \Delta$.

3.2.2. IRI reference

An IRI reference to an OMS existing on the Web[11], possibly abbreviated using prefixes.[12] For example:

```
<http://owl.cs.manchester.ac.uk/co-ode-files/ontologies/pizza.owl>
```

or alternatively

[9] The *theory* of O, written, $\mathbf{Th}(O)$, is the closure of $\mathbf{Ax}(O)$ under logical entailment. Note, however, that throughout the text we use 'theory' also more informally as denoting some set of axioms in a particular signature and logic.

[10] I is normally determined by the context of the enclosing library and passed around as an additional parameter of the semantics. For simplicity, here we let I become part of the basic OMS.

[11] Note that not all OMS can be downloaded by dereferencing their IRIs. Implementing a catalogue mechanism in DOL-aware applications might remedy this problem.

[12] Some of the following listings abbreviate IRIs using prefixes but omit the prefix bindings for readability.

```
%prefix(
  co-ode: <http://owl.cs.manchester.ac.uk/co-ode-files/ontologies/> )%
co-ode:pizza.owl
```

The semantics of such an IRI reference would require a *global environment* mapping IRIs to (semantics of) OMS. However, for simplicity, we omit the global environment (and therefore also the semantics of IRI references) here.

3.2.3. Extension

An *extension* of an OMS by new symbols and axioms, written O_1 **then** O_2, where O_2 is an OMS (fragment) in a conforming OMS language. The resulting signature is that of O_1, augmented with the symbols in O_2. A model of an extension OMS is a model of this signature, that satisfies the axioms on O_2 and is (when appropriately reduced) a model of O_1. An extension can optionally be marked as conservative (%mcons or %ccons after the "**then**"). The semantics is that each O_1-model must have at least one expansion to the whole extension O_1 **then** O_2 (for %mcons) resp. that each logical consequence of O_1 **then** O_2 is already one of O_1 if it is over the signature of O_1 (for %ccons). In case that O_2 does not introduce any new symbols, the keyword %implied can be used instead of %ccons or %mcons; the extension then merely states intended logical consequences. The keyword %def stands for definitional extensions. This is similar to %mcons, but the model expansion must always exist uniquely. The following OWL ontology is an example for the latter:

```
  Class Person
  Class Female
then %def
  Class: Woman EquivalentTo: Person and Female
```

The semantics of $O = O_1$ **then** O_2 is

- $\mathbf{Inst}(O) = \mathbf{Inst}(O_1) = \mathbf{Inst}_{O_1}(O_2)$
- $\mathbf{Sign}(O) = \mathbf{Sign}(O_1) \cup \mathbf{Sign}_{O_1}(O_2)$
- $\mathbf{Mod}(O) = \{M \in \mathbf{Mod}(\mathbf{Sign}(O)) \mid M|_{\mathbf{Sign}(O_i)} \in \mathbf{Mod}(O_i), \text{ for } i = 1, 2\}$
- $\mathbf{Ax}(O) = \mathbf{Ax}(O_1) \cup \mathbf{Ax}_{O_1}(O_2)$

where O_2 is analysed in the context of previous declarations in O_1, as indicated by adding an index in its semantics.

3.2.4. Union

A *union* of two self-contained OMS (not fragments), written O_1 **and** O_2. Models of this union are those models that are (perhaps after appropriate reduction) models of both O_1 and O_2. For example, the class of commutative monoids can be expressed as

```
algebra:Monoid and algebra:Commutative
```

Forming a union of OMS is a particularly common operation in the RDF logic, where it is known as merging graphs [18, section 0.3]; however, the RDF language provides no explicit syntax for this operation. When multiple RDF ontologies ("graphs") contain statements about the same symbol ("resource"), i.e., syntactically, triples having the same subject, the effect is that in the merged graph the resource will have all properties that have previously been stated about it separately. Different kinds of properties, e.g. multilingual labels, geodata, or outgoing links to external graphs, are often maintained in different RDF graphs, which are then merged; consider the following excerpt:

```
{ :OVGU rdfs:label "Otto-von-Guericke-Universität Magdeburg"@de . } and
{ :OVGU geo:lat "52.1403"^^xsd:float . } and
{ :OVGU owl:sameAs[13]
    <http://de.dbpedia.org/page/OvGU> . }
```

The semantics of $O = O_1$ **and** O_2 is

- $\mathbf{Inst}(O) = I$ where $\mathbf{Inst}(O_1) \bigcup \mathbf{Inst}(O_2) = (I, (\Phi_1, \alpha_1, \beta_1) : \mathbf{Inst}(O_1) \to I, (\Phi_2, \alpha_2, \beta_2) : \mathbf{Inst}(O_2) \to I)$
- $\mathbf{Sign}(O) = \Phi_1(\mathbf{Sign}(O_1)) \cup \Phi_2(\mathbf{Sign}(O_2))$
- $\mathbf{Mod}(O) = \{M \in \mathbf{Mod}(\mathbf{Sign}(O)) \mid \beta_{\Sigma_i}(M|_{\Phi_i(\mathbf{Sign}(O_i))}) \in \mathbf{Mod}(O_i), \text{ for } i = 1, 2\}$
- $\mathbf{Ax}(O) = \alpha_1(\mathbf{Ax}(O_1)) \cup \alpha_2(\mathbf{Ax}(O_2))$.

3.2.5. Translation

A *translation* of an OMS to a different signature (written O **with** σ, where σ is a signature morphism) or into some OMS language (written O **with translation** ρ, where ρ is an institution comorphism). For example, we can combine an OWL ontology with a first-order axiom (formulated in Common Logic) as follows:

```
logic OWL : {
  ObjectProperty: isProperPartOf
  Characteristics: Asymmetric
  SubPropertyOf: isPartOf }
with translation OWL2CommonLogic
then
  (if (and (isProperPartOf x y) (isProperPartOf y z)) (isProperPartOf x z))
```

Note that OWL can express transitivity, but not together with asymmetry.

The semantics of $O = O'$ **with** σ is

- $\mathbf{Inst}(O) = \mathbf{Inst}(O')$
- $\mathbf{Sign}(O) = \Sigma'$ where $\sigma : \mathbf{Sign}(O') \to \Sigma'$
- $\mathbf{Mod}(O) = \{M \in \mathbf{Mod}(\Sigma') \mid M|_\sigma \in \mathbf{Mod}(O')\}$

[13] While *owl:sameAs* is borrowed from the *vocabulary* of OWL, it is commonly used in the RDF logic to link to resources in external graphs, which should be treated as if their IRI were the same as the subject's IRI.

- $\mathbf{Ax}(O) = \sigma(\mathbf{Ax}(O'))$.

The semantics of $O = O'$ **with translation** ρ is

- $\mathbf{Inst}(O) = I$, where $\rho = (\Phi, \alpha, \beta) : \mathbf{Inst}(O') \to I$
- $\mathbf{Sign}(O) = \Phi(\mathbf{Sign}(O))$
- $\mathbf{Mod}(O) = \{M \in \mathbf{Mod}(\mathbf{Sign}(O)) \mid \beta_{\mathbf{Sign}(O)}(M) \in \mathbf{Mod}(O')\}$
- $\mathbf{Ax}(O) = \alpha_{\mathbf{Sign}(O)}(\mathbf{Ax}(O'))$.

3.2.6. Reduction

A *reduction* of an OMS to a smaller signature Σ is written O **reveal** Σ. Alternatively, it can be written O **hide** Σ, where Σ is the set of symbols to be hidden (i.e. this is equivalent to O **reveal** $\mathsf{Sig}(O) \setminus \Sigma$). The effect is an existential quantification over all hidden symbols. For example, when specifying a group in sorted first-order logic, using the CASL language,

```
sort Elem
ops 0: Elem; __+__: Elem * Elem -> Elem; inv: Elem -> Elem
forall x, y, z : Elem
       . 0 + x       = x
       . x + (y + z) = (x + y) + z
       . x + inv(x)  = 0
reveal Elem, 0, __+__
```

revealing everything except the inverse operation `inv` results in a specification of the class of all monoids that can be extended with an inverse operation, i.e. the class of all groups with inverse left implicit.

Here is an example of hiding:

```
ontology Pizza = %% a simplified remake of the Pizza ontology [19]
  Individual: TomatoTopping
  Individual: MozzarellaTopping DifferentFrom: TomatoTopping
  ObjectProperty: hasTopping
  Class: VegetarianTopping
    EquivalentTo: { TomatoTopping, MozzarellaTopping, ... }
  Class: VegetarianPizza SubClassOf: some hasTopping VegetarianTopping
  ...
end

ontology Pizza_hide_VegetarianTopping =
  Pizza hide VegetarianTopping
end
```

A reduction to a less expressive logic is written O **hide along** μ, where μ is an institution morphism. This is a common operation in TBox/ABox settings, where an ontology in an expressive language provides the terminology (TBox) used in

assertions (ABox) stated in a logic that is less expressive but scales to larger data sets; OWL DL (whose logic is $\mathcal{SROIQ}$) vs. RDF is a typical language combination:

```
ontology TBoxABox =
  Pizza hide along OWL22RDF
  then logic RDF : {
    :myPizza :hasTopping
      [ a :TomatoTopping ], [ a :MozzarellaTopping ] .
  }
```

The semantics of $O = O'$ **reveal** Σ' is

- $\mathbf{Inst}(O) = \mathbf{Inst}(O')$
- $\mathbf{Sign}(O) = \Sigma'$
- $\mathbf{Mod}(O) = \{M|_\iota \mid M \in \mathbf{Mod}(O')\}$ where $\iota : \Sigma' \to \mathbf{Sign}(O')$ is the inclusion
- $\mathbf{Ax}(O)$ is undefined.

The semantics of $O = O'$ **hide** Σ' is

- $\mathbf{Inst}(O) = \mathbf{Inst}(O')$
- $\mathbf{Sign}(O) = \mathbf{Sign}(O') \setminus \Sigma'$
- $\mathbf{Mod}(O) = \{M|_\iota \mid M \in \mathbf{Mod}(O')\}$ where $\iota : \mathbf{Sign}(O') \setminus \Sigma' \to \mathbf{Sign}(O')$ is the inclusion
- $\mathbf{Ax}(O)$ is undefined.

The semantics of $O = O'$ **hide along** μ is

- $\mathbf{Inst}(O) = I$ where $\mu = (\Phi, \alpha, \beta) : \mathbf{Inst}(O) \to I$
- $\mathbf{Sign}(O) = \Phi(\mathbf{Sign}(O'))$
- $\mathbf{Mod}(O) = \{\beta_{\mathbf{Sign}(O')}(M) \mid M \in \mathbf{Mod}(O')\}$
- $\mathbf{Ax}(O)$ is undefined.

3.2.7. Filtering

A *filtering* O **select** $\langle \Sigma, \Delta \rangle$, which selects those sentences from O that have signature Σ, plus those in Δ (where Δ is a subset $\mathbf{Ax}(O)$). It can also be written O **reject** $\langle \Sigma, \Delta \rangle$, where Σ is the set of symbols and Δ the set of axioms to be hidden. For example, we can select all axioms of Galen[14] involving Drugs, Joints, or Bodyparts by:

```
logic OWL
ontology myGalen =
   <http://example.org/GALEN/galen.owl>
 select Drugs, Joints, Bodyparts
end
```

The semantics of $O = O'$ **select** $\langle \Sigma, \Delta \rangle$ is defined only if $\Sigma \subseteq \mathbf{Sign}(O)$ and $\Delta \subseteq \mathbf{Ax}(O)$, and in that case, it is given by

[14] We assume that GALEN is available as an OWL ontology.

- $\mathbf{Inst}(O) = \mathbf{Inst}(O')$
- $\mathbf{Sign}(O) = \Sigma'$ where Σ' is the smallest signature with $\Sigma \subseteq \Sigma'$ and $\Delta \subseteq \mathbf{Sen}(\Sigma)$[15]
- $\mathbf{Ax}(O) = (\mathbf{Ax}(O') \cap \mathbf{Sen}(\mathbf{Sign}(O))) \cup \Delta$
- $\mathbf{Mod}(O)$ is the class of all $\mathbf{Ax}(O)$-models.

The semantics of $O = O'$ **reject** (Σ, Δ) is

- $\mathbf{Inst}(O) = \mathbf{Inst}(O')$
- $\mathbf{Sign}(O) = \mathbf{Sign}(O') \setminus \Sigma$
- $\mathbf{Ax}(O) = \mathbf{Ax}(O') \cap \mathbf{Sen}(\mathbf{Sign}(O)) \setminus \Delta$
- $\mathbf{Mod}(O)$ is the class of all $\mathbf{Ax}(O)$-models.

3.2.8. Interpolation

An *approximation* [31] (or technically, *uniform interpolation*) of an OMS, either in a subsignature or a sublogic (written O **keep** Σ, O **keep** Σ **keep** L or O **keep** L, where Σ is a signature and L is a logic). The effect is that sentences not expressible in Σ (resp. L) are weakened or removed, but the resulting theory still has the same consequences, as fas as these are expressible in Σ (and/or L). Technically, this is a uniform interpolant [46, 32]. For example, we can interpolate the first-order DOLCE mereology in OWL:[16]

```
DOLCE_Mereology keep OWL
```

Dually, O **forget** Σ or O **forget** Σ **keep** L interpolates O with the signature $\mathsf{Sig}(O) \setminus \Sigma$, i.e. Σ specifies the symbols that need to be left out (and optionally, L specifies a sublogic that needs to be targeted). Cf. the notion of forgetting in [46, 32]. For example,

```
Pizza forget VegetarianTopping
```

This has both a model-theoretic and a theory-level semantics, i.e., it yields a theory in the reduced signature (without `VegetarianTopping`). In contrast, `Pizza hide VegetarianTopping` has only a model-level semantics (see also the comparison in section 3.2.12).

The semantics of $O = O'$ **keep** Σ **keep** I is

- $\mathbf{Inst}(O) = I$ and $(\Phi, \alpha, \beta) : \mathbf{Inst}(O') \to I$ is the default projection (in case I is missing, it is the identity on $\mathbf{Inst}(O')$)
- $\mathbf{Sign}(O) = \Phi(\Sigma)$
- $\mathbf{Ax}(O) = \alpha^{-1}_{\mathbf{Sign}(O')}(\mathbf{Ax}(O')^\bullet) \cap \mathbf{Sen}^I(\mathbf{Sign}(O))$[17], i.e. that part of $\mathbf{Ax}(O')$ that can be expressed in the smaller signature and logic
- $\mathbf{Mod}(O)$ is the class of $\mathbf{Ax}(O)$-models

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

[16] Interpolants need not always exist, and even if they do, tools might only be able to approximate them.

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

The semantics of O **forget** Σ' **keep** I is the same as the semantics of O **keep** $(\mathbf{Sign}(O) \setminus \Sigma')$ **keep** I.

3.2.9. Extraction

A module *extracted* from an OMS, written O **extract** Σ, where Σ is a sub-signature of $\mathsf{Sig}(O)$. The extracted module is a subOMS of O with signature larger than (or equal to) Σ, such that O is a conservative extension of the extracted module. Intuitively, a module (in the sense of module extraction) is a small sub-OMS that says the same about Σ as the OMS O itself. For example, we can extract from GALEN a module referring to drugs, joints and body parts:

```
logic OWL
ontology myGalen =
   <http://example.org/GALEN/galen.owl>
 extract Drugs, Joints, Bodyparts
end
```

(This example is continued in section 3.3.5).

The semantics of $O = O'$ **extract** Σ is

- $\mathbf{Inst}(O) = \mathbf{Inst}(O')$
- $\mathbf{Sign}(O) = \Sigma \cup \Sigma'$
- $\mathbf{Ax}(O) = \Delta'$
- $\mathbf{Mod}(O)$ is the class of $\mathbf{Ax}(O)$-models

where $\langle \Sigma', \Delta' \rangle$ is the smallest depleting Σ-module [22], i.e. the smallest[18] sub-theory $\langle \Sigma', \Delta' \rangle$ of $(\mathbf{Sign}(O'), \mathbf{Ax}(O'))$ such that the following model-theoretic inseparability holds

$$\mathbf{Ax}(O') \setminus \Delta' \equiv_{\Sigma' \cup \Sigma} \emptyset.$$

This means intuitively that $\mathbf{Ax}(O') \setminus \Delta'$ cannot be distinguished from $\emptyset$ (what $\Sigma' \cup \Sigma$ concerns) and formally that

$$\begin{array}{rl} & \{M|_{\Sigma' \cup \Sigma} \mid M \in \mathbf{Mod}(\mathbf{Sign}(O')), M \models \mathbf{Ax}(O') \setminus \Delta'\} \\ = & \{M|_{\Sigma' \cup \Sigma} \mid M \in \mathbf{Mod}(\mathbf{Sign}(O'))\}. \end{array}$$

Dually, O **remove** Σ extracts w.r.t. the signature $\mathsf{Sig}(O) \setminus \Sigma$,[19] i.e. the semantics is given by that of O **extract** $\mathsf{Sig}(O) \setminus \Sigma$.

[18] If the smallest such sub-theory does not exist, the semantics is undefined. In [22], it is shown that it does exist in usual institutions.

[19] Note that the resulting module can still contain symbols from Σ, because the resulting signature may be enlarged.

3.2.10. Combination

A *combination* of OMS, written **combine** N, where N is a network. The simplest example of a combination is a disjoint union (we here translate OWL OMS into many-sorted OWL in order to be able to distinguish between different universes of individuals):

```
ontology Publications1 =
  Class: Publication
  Class: Article SubClassOf: Publication
  Class: InBook SubClassOf: Publication
  Class: Thesis  SubClassOf: Publication
  ...

ontology Publications2 =
  Class: Thing
  Class: Article SubClassOf: Thing
  Class: BookArticle SubClassOf: Thing
  Class: Publication SubClassOf: Thing
  Class: Thesis  SubClassOf: Thing
  ...

logic MS-OWL

network Publications_Network =
 1 : Publications1 with translation OWL2MS-OWL,
 2 : Publications2 with translation OWL2MS-OWL
end

ontology Publications_Combined =
combine
  Publications_Network
  %% implicitly: Article ↦ 1:Article ...
  %%             Article ↦ 2:Article ...
end
```

If mappings or alignments are present, the semantics of a combination is a quotient of a disjoint union (symbols related along the edges are identified). Technically, this is a colimit, see [48, 7]. An example for this is given along with the examples for alignments below.

The semantics of $O =$ **combine** N is

- $\mathbf{Inst}(O) = I$
- $\mathbf{Sign}(O) = \Sigma$, where $(I, \Sigma, \{\mu_i\}_{i \in |G|})$ is the colimit of the graph G given by the semantics of N
- $\mathbf{Ax}(O) = \cup_{i \in |G|} \mu_i(\mathbf{Ax}(O_i))$, where O_i is the OMS label of the node i in G

- $\mathbf{Mod}(O) = \{M \in \mathbf{Mod}(\Sigma) \mid M|_{\mu_i} \in \mathbf{Mod}(O_i), i \in |G|\}$, where O_i is the OMS label of the node i in G.

3.2.11. Minimisation

A *minimisation* of an OMS imposes a closed-world assumption on part of the OMS. It forces the non-logical symbols declared in O to be interpreted in a minimal way. This is written **minimize { O }**. Symbols declared before the minimised part are considered to be fixed for the minimisation (that is, we minimise among all models with the same reduct). Symbols declared after the minimisation can be varied. This is borrowed from circumscription [29, 3]. Alternatively, the non-logical symbols to be minimised and to be varied can be explicitly declared: O **minimize** Σ_1 **vars** Σ_2. For example, in the following OWL theory, B2 is a block that is not abnormal, because it is not specified to be abnormal, and hence it is also on the table.

```
  Class: Block
  Individual: B1 Types: Block
  Individual: B2 Types: Block DifferentFrom: B1
then minimize {
       Class: Abnormal
       Individual: B1 Types: Abnormal }
then
  Class: OnTable
  Class: BlockNotAbnormal EquivalentTo:
    Block and not Abnormal SubClassOf: OnTable
then %implied
  Individual: B2 Types: OnTable
```

The semantics of $O =$ **minimize** O' is

- $\mathbf{Inst}(O) = \mathbf{Inst}(O')$
- $\mathbf{Sign}(O) = \mathbf{Sign}(O')$
- $\mathbf{Mod}(O) = \{M \in \mathbf{Mod}(O') \mid M \text{ is minimal in } \mathbf{Mod}(O')\}$
- $\mathbf{Ax}(O)$ is undefined.

Note that for minimality we need the inclusions in model categories. Dually to minimisations, there are also maximisations.

3.2.12. Hide vs. Remove vs. Forget vs. Reject

We have four ways of removing the class `VegetarianTopping` from the ontology `Pizza` using the keywords **hide**, **remove**, **forget**, and **reject**, respectively. Table 1 illustrates some of the connections between (3.2.6)–(3.2.9).

Using hiding, we keep the model class of `Pizza`, but just remove the interpretation of `VegetarianTopping` from each model. Note that the resulting ontology has

```
VegetarianPizza SubClassOf:
  Annotations: dol:iri (*)
  some hasTopping { TomatoTopping, MozzarellaTopping, ... }
```

as a logical consequence. This is also a consequence of the corresponding uniform interpolant

```
Pizza forget VegetarianTopping
```

which captures the theory of `Pizza hide VegetarianTopping`. Note that there is a subtle difference between (model-theoretic) hiding and (consequence-theoretic) forgetting: a model satisfying the *theory* of O **hide** Σ might itself not be a model of O **hide** Σ. In examples involving "**keep** L", the uniform interpolant can be weaker than the hiding, because it is only required to have the same logical consequences in some language L, and a formula like (*) might not be a formula of L. Also, an extracted module does not contain (*), because it only selects a subontology, and `Pizza` does not contain (*). Finally, `Pizza reject VegetarianTopping` simply drops all sentences involving `VegetarianTopping`, and therefore also consequences like (*) are lost.

Note that while forget/keep and hide/reveal both work w.r.t. smaller signatures and sublogics, remove/extract and select/reject do not work for sublogics. This is because remove/extract must always respect the conservative extension property, which may not be possible when projecting to a sublogic. And if conservativity cannot be guaranteed, then forget/keep can be used in any case. In the case of select/reject, it is unclear what selecting of a sublogic should bring other than projecting to the sublogic using **hide along**.

	hide/reveal	remove/extract	forget/keep	select/reject
semantic background	model reduct	conservative extension	uniform interpolation	theory filtering
relation to original	interpretable	subtheory	interpretable	subtheory
approach	model level	theory level	theory level	theory level
type of OMS	elusive	flattenable	flattenable	flattenable
signature of result	$= \Sigma$	$\geq \Sigma$	$= \Sigma$	$\geq \Sigma$
change of logic	possible	not possible	possible	not possible
application	specification	ontologies	ontologies	blending

TABLE 1. Hiding – Extraction – Approximation – Filtering

Proposition 3.1. *The following relations among the constructs in Table* 1 *hold:*

$$\begin{array}{rl} & \mathbf{Mod}(O \textbf{ \textit{hide}} \ \Sigma) \\ = & \mathbf{Mod}(O \textbf{ \textit{remove}} \ \Sigma)|_{Sig(O)\setminus\Sigma} \\ \subseteq & \mathbf{Mod}(O \textbf{ \textit{forget}} \ \Sigma) \\ \subseteq & \mathbf{Mod}(O \textbf{ \textit{reject}} \ \Sigma) \end{array}$$

3.3. OMS Mappings

OMS mappings are generated by the following grammar:

```
MappingDefn ::= interpretation NAME : OMS to OMS = σ
              | entailment NAME = OMS entails OMS
              | equivalence NAME : OMS <-> OMS  = ⟨Σ, Δ⟩
              | conservative extension NAME = O1 of O2 for Σ
              | alignment NAME CARD1 CARD2 : OMS to OMS = Correspondences
              | refinement NAME : OMS to OMS = σ
              | refinement NAME = NAME then NAME
```

The semantics of an OMS mapping is given as a graph whose nodes N are labeled with

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

and whose edges are labeled with signature morphisms between the signatures of the source and target nodes. The theory of a node corresponding to an elusive OMS may be undefined. The class of models of a node corresponding to a flattenable OMS is the class of models of $\mathbf{Ax}(N)$. For brevity, we may write the label of a node as a tuple. We make the simplifying assumption that any OMS is assigned a unique name. The theory-level semantics of an OMS is needed for alignments.

In the following we discuss the different types of OMS mappings.

3.3.1. Interpretation

Theory *interpretations*, written **interpretation** $Id : O_1$ **to** $O_2 = \sigma$, expressing that the σ-reduct of each model of O_2 is a model of O_1. Instead of σ, an institution comorphism can be referred to. For example, we can express that the natural numbers are a total order as follows:

```
interpretation i : TotalOrder to Nat = Elem ↦ Nat
```

Here is a more complex example in Common Logic from the COLORE repository [9]:

```
interpretation geometry_of_time %mcons :
 %% Interpretation of linearly ordered time intervals...
  int:owltime_le
```

```
%% ... that begin and end with an instant as lines
%% that are incident with linearly ...
 to { ord:linear_ordering and bi:complete_graphical
    %% ... ordered points in a special geometry, ...
      and  int:mappings/owltime_interval_reduction }
 = int:ProperInterval ↦ int:Interval end
```

The semantics of **interpretation** N : O_1 **to** $O_2 = \sigma$ is defined iff σ is a signature morphism from $\mathbf{Sign}(O_1)$ to $\mathbf{Sign}(O_2)$ such that for each $M_2 \in \mathbf{Mod}(O_2)$, $M_2|_\sigma \in \mathbf{Mod}(O_1)$. In that case, the graph of N is $(O_1, \mathbf{Inst}(O_1),$ $\mathbf{Sign}(O_1), \mathbf{Mod}(O_1), \mathbf{Ax}(O_1)) \xrightarrow{\sigma} (O_2, \mathbf{Inst}(O_2), \mathbf{Sign}(O_2), \mathbf{Mod}(O_2), \mathbf{Ax}(O_2))$

3.3.2. Refinement

Refinements, written **refinement** Id : O_1 **to** $O_2 = \sigma$, expressing that O_2 is an acceptable realisation of O_1. Semantically, this is equivalent with a theory interpretation from O_1 to O_2 along σ. Refinements can be combined using the **then** keyword, as in the example below, where the requirement of implementing a monoid is refined to implementing the monoid of natural numbers with addition, using the representation of numbers as lists of binary digits, for efficiency:

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

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

spec Nat =
  NatWithSuc hide suc
end

refinement R1 =
Monoid refined via Elem |-> Nat to Nat
end

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

ops __+__ , __++__ : Bin * Bin -> Bin
forall x, y : Bin
```

```
 .  0 0 = 0  .  0 1 = 1
 .  not  (0 = 1)  .  x 0 = y 0 => x = y
 .  not  (x 0 = y 1)  .  x 1 = y 1 => x = y
 .  0 + 0 = 0  .  0 ++ 0 = 1
 .  x 0 + y 0 = (x + y) 0  .  x 0 ++ y 0 = (x + y) 1
 .  x 0 + y 1 = (x + y) 1  .  x 0 ++ y 1 = (x ++ y) 0
 .  x 1 + y 0 = (x + y) 1  .  x 1 ++ y 0 = (x ++ y) 0
 .  x 1 + y 1 = (x ++ y) 0  .  x 1 ++ y 1 = (x ++ y) 1
end

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

refinement R3 = R1 then R2
```

The semantics of **refinement** $R: \ O_1$ **to** $O_2 = \sigma$ is defined iff σ is a signature morphism from $\mathbf{Sign}(O_1)$ to $\mathbf{Sign}(O_2)$ such that for each $M_2 \in \mathbf{Mod}(O_2)$, $M_2|_\sigma \in \mathbf{Mod}(O_1)$. In that case, the graph of N is $(O_1, \mathbf{Inst}(O_1), \mathbf{Sign}(O_1), \mathbf{Mod}(O_1), \mathbf{Ax}(O_1)) \xrightarrow{\sigma} (O_2, \mathbf{Inst}(O_2), \mathbf{Sign}(O_2), \mathbf{Mod}(O_2), \mathbf{Ax}(O_2))$

The semantics of R_1 **then** R_2 is defined if and only if the semantics of R_1 is $(N_1, I_1, \Sigma_1, \mathcal{M}_1, \Delta_1) \xrightarrow{\sigma_1} (N_2, I_2, \Sigma_2, \mathcal{M}_2, \Delta_2)$, the semantics of R_2 is $(N_1', I_1', \Sigma_1', \mathcal{M}_1', \Delta_1') \xrightarrow{\sigma_2} (N_2', I_2', \Sigma_2', \mathcal{M}_2', \Delta_2')$, such that $I_2 = I_1'$, $\Sigma_2 = \Sigma_1'$ and $\mathcal{M}_1' \subseteq \mathcal{M}_2$, and then the graph of the composition is $(N_1'', I_1, \Sigma_1, \{M|_{\sigma_1;\sigma_2} \mid M \in \mathcal{M}_2'\}, \bot) \xrightarrow{\sigma_1;\sigma_2} (N_2', I_2', \Sigma_2', \mathcal{M}_2', \Delta_2')$, where N_1'' is a new name.

3.3.3. Entailment

Entailments, written **entailment** $Id = O_1$ **entails** O_2, express that O_2 is logically entailed by O_1. For example, we can express that in a group, the inverse of an element still exists after hiding the explicit inverse operation from the specification as follows:

```
logic CASL

spec InterpolatedGroup =
  sort Elem
  ops 0:Elem; __+__:Elem*Elem->Elem; inv:Elem->Elem
  forall x, y, z: Elem . x+0=x
                       . x+(y+z) = (x+y)+z
                       . x+inv(x) = 0
  forget inv
end

entailment ent = InterpolatedGroup
```

```
  entails { . forall x:Elem . exists y . Elem . x+y=0 }
end
```

The semantics of **entailment** $N = O_1$ **entails** O_2 is defined iff $\mathbf{Sign}(O_1) = \mathbf{Sign}(O_2)$ and $\mathbf{Mod}(O_1) \models \mathbf{Ax}(O_2)$. In that case, the graph of N is $(O_1, \mathbf{Inst}(O_1), \mathbf{Sign}(O_1), \mathbf{Mod}(O_1), \mathbf{Ax}(O_1)) \stackrel{id}{\to} (O_2, \mathbf{Inst}(O_2), \mathbf{Sign}(O_2), \mathbf{Mod}(O_2), \mathbf{Ax}(O_2))$

3.3.4. OMS Equivalence

OMS *equivalences*, written **equivalence** $Id : O_1 \leftrightarrow O_2 = O_3$, expressing that O_1 and O_2 have model classes that are in bijective correspondence. This is done by providing a (fragment) OMS O_3 such that O_i **then** O_3 is a definitional extension [27]. For example, Boolean algebras are equivalent to Boolean rings:

```
equivalence e : algebra:BooleanAlgebra ↔ algebra:BooleanRing =
forall x, y : Elem
. x ∧ y = x*y
. x ∨ y = x + y + x*y
. ¬x = 1 + x
. x*y = x ∧ y,
. x+y = (x ∨ y) ∧ ¬(x ∧ y).
end
```

The semantics of **equivalence** $N : O_1 \leftrightarrow O_2 = O_3$ is defined iff for each model $M_i \in \mathbf{Mod}(O_i)$ there exists a unique model $M \in \mathbf{Mod}_{(\mathbf{Sign}(O_1) \cup \mathbf{Sign}(O_2), \emptyset)}(O_3)$ such that $M|_{\mathbf{Sign}(O_i)} = M_i$. In that case, the graph of N is $(O_1, I, \mathbf{Sign}(O_1), \mathbf{Mod}(O_1), \mathbf{Ax}(O_1)) \stackrel{\iota_1}{\to} (O_3, I, \mathbf{Sign}_{(\mathbf{Sign}(O_1) \cup \mathbf{Sign}(O_2), \emptyset)}(O_3), \mathbf{Mod}_{(\mathbf{Sign}(O_1) \cup \mathbf{Sign}(O_2), \emptyset)}(O_3), \mathbf{Ax}_{(\mathbf{Sign}(O_1) \cup \mathbf{Sign}(O_2), \emptyset)}(O_3)) \stackrel{\iota_2}{\leftarrow} (O_2, I, \mathbf{Sign}(O_2), \mathbf{Mod}(O_2), \mathbf{Ax}(O_2))$ where ι_i are inclusions.

3.3.5. Conservative Extension

A *conservative extension* is written **conservative extension** $Id\ c : O_1$ **of** O_2 **for** Σ. This expresses that O_2 contains all knowledge about the signature Σ from the O_1 or, more precisely, O_1 is a conservative extension of O_2 with restriction signature Σ and conservativity c. If c is %mcons, this means that every Σ-reduct of an O_2-model can be expanded to an O_1-model. If c is %ccons, this means that every Σ-sentence φ following from O_1 already follows from O_2. This relation shall hold for any module O_2 extracted from O_1 using the **extract** construct. For example, we can specify that we obtained a module of GALEN by extracting the parts corresponding to drugs, joints and body parts as follows:

```
module myGalenIsAModule : myGalen of
  <http://example.org/GALEN/galen.owl>
```

```
  for Drugs, Joints, Bodyparts
end
```

The semantics of **conservative extension** N $c : O_1$ **of** O_2 **for** Σ is defined iff $\Sigma \subseteq \mathbf{Sign}(O_2) \subseteq \mathbf{Sign}(O_1)$ and if $c = \%$mcons and for each $M \in \mathbf{Mod}(O_2)$ there is a model $M' \in \mathbf{Mod}(O_1)$ such that $M'|_\Sigma = M|_\Sigma$, or if $c = \%$ccons and for each $\varphi \in \mathbf{Sen}(\Sigma)$, $O_1 \models \varphi$ implies $O_2 \models \varphi$. Then the graph of N is $(O_2, \mathbf{Inst}(O_2), \mathbf{Sign}(O_2), \mathbf{Mod}(O_2), \mathbf{Ax}(O_2)) \xrightarrow{\iota} (O_1, \mathbf{Inst}(O_1), \mathbf{Sign}(O_1), \mathbf{Mod}(O_1), \mathbf{Ax}(O_1))$, with ι being the inclusion.

3.3.6. Alignment definition

Alignment definitions, written **alignment** *Id* $card_1$ $card_2 : O_1$ **to** $O_2 = c_1, \ldots, c_n$, **assuming** *domain* where $card_1$ resp. $card_2$ specify constraints on the alignment relation concerning the source resp. target. Each $card_i$ is one of 1, ?, +, * ('1' for injective and total, '+' for total, '?' for injective and '*' for none). The c_j are correspondences of form sym_1 *rel conf* sym_2. Here, sym_i is a symbol from O_i, *rel* is one of the built-in relations $>$, $<$, $=$, $\%$, $\ni$, $\in$, $\mapsto$, or an identifier of a relation specified externally, and *conf* is an (optional) confidence value between 0 and 1. The user can specify the assumption about the universe where the relations in the correspondences are interpreted using the *assuming* clause, with possible values **SingleDomain** (all ontologies are interpreted over the same universe, which is also the default), **GlobalDomain** (the domains of the ontologies are reconciled w.r.t. a global domain of interpretation) and **ContextualizedDomain** (the domains are connected via relations). This syntax of alignments follows the Alignment API [10].[20] If all correspondences of an alignment have the confidence value 1, the alignment can be given a formal semantics as a network.

```
ontology Onto1 =
  Class: Person
  Class: Woman SubClassOf: Person
  Class: Bank
end

ontology Onto2 =
  Class: HumanBeing
  Class: Woman SubClassOf: HumanBeing
  Class: Bank
end

alignment VAlignment : Onto1 to Onto2 =
```

[20] Note that BioPortal's [40] mappings are correspondences in the sense of the Alignment API and hence of DOL. BioPortal only allows users to collect correspondences, but not to group them into alignments. In a sense, for each pair of ontologies, all BioPortal users contribute to a big alignment between these.

```
  Person = HumanBeing,
  Woman = Woman
end

network N =
 1 : Onto1,
 2 : Onto2,
 VAlignment

ontology VAlignedOntology =
  combine N
  %% 1:Person is identified with 2:HumanBeing
  %% 1:Woman is identified with 2:Woman
  %% 1:Bank and 2:Bank are kept distinct
end

ontology VAlignedOntologyRenamed =
  VAlignedOntology with 1:Bank ↦ RiverBank, 2:Bank ↦ FinancialBank,
                        Person_HumanBeing ↦ Person
end
```

We sketch the semantics of alignments with the case when the domain of interpretation is assumed to be shared by the ontologies being aligned. In this case, the semantics is given by a W-shaped graph like in Fig. 2 where O_1 and O_2 are the nodes of the ontologies being aligned, O_1' and O_2' collect the symbols of O_1 and O_2, respectively, that appear in the correspondences of the alignment, ι_1 and ι_2 are inclusions and the bridge ontology B together with the morphisms σ_1 and σ_2 is constructed by turning the correspondences into bridge axioms. Details can be found in [8].

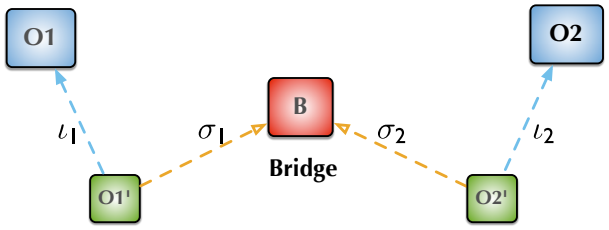

FIGURE 2. Semantics of alignments

3.4. Networks of OMS

OMS networks are introduced by the following grammar:

```
NetworkDefn := network NAME = Network
Network ::= NAME* [ excluding NAME* ]
```

Here, the NAMEs can name OMS, mappings or other networks. A network is specified as a list of network elements (OMS, OMS mappings and sub-networks), followed by an optional list of excluded network elements. For disambiguating the symbols in the combination of the network, the individual OMS can be prefixed with labels, like $n : O$, which are scoped to the current OMS network. An example has already been presented in the section on alignments. Together with two OMS included in the network, the graph of the network implicitly includes all paths along importations between the two nodes. For example, in the example below, NAT_PLUS imports the specification NAT. Without the implicit inclusion of this import, the combination would duplicate the theory of NAT.

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

spec NAT_PLUS =
 NAT
  then
 op __+__ : Nat * Nat -> Nat
 ...
end

...

network N =
  NAT, NAT_PLUS,...
end

spec N_COMBINED =
  combine N
end
```

Formally, the graph of a network is constructed by taking the union of all graphs of its constituents, provided that we regard the semantics of OMS as a graph with one node and no edges, and removing from it all subparts specified in the **excluding** clause.

3.5. Libraries of OMS

Libraries start with the keyword **library** and the name of the library, followed by a *qualification* choosing the OMS *language*, *logic*, and/or *serialisation*. This is followed by a list of imports of other libraries, definitions of OMS, OMS mappings, networks of OMS, or other qualifications which change the current logic. Optionally, a prefix map placed at the beginning of a library may be used to abbreviate IRIs. A library can also be the inclusion of an OMS written in a language corresponding to some institution.

```
%prefix(
        bfo: <https://bfo.googlecode.com/svn/releases/1.1.1/>
       )%

library Parthood

logic CommonLogic

ontology BFOWithAssociatedAxioms =
   bfo:bfo.owl with translation OWL22CommonLogic
  then
   (forall (x y) (if (snap:properTemporalPartOf x y)
                     (exists (z) (and (snap:properTemporalPartOf z y)
                     (not (exists (w)
                               (and (snap:temporalPartOf w x) (snap:temporalPartOf w z)
    )))))))
end
```

Note that the prefixes declared in an imported library are available in the imported library, as illustrated in the example above with the prefix `snap:`.

This completes our overview of DOL. The full syntax and semantics of DOL will be available at `wiki.ontohub.org` and has been submitted to OMG for standardisation. The most recent version of the document is available at `ontoiop.org`.

4. Tool Support for DOL

Currently, DOL is supported by two tools: Ontohub and the Heterogeneous Tool Set (HETS). Ontohub (see `http://ontohub.org`) is a web-based repository engine for OMS that are written either in DOL or in some specific OMS language.[21]

Ontohub provides means for organising OMS into repositories. The distributed nature enables communities to share and exchange their contributions easily. The heterogeneous nature makes it possible to integrate OMS written in various OMS languages. Ontohub supports a wide range of DOL-conforming OMS languages building on DOL and also supports DOL's interpretations, equivalences and alignments. Users of Ontohub can upload, browse, search and annotate OMS and OMS libraries in various languages via a web front end. Figure 3 shows an excerpt of the 25 logics currently available in Ontohub.

The parsing and inference back end is the Heterogeneous Tool Set (Hets [34, 39], available at `hets.eu`). Hets supports a large number of basic OMS languages and logics, as well as the DOL metalanguage as described in this paper.[22]

[21] Ontohub's sources are freely available at `https://github.com/ontohub/ontohub`.
[22] Some (but only few) of DOL's features are still being implemented at the time of the writing of this paper.

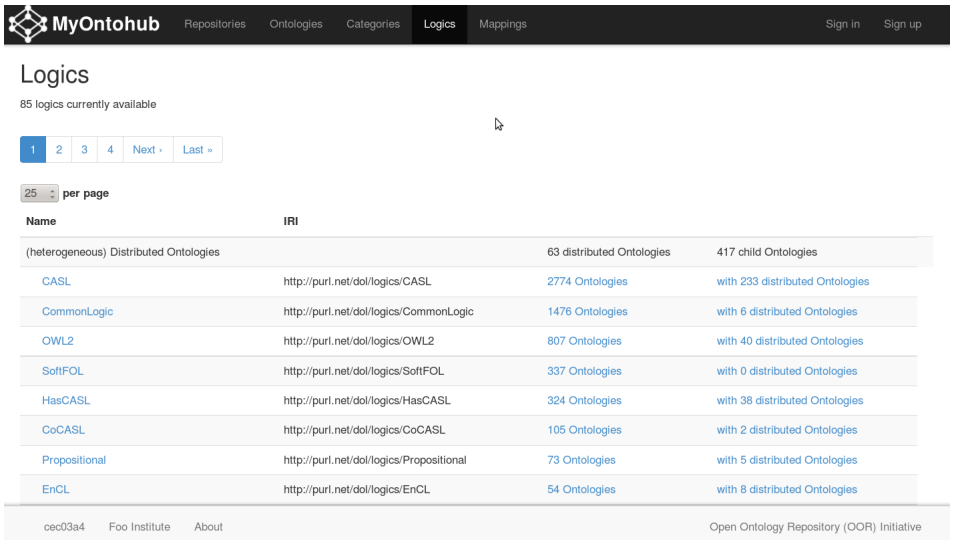

FIGURE 3. Overview of logics in Ontohub

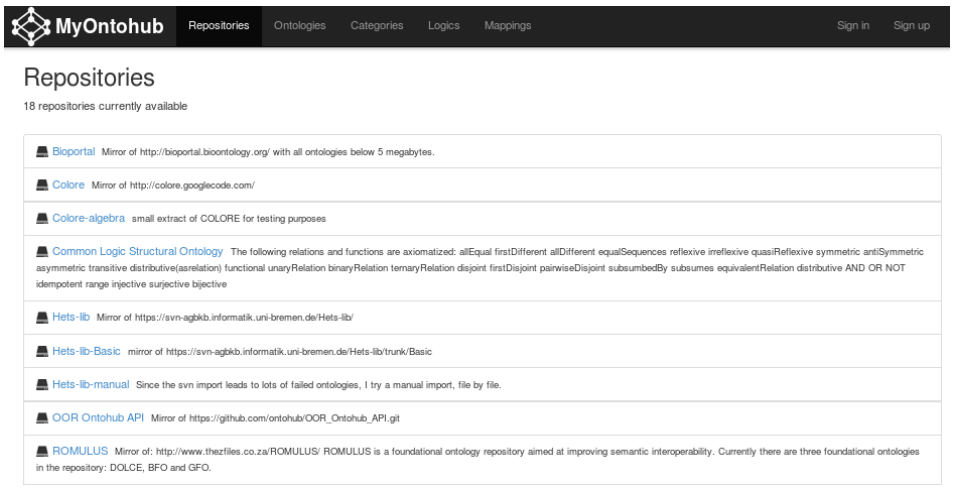

FIGURE 4. Some of the repositories hosted on Ontohub

The structural information extracted from DOL OMS by Hets is stored in the Ontohub database and exposed to human users via a web interface and to machine clients as linked data.[23]

[23] "Linked data" is a set of best practises for publishing structured data on the Web in a machine-friendly way [1]. DOL and Ontohub conform with linked data.

5. Conclusion and Future Work

Interoperability between systems as well as reusability, we argued in the introduction to this paper, are critical challenges.

We here proposed to address these challenges by introducing two abstractions: firstly, we introduced the notion of OMS, spanning formalised ontologies, models, and specifications; secondly, we introduced the DOL language, an abstraction in the sense that it provides a structuring, module, and mapping language independently of the particular logical formalism used.

The work presented here brings together previous work pursued in a number of communities, including in particular logical pluralism, modular ontologies, algebraic specification, and modelling of systems. It therefore combines many isolated logical modelling and specification solutions into one coherent framework with formal semantics.

A number of open problems and challenges, however, remain:

- What is a suitable abstract meta framework for non-monotonic logics and rule languages such as RIF and RuleML? Are institutions suitable here? Are the modularity questions for these languages different from those for monotonic logics?
- What is a useful abstract notion of OMS query (language)? How to handle answer substitutions in a logic-agnostic way?
- Can the notions of class hierarchy and of satisfiability of a class be generalised from OWL to other languages?
- Can logical frameworks be used for the specification of OMS languages and translations?

Despite these challenges, we hope that the development of DOL will have a profound impact on ontology engineering practices as well as on the way the modelling, ontology, and specification communities interact and how the systems they develop may interoperate. The impact on communities can already be seen e.g. by the use of Ontohub/DOL for the FOIS 2014 ontology competition. We have illustrated the benefits of DOL for a wide range of use cases; including for a framework of heterogeneous modelling in UML [20, 21], in biomedical ontology [26], for the specification of blending diagrams in computational creativity [23], and for the heterogeneous modelling of musical harmonies [6].

We hope that the future will bring many more diverse and interesting use cases for the DOL language.

Acknowledgements

The development of DOL is supported by the German Research Foundation (DFG), Project I1-[OntoSpace] of the SFB/TR 8 “Spatial Cognition”.

The project COINVENT acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open Grant number: 611553.

The authors would like to thank the OntoIOp working group for their valuable input, particularly Michael Grüninger, Maria Keet, Christoph Lange, and Peter Yim.

We also want to thank Yazmin Angelica Ibañez, Thomas Schneider and Carsten Lutz for valuable input on interpolation and module extraction.

References

1. Berners-Lee, T. Design Issues: Linked Data. July 27, 2006. http://www.w3.org/DesignIssues/LinkedData.html (visited on 2010-01-20).
2. Bidoit, M. and Mosses, P. D. *CASL User Manual*. LNCS (IFIP Series) 2900. Freely available at http://www.cofi.info. Berlin, Heidelberg: Springer, 2004.
3. Bonatti, P. A., Lutz, C., and Wolter, F. The Complexity of Circumscription in DLs. In *J. Artif. Intell. Res. (JAIR) 35* (2009), pp. 717–773.
4. Borzyszkowski, T. Logical systems for structured specifications. In *Theoretical Computer Science 286* (2002), pp. 197–245.
5. Burstall, R. M. and Goguen, J. A. The Semantics of CLEAR, A Specification Language. In *Abstract Software Specifications, 1979 Copenhagen Winter School, January 22 - February 2, 1979, Proceedings*. Ed. by D. Bjørner. Vol. 86. Lecture Notes in Computer Science. Springer, 1979, pp. 292–332. http://dx.doi.org/10.1007/3-540-10007-5_41.
6. Cambouropoulos, E., Kaliakatsos-Papakostas, M., Kühnberger, K.-U., Kutz, O., and A.Smaill. Concept invention and music: Creating novel harmonies via conceptual blending. In *Proc. of the 9th Int. Conference on Interdisciplinary Musicology (CIM-2014)*. Berlin, 2014.
7. Codescu, M. and Mossakowski, T. Heterogeneous colimits. In *MoVaH'08 Workshop on Modeling, Validation and Heterogeneity*. Ed. by F. Boulanger, C. Gaston, and P.-Y. Schobbens. IEEE press, 2008. http://www.computer.org/portal/web/csdl/abs/proceedings/icstw/2008/3388/00/3388toc.htm.
8. Codescu, M., Mossakowski, T., and Kutz, O. A Categorical Approach to Ontology Alignment. In *Proc. of the 9th International Workshop on Ontology Matching (OM-2014), ISWC-2014, Riva del Garda, Trentino, Italy*. CEUR-WS online proceedings. 2014.
9. COLORE. An open repository of first-order ontologies represented in Common Logic. http://colore.googlecode.com.
10. David, J., Euzenat, J., Scharffe, F., and dos Santos, C. T. The Alignment API 4.0. In *Semantic Web 2*, 1 (2011), pp. 3–10.
11. Diaconescu, R., Goguen, J., and Stefaneas, P. Logical Support for Modularisation. In *2nd Workshop on Logical Environments*. New York: CUP, 1993, pp. 83–130.

12. Goguen, J. A. and Burstall, R. M. Institutions: Abstract Model Theory for Specification and Programming. In *Journal of the Association for Computing Machinery 39* (1992). Predecessor in: LNCS 164, 221–256, 1984., pp. 95–146.
13. Goguen, J. and Roşu, G. Institution morphisms. In *Formal aspects of computing 13* (2002), pp. 274–307.
14. Goguen, J. A. and Rosu, G. Composing Hidden Information Modules over Inclusive Institutions. In *From Object-Orientation to Formal Methods, Essays in Memory of Ole-Johan Dahl*. Ed. by O. Owe, S. Krogdahl, and T. Lyche. Vol. 2635. Lecture Notes in Computer Science. Springer, 2004, pp. 96–123. http://dx.doi.org/10.1007/978-3-540-39993-3_7.
15. Grau, B. C., Honavar, V., Schlicht, A., and Wolter, F., eds. Proceedings of the 2nd International Workshop on Modular Ontologies, WoMO 2007, Whistler, Canada, October 28, 2007. Vol. 315. CEUR Workshop Proceedings. CEUR-WS.org, 2008.
16. Haase, P., Honavar, V., Kutz, O., Sure, Y., and Tamilin, A., eds. Proceedings of the 1st International Workshop on Modular Ontologies, WoMO'06, co-located with the International Semantic Web Conference, ISWC'06 November 5, 2006, Athens, Georgia, USA. Vol. 232. CEUR Workshop Proceedings. CEUR-WS.org, 2007.
17. The NeOn Ontology Engineering Toolkit. http://www.neon-project.org/. 2008. http://watson.kmi.open.ac.uk/Downloads%20and%20Publications_files/neon-toolkit.pdf.
18. Hayes, P. RDF Semantics. W3C Recommendation. World Wide Web Consortium (W3C), Feb. 10, 2004. http://www.w3.org/TR/2004/REC-rdf-mt-20040210/.
19. Horridge, M. Protégé OWL Tutorial. Version v1.3. Mar. 24, 2011. http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/.
20. Knapp, A., Mossakowski, T., and Roggenbach, M. Towards an Institutional Framework for Heterogeneous Formal Development in UML - A Position Paper. In *Software, Services and Systems. Essays Dedicated to Martin Wirsing on the Occasion of His Emeritation*. Ed. by R. D. Nicola and R. Hennicker. Vol. 8950. Lecture Notes in Computer Science. Springer-Verlag Berlin Heidelberg, 2015.
21. Knapp, A., Mossakowski, T., Roggenbach, M., and Glauer, M. An Institution for Simple UML State Machines. In *CoRR abs/1411.4495* (2014). http://arxiv.org/abs/1411.4495.
22. Kontchakov, R., Wolter, F., and Zakharyaschev, M. Logic-based ontology comparison and module extraction, with an application to DL-Lite. In *Artif. Intell. 174*, 15 (2010), pp. 1093–1141. http://dx.doi.org/10.1016/j.artint.2010.06.003.
23. Kutz, O., Bateman, J., Neuhaus, F., Mossakowski, T., and Bhatt, M. E pluribus unum: Formalisation, Use-Cases, and Computational Support for Conceptual Blending. In *Computational Creativity Research: Towards Creative Machines*. Ed. by T. R Besold, M. Schorlemmer, and A. Smaill. Thinking Machines. Atlantis/Springer, 2014.
24. Kutz, O., Hois, J., Bao, J., and Cuenca Grau, B., eds. *Modular Ontologies—Proceedings of the Fourth International Workshop (WoMO 2010)*. Vol. 210. Frontiers in Artificial Intelligence and Applications. Toronto, Canada: IOS Press, 2010.
25. Kutz, O., Lücke, D., Mossakowski, T., and Normann, I. The OWL in the CASL—Designing Ontologies Across Logics. In *OWL: Experiences and Directions,*

5th International Workshop (OWLED-08). Ed. by C. Dolbear, A. Ruttenberg, and U. Sattler. co-located with ISWC-08, Karlsruhe, Germany, October 26–27: CEUR-WS, Vol-432, 2008.

26. Kutz, O., Mossakowski, T., Hastings, J., Castro, A. G., and Sojic, A. Hyper-ontology for the Biomedical Ontologist: A Sketch and Some Examples. In *Workshop on Working with Multiple Biomedical Ontologies (WoMBO at ICBO 2011)*. Buffalo, NY, USA, 2011.
27. Kutz, O., Mossakowski, T., and Lücke, D. Carnap, Goguen, and the Hyperontologies: Logical Pluralism and Heterogeneous Structuring in Ontology Design. In *Logica Universalis 4*, 2 (2010). Special issue on 'Is Logic Universal?'
28. Kutz, O. and Schneider, T., eds. *Modular Ontologies—Proceedings of the Fifth International Workshop (WoMO 2011)*. Vol. 230. Frontiers in Artificial Intelligence and Applications. IOS Press, 2011.
29. Lifschitz, V. Circumscription. In *Handbook of Logic in Artificial Intelligence and Logic Programming*. Vol. 3. Oxford University Press, 1994, pp. 297–352.
30. Lüttich, K., Masolo, C., and Borgo, S. Development of Modular Ontologies in CASL. In *WoMO*. Ed. by P. Haase, V. Honavar, O. Kutz, Y. Sure, and A. Tamilin. Vol. 232. CEUR Workshop Proceedings. CEUR-WS.org, 2006.
31. Lutz, C., Seylan, I., and Wolter, F. An Automata-Theoretic Approach to Uniform Interpolation and Approximation in the Description Logic EL. In *KR*. Ed. by G. Brewka, T. Eiter, and S. A. McIlraith. AAAI Press, 2012.
32. Lutz, C. and Wolter, F. Foundations for Uniform Interpolation and Forgetting in Expressive Description Logics. In *IJCAI*. 2011, pp. 989–995.
33. Meseguer, J. General Logics. In *Logic Colloquium '87*. Ed. by H. J. Ebbinghaus. North Holland, 1989, pp. 275–329.
34. Mossakowski, T. Hets: the Heterogeneous Tool Set. `http://hets.eu` (visited on 2012-12-10).
35. Mossakowski, T., Haxthausen, A., Sannella, D., and Tarlecki, A. CASL: The Common Algebraic Specification Language. In *Logics of Formal Specification Languages*. Ed. by M. H. D. Bjorner. Monographs in Theoretical Computer Science. Springer-Verlag Heidelberg, 2008. Chap. 3, pp. 241–298. `http://dx.doi.org/10.1007/978-3-540-74107-7_5`.
36. Mossakowski, T. and Kutz, O. The Onto-Logical Translation Graph. In *Modular Ontologies*. Ed. by O. Kutz and T. Schneider. IOS, 2011.
37. Mossakowski, T., Kutz, O., and Lange, C. Semantics of the distributed ontology language: Institutes and Institutions. In *Recent Trends in Algebraic Development Techniques, 21th International Workshop, WADT 2012*. Ed. by N. Martí-Oliet and M. Palomino. Vol. 7841. Lecture Notes in Computer Science. Springer, 2013, pp. 212–230. `http://link.springer.com/chapter/10.1007/978-3-642-37635-1_13`.
38. Mossakowski, T., Lange, C., and Kutz, O. Three Semantics for the Core of the Distributed Ontology Language. In *7th International Conference on Formal Ontology in Information Systems (FOIS)*. Ed. by M. Donnelly and G. Guizzardi. Vol. 239. Frontiers in Artificial Intelligence and Applications. FOIS Best Paper Award. IOS Press, 2012, pp. 337–352.
39. Mossakowski, T., Maeder, C., and Lüttich, K. The Heterogeneous Tool Set. In *TACAS 2007*. Ed. by O. Grumberg and M. Huth. Vol. 4424. Lecture Notes in Computer Science. Springer-Verlag Heidelberg, 2007, pp. 519–522.

40. Noy, N. F., Shah, N. H., Patricia L. Whetzel, ., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Rubin, D. L., Storey, M.-A., Chute, C. G., and Musen, M. A. BioPortal: ontologies and integrated data resources at the click of a mouse. In *Nucleic Acids Research 37* (2009). http://bioportal.bioontology.org, W170–W173.
41. Sannella, D. and Tarlecki, A. Specifications in an arbitrary institution. In *Information and Computation 76* (1988), pp. 165–210.
42. Sannella, D. and Tarlecki, A. *Foundations of Algebraic Specification and Formal Software Development*. EATCS Monographs in Theoretical Computer Science. Springer, 2012.
43. Sattler, U. and Tamilin, A., eds. Workshop on Ontologies: Reasoning and Modularity (WORM-08). Vol. Vol-348. (ESWC) Tenerife, Spain: CEUR Workshop Proceedings, 2008. http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-348/.
44. Schneider, M., Rudolph, S., and Sutcliffe, G. Modeling in OWL 2 without Restrictions. In *Proceedings of the 10th International Workshop on OWL: Experiences and Directions (OWLED 2013) co-located with 10th Extended Semantic Web Conference (ESWC 2013), Montpellier, France, May 26-27, 2013.* Ed. by M. Rodriguez-Muro, S. Jupp, and K. Srinivas. Vol. 1080. CEUR Workshop Proceedings. CEUR-WS.org, 2013. http://ceur-ws.org/Vol-1080/owled2013_14.pdf.
45. Schneider, T. and Walther, D., eds. Proc. of the 6h Int. Workshop on Modular Ontologies. Vol. 875. CEUR-WS, 2012.
46. Wang, Z., Wang, K., Topor, R. W., and Pan, J. Z. Forgetting for knowledge bases in DL-Lite. In *Ann. Math. Artif. Intell. 58*, 1–2 (2010), pp. 117–151.
47. Wirsing, M. Structured Algebraic Specifications: A Kernel Language. In *Theor. Comput. Sci. 42* (1986), pp. 123–249. http://dx.doi.org/10.1016/0304-3975(86)90051-4.
48. Zimmermann, A., Krötzsch, M., Euzenat, J., and Hitzler, P. Formalizing Ontology Alignment and its Operations with Category Theory. In *Proc. of FOIS-06*. 2006, pp. 277–288.

Till Mossakowski
Institute of Knowledge and Language Engineering,
Otto-von-Guericke-University Magdeburg, Germany
e-mail: mossakow@iws.cs.uni-magdeburg.de

Mihai Codescu
Institute of Knowledge and Language Engineering,
Otto-von-Guericke-University Magdeburg, Germany
e-mail: codescu@iws.cs.uni-magdeburg.de

Fabian Neuhaus
Institute of Knowledge and Language Engineering,
Otto-von-Guericke-University Magdeburg, Germany
e-mail: fneuhaus@iws.cs.uni-magdeburg.de

Oliver Kutz
Institute of Knowledge and Language Engineering,
Otto-von-Guericke-University Magdeburg, Germany
e-mail: okutz@iws.cs.uni-magdeburg.de