The OWL in the CASL
Designing Ontologies Across Logics

Oliver Kutz[1], Dominik Lücke[1], Till Mossakowski[1,2], and Immanuel Normann[3]

[1] SFB/TR 8 Spatial Cognition, University of Bremen, Germany
{okutz,luecke,till}@informatik.uni-bremen.de
[2] DFKI GmbH Bremen, Germany
Till.Mossakowski@dfki.de
[3] Department of Linguistics, University of Bremen, Germany
normann@uni-bremen.de

Abstract. In this paper, we show how the web ontology language OWL can be accommodated within the larger framework of the **het**erogeneous **c**ommon **a**lgebraic **s**pecification **l**anguage HETCASL. Through this change in perspective, OWL can benefit from various useful HETCASL features concerning structuring, modularity, and heterogeneity. This tackles a major problem area in ontology engineering: re-use of ontologies and re-combination of ontological modules. We discuss in particular: (1) the extension of the Manchester syntax for OWL with structuring mechanisms of CASL, allowing for explicit modularisation; (2) automatic translations between ontology languages to support ontology design across different ontology languages (heterogeneity); (3) heterogeneous ontology refinements, and corresponding automated reasoning support for different logics.

1 Introduction

Ontologies play an increasingly important role in various areas of Knowledge Representation, ranging from the life sciences and engineering domains to linguistic semantics. In the process, ontologies are being designed in a broad spectrum of logics, with considerably varying expressivity and supporting quite different reasoning methods.

Many (domain) ontologies are written in description logics like $\mathcal{SHOIN}(D)$ (underlying OWL-DL) and $\mathcal{SROIQ}(D)$ (underlying OWL 2.0). These logics are characterised by having a rather fine-tuned expressivity, exhibiting (still) decidable satisfiability problems, whilst being amenable to highly optimised implementations.

However, there are many cases where either weaker DLs are enough, such as sub-Boolean $\mathcal{EL}$, and more specialised (and faster) algorithms can be employed, or, contrarily, the expressivity has to be extended beyond the scope of standard description logics. An example for the former would be the NCI thesaurus (containing about 45.000 concepts) which is intended to become the reference terminology for cancer research [33], an example for the latter many foundational ontologies, for instance DOLCE [12] and GUM [3].

While the web ontology language OWL is being constantly refined and extended, its main target application is the Semantic Web and related areas, and it can thus not be expected to be fit for any purpose: there will always be new, typically interdisciplinary application areas for ontologies where the employed (or required) formal languages do not directly fit into the OWL landscape. *Heterogeneity* (of ontology languages) is thus clearly an important issue. This does not only include cases where the expressivity of OWL is

simply exceeded (such as when moving to full first-order logic), but, ultimately, also cases where combinations with or connections to formalism with different semantics have to be covered, such as temporal, spatial, or epistemic logics, cf. e.g. [1; 2; 24; 10; 5].

In this context, it can be a rather difficult task for an ontology designer to choose an appropriate logic and formalism for a specific ontology design beforehand—and failing in making the right choice might lead to the necessity of re-designing large parts of an ontology from scratch, or limit future expandability. Another issue is the mere size of ontologies making the design process potentially quite hard and error prone (at least for humans). This issue has been partly cured in OWL by the `imports` construct, but still leaves the problem of 'debugging' large ontologies as an important issue [19]. Also, simple operations such as the re-use of parts of an ontology in a different 'context' whilst *renaming* (parts of) the signature are not possible in the OWL languages.

We here propose a cure to the above issues based on the concept of *heterogeneity*: facing the fact that several logics and formalisms are used for designing ontologies, we suggest heterogeneous structuring constructs that allow to combine ontologies in various ways and in a systematic and formally and semantically well-founded way. Our approach is based on the theory of institutions and formal structuring techniques from algebraic specification theory (discussed in Sec. 2). Its main features are the following:

- The ontology designer can use description logics to specify most parts of an ontology, and can use first-order (or even higher-order) logic where needed. Moreover, the overall ontology can be assembled from (and can be split up into) semantically meaningful parts ('modules') that are systematically related by structuring mechanisms. These parts can then be re-used and/or extended in different settings.
- Institution theory provides 'logic translations' between different ontology languages, translating the syntax and semantics of different formalisms.
- Various concepts of 'ontological module' are covered, including simple imports (extensions) and union of theories, as well as conservative and definitional extensions.
- Structuring into modules is made explicit in the ontology and generates so-called proof obligations for conservativity. Proof obligations can also be used to keep track of desired consequences of an ontology (module), especially during the design process.
- Re-using (parts of) ontologies whilst renaming (parts of) the signature is handled by *symbol maps* and *hiding symbols*: essentially, this allows the internalisation of (strict) alignment mappings.
- The approach allows heterogeneous refinements: it is possible to prove that an ontology O_2 is a refinement of another ontology O_1, formalised in a different logic. For instance, one can check if a domain ontology is a refinement of (a part of) a foundational one. An interesting by-product of the definition of heterogeneous refinements is that it also provides a rather general definition of heterogeneous sub-ontology.

We have formalised several logics that are important from an ontology-design perspective as so-called institutions [13] and supply institution comorphisms as mappings between them, including the DL $\mathcal{SROIQ}(D)$ and many-sorted first-order logic (using the language CASL).

Tool support for developing heterogeneous ontologies is available via the Heterogeneous Tool Set HETS, which provides parsing, static analysis and proof management for heterogeneous logical theories. HETS visualises the module structure of complex logical theories, using so-called development graphs. For individual nodes (corresponding to logical theories) in such a graph, the concept hierarchy can be displayed. Moreover, HETS is able to prove intended consequences of theories, prove refinements between theories, or

demonstrate their consistency. This is done by integrating several first-order provers and model-finders (SPASS, DARWIN), the higher-order prover (ISABELLE), as well as the DL reasoner PELLET.

Our contributions in this paper are: (i) we suggest a heterogeneous framework for the design of ontologies, based on the theory of institutions and the notion of development graph (Sec. 2); (ii) we supply an implementation of this framework (including reasoning support) based on the tool HETS and present a concrete syntax for $\mathcal{SROIQ}(D)$ that fits seamlessly into our heterogeneous approach (Sec. 3); (iii) we present a simple example showing how the structuring techniques for heterogeneous ontologies can be used in practice (Sec. 4), show how heterogeneous refinements are covered by our approach, indicate how they can be proved (automatically), and give a definition of heterogeneous sub-ontology (Sec. 5). Finally, Sec. 6 discusses future work.

2 Structuring, Modularity, and Heterogeneity for Ontologies

2.1 CASL and Institution Theory

The Common Algebraic Specification Language CASL [4, 6] provides a user-friendly notation for first-order logic, much in the same way that HETDL (see below) and Manchester syntax provide a user-friendly notation for OWL-DL. A sample CASL specification is shown in Fig. 1.

A major strength of CASL is the provision of language constructs for writing modular (and heterogeneous) theories, and for the specification of refinements between theories.

```
spec Prey_Animals =
     sort Thing
     pred Hare      : Thing
     pred Mouse     : Thing
     pred isTastier : Thing * Thing
     forall a,b :Thing
     . isTastier (a,b) =>
       not isTastier (b,a)
     . Hare(a)  /\ Mouse (b) =>
       isTastier (a,b)
end
```

Fig. 1: Example in CASL

The study of modularity principles can be carried out to a quite large extent independently of the details of the underlying logical system that is used. The notion of **institutions** was introduced by Goguen and Burstall in the late 1970s exactly for this purpose (see [13]).

Indeed, CASL's structuring concepts can be used for an arbitrary institution. In order to avoid the use of category theory and to make the paper as accessible as possible, we here present the notion of an institution in an informal way.

Definition 1. *An* institution *is a mathematical structure providing the following data:*

- *A set of* ***signatures****, where a signature Σ is a vocabulary of symbols. In* OWL*, a signature comprises a set of concept names and a set of role names. In principle, any set of objects can be used as a signature (similar remarks apply to the notions of sentences, models, and signature morphisms below).*
- *For each signature Σ, a set of Σ-****sentences****. In* OWL*, sentences are the usual description logic formulas (using only concept and role names of Σ).*
- *For each signature, a class of Σ-****models****. Σ-models typically provide semantic interpretations of all the symbols in Σ.*
- *For each signature Σ, a* ***satisfaction relation*** *$\models_\Sigma$ between Σ-models and Σ-sentences. This typically is the standard Tarskian notion of truth, but non-standard notions of satisfaction (truth) can be used as well.*

- *For each pair of signatures, a set of* ***signature morphisms***[4]*, i.e. mappings between the signatures. In* OWL*, signature morphisms consist of two mappings, one for concepts and one for roles.*
- *For each signature morphism* $\sigma\colon \Sigma_1 \longrightarrow \Sigma_2$*,* Σ_1*-sentences can be* ***translated*** *along the signature morphism: given a* Σ_1*-sentence* φ_1*, its translation is written* $\sigma(\varphi_1)$*.*
- *For each signature morphism* $\sigma\colon \Sigma_1 \longrightarrow \Sigma_2$*,* Σ_2*-models can be* ***reduced*** *against the direction of the signature morphism: given a* Σ_2*-model* M_2*, its reduct is written* $M_2{\restriction_\sigma}$*.*

The only condition governing institutions (i.e. the relation between the above items) is the so-called ***satisfaction condition****, stating that truth is invariant under change of notation:*

$$M_2{\restriction_\sigma} \models_{\Sigma_1} \varphi_1 \textit{ iff } M_2 \models_{\Sigma_2} \sigma(\varphi_1)$$

Nearly all logics occurring in practice[5] can be formalised as institutions. The usual notions of logical consequence and satisfiability can be defined in an arbitrary institution. For example, given a set of Σ-sentences Γ and a Σ-sentence φ, we say that φ is a logical consequence of Γ, written $\Gamma \models \varphi$, if all Σ-models satisfying Γ also satisfy φ. Given a signature morphism $\sigma\colon \Sigma_1 \longrightarrow \Sigma_2$ and a Σ_1-model M_1, a σ**-expansion** is any Σ_2-model M_2 with $M_2{\restriction_\sigma} = M_1$.

2.2 Development Graphs for Structuring and Modularity

The advantage of the notion of institution is that it offers the possibility of defining and studying structuring constructs (and their semantics) in a way that abstracts from the details of the particular logical system. In particular, this means that we can use the same structuring constructs for *both*, description logics (e.g. HETDL introduced below) *and* first-order logic (e.g. CASL) (as well as many others, e.g. modal logic).

Tools like the heterogeneous tool set HETS do not directly work on CASL's structuring constructs, but on a graphical translation of these, the so-called *development graphs*. Practically all languages for structuring and modularity can be mapped into this formalism of development graphs.

We use this notion of a *development graph* as a general semantic-based representation formalism for structured ontologies. The basic structuring operation for ontologies is surely that of *importing* other ontologies, and development graphs capture this as theory extensions. However, they also cover renaming of symbols and conservative/definitional extensions.

Definition 2. *Fix an institution (which will give the semantic background for making notions such as signature, sentence, model, signature morphism and reduct precise). A* ***development graph*** *is an acyclic, directed graph, subject to the following conditions. Each node is decorated with a signature and a set of sentences over that signature, which together constitute the* ***local theory*** *of that node. This corresponds to an unstructured logical module, for example, a single* OWL-DL *ontology.*

The links in the graph can be of different types. ***Global***[6] ***definition links*** $K \xrightarrow{\sigma} N$ *represent imports of other theories; they are decorated with a signature morphism* σ *be-*

[4] Signature morphisms are required to form a *category*, that is, they can be composed and there are identity signature morphisms.

[5] Non-monotonic logics can be represented by a trick that models entailments between ordinary sentences as institutions sentences.

[6] There are also local and hiding definition links, which require a more refined model-theoretic semantics.

tween the signatures of the involved nodes. Note that the signature morphism offers the possibility of renaming symbols while importing them.

Given a node N in a development graph $\mathcal{DG}$, its associated theory $\mathbf{Th}_{\mathcal{DG}}(N)$ is inductively defined to consist of

- *all the local axioms of N, and*
- *for each global definition link $K \xrightarrow{\sigma} N \in \mathcal{DG}$, all of $\mathbf{Th}_{\mathcal{DG}}(K)$ translated by σ.*

The class of models $\mathbf{Mod}_{\mathcal{DG}}(N)$ of a node N is defined to consist of all models over N's signature that satisfy the theory $\mathbf{Th}_{\mathcal{DG}}(N)$. □

Complementary to definition links, which *define* the theories of related nodes, we also allow for **theorem links** with the help of which we are able to *postulate* relations between different theories, and hence can be seen as *proof obligations*, refinements, or also alignments. A (global) theorem link is an edge $K \overset{\sigma}{\dashrightarrow} N$, where σ runs between the signatures of N and K. A development graph $\mathcal{DG}$ **implies** a theorem link $K \overset{\sigma}{\dashrightarrow} N$ (denoted $\mathcal{DG} \models K \overset{\sigma}{\dashrightarrow} N$) if and only if all reducts of N-models are K-models, formally, for all $M \in \mathbf{Mod}_{\mathcal{DG}}(N)$, $M{\restriction_\sigma} \in \mathbf{Mod}_{\mathcal{DG}}(K)$.

A global definition (or also theorem) link $K \xrightarrow{\sigma} N$ can be strengthened to a **conservative extension link** (denoted as $K \xrightarrow[cons]{\sigma} N$); it holds if every K-model has a σ-expansion to an N-model. Such annotations can be seen as another kind of proof obligations. **Definitional extensions** are introduced in a similar way (annotated with def); here the σ-expansion has to be unique. This means that a definitional extension is one where all the additional symbols in N (i.e. those not imported from K) are axiomatised (in N) in a unique way (allowing only a *unique* interpretation given a K-model). By contrast, a conservative extension only requires that these symbols can be interpreted in *some* way.

Many languages for structuring, modularity and alignment of ontologies can be mapped into this formalism of development graphs. Issues of modularity have been recognised as being rather important for a while now, and have resulted in extensive research concerning modularity principles (compare the proceedings of the workshops [15; 7; 31], and recent work on (deciding) conservative extensions [28; 8]). We here use the term *modularity* to refer to the notion of 'ontological module' defined through conservativity properties, as it has been investigated for instance in [20, 8, 25], and the term *structuring* for the systematic combination of (possibly heterogeneous) ontologies through (not necessarily conservative) operations such as union, extension, etc. The verification of conservativities, or the check of syntactic 'safety' conditions for conservativity, is accomplished from within the tool HETS, employing corresponding algorithmic approaches.

2.3 Heterogeneous Ontologies and HETCASL

Since ontologies are being written in many different formalisms, like description logics, first-order logic, and modal (first-order) logics, combinations of ontologies need to be constructed across different institutions, as is argued convincingly in [32].

To obtain heterogeneous logical theories, one first needs to fix some graph of logics and logic translations, usually formalised as institutions and so-called institution comorphisms, mapping signatures, sentences and models in a way that satisfaction is preserved, see the discussion above and [14] for further details.

The so-called **Grothendieck institution** allows to give a semantics to heterogeneous theories involving several institutions (see [9, 29]). Basically, it is a flattening, or disjoint union, of the logic graph. A signature in the Grothendieck institution is a pair consisting of a logic (institution) and a signature in that logic. Similarly, a Grothendieck signature morphism consists of a logic translation plus a signature morphism (in the target logic). Sentences, models and satisfaction in the Grothendieck institution are defined componentwise. We now arrive at the following:

Definition 3. *An **abstract structured heterogeneous ontology** (w.r.t. some logic graph) is a node O in a development graph $\mathcal{DG}$ in the corresponding Grothendieck institution.*

(We sometimes also identify O with its theory $\mathbf{Th}_{\mathcal{DG}}(O)$; however, note that then the structuring is lost.) To be able to write down such heterogeneous ontologies in a concise manner, we extend CASL to HETCASL as follows: HETCASL provides the notation `logic <logic-name>`, which defines the institution of the following specifications until that keyword occurs again. Also, a specification can be translated along a comorphism; this is written `<spec> with logic <comorphism-name>`.

A HETCASL library consists of specification definitions as shown in Fig. 2. We now briefly describe the form of HETCASL specifications, together with their translation into development graphs. A specification `<spec>` can be a basic specification consisting of a signature and some axioms (with syntax specific to the given institution). It corresponds to a node with local axioms in a development graph. A specification can be extended with further signature elements and axioms, written `<spec> then <spec>` (see Fig. 2). This leads to a definition link (decorated with an inclusion signature morphism) in the development graph, via which the node for the second specification imports the node for the first one. Extensions (and thereby, their definition links) can be declared to be conservative or definitional (with semantics as introduced above). Two specifications can be united, written `<spec> and <spec>`. Their nodes are linked, again using definition links, into a new node representing the union. Semantically, unions unite the requirements of two specifications, thereby intersecting their model classes. Renamings, written `<spec> with <signature-morphism>`, rename a specification along a signature morphism; again, this leads to a definition link in the development graph (this time usually decorated with a non-inclusion signature morphism). The declaration `view view1: sp1 to sp2` will generate a theorem link between the nodes representing `sp1` and `sp2` in the development graph. Details of the translation to development graphs, as well as a treatment of hiding, can be found in [30].

```
logic DL
spec MoreTigers =
    Tigers
then
    Individual: Tethys
        Type: Tiger
        DifferentFrom: Phobos, Deimos
end
```

Fig. 2: An extension

3 HETDL: A spawn of Manchester Syntax for OWL

We have defined a new syntax for the description logic $\mathcal{SROIQ}(D)$ [18], called HETDL (Heterogeneous DL), which is based on the Manchester Syntax for OWL 1.0 [16] and which has been developed in parallel to the Manchester Syntax for OWL 2.0 [17].[7] A grammar for HETDL is supplied via a technical report [23]; tool support is offered via

[7] The main reason for this parallel development was the lack of a complete formal grammar for the Manchester Syntax at the time.

HETS. An important feature of HETDL, and its main difference to standard description logics as well as OWL 2.0, is its support for the structuring features of HETCASL, discussed in the previous section. This makes possible the heterogeneous development of and proof-support for ontologies, involving OWL as well as several other logics of different expressivity and with different semantics, in a single ontology design

A simple example for a specification in HETDL is given in Fig. 3: in it, we define two different tigers, called Phobos and Deimos, which are carnivores and have 4 legs. To simplify the definition of many individuals that share the same facts, HETDL introduces the keyword `Individuals`, which is used to define properties for several individuals in a single block of text. `Individuals` has a field `Equality`, which is used to declare all individuals in the list to be the same or different. To illustrate this, consider as an example the definition in Fig. 4 on the left, which is equivalent to the longer definition in Fig. 4 on the right. The construct `Individuals` was introduced as a short-cut notation for the convenience of the ontology-developer.

```
logic DL
spec Tigers =
    Class: Tiger
        SubclassOf: Carnivore,
                    hasLegs exactly 4

    Individuals: Phobos, Deimos
        Type: Tiger
        Equality: Different
end
```

Fig. 3: A simple HETDL specification

```
Individuals: Phobos, Deimos, Tethys
    Types:       Tiger
    Equality:    Different
```

```
Individual:    Phobos
    Types:         Tiger
    DifferentFrom: Deimos, Tethys

Individual:    Deimos
    Types:         Tiger
    DifferentFrom: Phobos, Tethys

Individual:    Tethys
    Types:         Tiger
    DifferentFrom: Phobos, Deimos
```

Fig. 4: Short and longer definition of distinct individuals of the same type

Of course, abstract structured heterogeneous ontologies can be formulated in different notations, and HETCASL is only one of them. Another option would be an extension of OWL with keywords dealing with corresponding structuring mechanisms.

We have designed an institution comorphism from HETDL to first-order logic (CASL). This comorphism is designed along OWL 2.0's model-theoretic semantics, adapted to the structuring of HETDL specifications. The translation provided is essentially the same as the standard-translation to first-order logic, with the minor difference that we translate into a many-sorted first-order variant—see [23] for full details. Consider a subsumption like:

```
Class: Tiger
    SubclassOf: hasClaws some Claws
```

This will be translated to a sentence and a predicate declaration. The expression `Class: Tiger` yields the declaration **pred** `Tiger : Thing`. Then, `SubclassOf: hasClaws some Claws` is translated to $\forall$ `x : Thing . Tiger(x)` $\Longrightarrow$ $[\![$`hasClaws some Claws`$]\!]$ `(x)`, where $[\![s]\!]$ denotes the mapping of the concept s along the comorphism. Note that the class `Tiger` is used in the formula for the subclass definition. More generally, all statements following one of the keywords `Class`, `ObjectProperty`, `DataProperty`, and `Individual(s)`, are treated in this way, until the next keyword of this type is reached.

4 Plugging things together: A heterogeneously structured ontology

After having discussed the theory of heterogeneous ontologies in some detail in Sec. 2.3, we now illustrate how to define an ontology heterogeneously from 3 parts formalised in different languages. Firstly, consider a basic specification written in HETDL, given in Fig. 5 on the left hand side, describing Tigers being carnivores and cats of prey.

```
logic DL
spec Predators =
     Class: Carnivore

     Class: Tiger
         SubclassOf: Carnivore,
                     CatsOfPrey
end
```

```
logic CASL
spec Prey =
     Prey_Animals
then %implies
     forall a,b : Thing
     . Hare(a)  /\ Mouse (b) => not isTastier (b,a)
end
```

Fig. 5: Predators and their prey

Secondly, consider another basic specification in CASL (please remember that `Prey_Animals` is given in Fig. 1) describing their prey, given in Fig. 5 on the right hand side. The keyword `%implies` here introduces a proof obligation, namely a theorem link expressing that the part after `%implies` logically follows from the parts before. This particular proof obligation follows from the asymmetry of `isTastier` specified in Fig. 1. Such annotations can be quite useful for an ontology designer as a control mechanism to keep track of *desired consequences*: in case such a proof obligation fails, a design error has been made. Thirdly, consider the specification below:

```
logic CASL
spec Animals =
     Predators and {Prey with Hare |-> Lepus}
then
     pred prefers : Thing * Thing * Thing
     forall a,b,c : Thing
     . Tiger(a) /\ isTastier (b,c) <=> prefers (a,b,c)
then %implies
     forall a,b,c : Thing
     . Tiger(a) /\ Lepus(b) /\ Mouse (c) => prefers (a,b,c)
end
```

This is the union of the above ones and adds a new 3-ary predicate that cannot directly be expressed in $\mathcal{SROIQ}(D)$. In the process of uniting the specifications, `Predators` is mapped along the comorphism from HETDL to CASL. Further, `Hare` is being renamed to `Lepus`. The development graph of this heterogeneous specification is displayed in HETS as shown in Fig. 6. Please note that the solid black arrows depict definition links (the double-lined arrows indicate *heterogeneous* definition links), whilst the light green arrows are theorem links (and the dotted green arrows *local* theorem links introduced by the development graph calculus [30]). The unnamed nodes, which contain the proof obligations, can now be proved by running a theorem-prover on them, i.e., on their local theories. This way, we do not have to deal with axioms that are introduced later and that are not important for this theory. Please note that the theory of the unnamed node with a theorem link from `Prey`

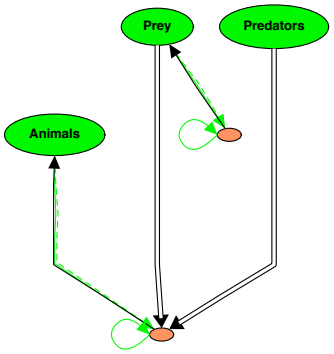

Fig. 6: Development Graph of the Heterogeneous specification

is formalised in a DL, thus allowing it to be proved by a DL reasoner. With this approach, many 'conjectures' can already be proven in a smaller, 'local' environment. Further, this approach helps the designer of an ontology to find inconsistencies: if the overall ontology turns out to be inconsistent, it is possible to check the consistency of the theories of all nodes in the development graph that contribute to the overall specification. If one of them turns out to be inconsistent, it might already be possible to fix the inconsistency in this smaller, local theory. Note that this 'scales down' the search space for finding inconsistencies in a way that is independent from the techniques developed in [19].

5 Heterogeneous Refinements and Sub-Ontologies

When comparing different ontologies it is of interest whether all axioms of an ontology O_1 are also entailed by another (larger or more complex) ontology O_2. This is formalised via the notion of refinement adapted from software engineering [11]. Note that since we do not assume that O_2 is a superset of O_1 (by 'larger' we simply mean 'greater number of axioms'), deciding refinements is in general non-trivial.[8] A new notion in the area of ontology design is that of a *heterogeneous refinement* that covers the important case where different logics are involved; it is formalised as follows:

Definition 4. *Given two ontologies O_1 and O_2 in the same logic, we call O_2 a **refinement** of O_1 if there is a theorem link $O_1 \xrightarrow{\sigma} O_2$ that follows from the underlying development graph. Now let ontologies O_1 and O_2 in logics $\mathcal{L}_{O_1}$ and $\mathcal{L}_{O_2}$ be given, such that there is a logic $\mathcal{L}$ with comorphisms $\mathcal{L}_{O_1} \xrightarrow{\theta} \mathcal{L}$ and $\mathcal{L}_{O_2} \xrightarrow{\eta} \mathcal{L}$, where η is conservative. The translations of the ontologies along the comorphism are referred to as ${O_1}'$ and ${O_2}'$. We call O_2 a **heterogeneous refinement** of O_1 if there is a theorem link $O_1' \xrightarrow{\sigma} O_2'$ that follows from the underlying development graph.*

Proposition 5. *For a heterogeneous refinement, any O_2-model can be translated to an O_1-model, and moreover, logical consequence is preserved along refinement: for $\sigma(\theta(\varphi)) = \eta(\psi)$, $O_1 \models \varphi$ implies $O_2 \models \eta(\psi)$.*

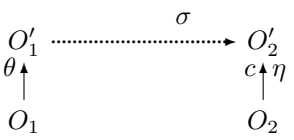

Fig. 7: A refinement Diagram.

Fig. 7 depicts a heterogeneous refinement as defined in Def. 4. In HETCASL, this concept is addressed by the notion of `view` creating a proof obligation, as discussed in Sec. 2.

Our abstract definition of refinement entails the common definition in software engineering: consider O_1 to be a specification of a program in an algebraic specification language, and O_2 its implementation in a programming language. Refinements are a common problem in the world of ontologies as well: establish whether a domain ontology is consistent with respect to the knowledge represented in a foundational ontology. The notion of refinement leads us to the definition of heterogeneous sub-ontologies.

Definition 6. *We call an ontology O_1 a* (heterogeneous) sub-ontology *of O_2 if and only if O_2 is a (heterogeneous) refinement of O_1.*

[8] There are other usages of the term 'refinement' in the DL literature, e.g. in concept learning [26].

5.1 A Very Simple Heterogeneous Refinement: Cats

We will clarify the notion of heterogeneous refinement with an example. Consider a small ontology dealing with cats of prey in HETDL:

```
logic DL
spec Cats =
     ObjectProperty: isFaster
         Characteristics: Transitive, Irreflexive

     Class: Cheetah
         SubclassOf: Carnivore, isFaster some Tiger
         DisjointWith: Tiger

     Class: Tiger
         SubclassOf: Carnivore
         DisjointWith: Cheetah
end
```

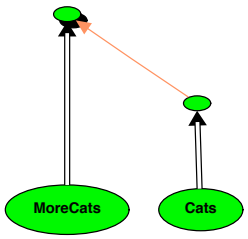

Fig. 8: Heterogeneous Refinement

and another ontology of these animals given in CASL and containing more information:

```
logic CASL
spec MoreCats =
     pred Cheetah, Tiger, Lion : Thing
     pred isFaster  : Thing * Thing
     pred Carnivore : Thing
     forall a,b,c :Thing
     . Cheetah (a) => Carnivore (a)
     . Tiger   (a) => Carnivore (a)
     . Lion    (a) => Carnivore (a)
     . Cheetah(a)  => exists b : Thing . isFaster(a,b) /\ Tiger(b)
     . Tiger(a)    => exists b : Thing . isFaster(a,b) /\ Lion(b)
     . not (Tiger(a) /\ Cheetah(a))
     . not (Tiger(a) /\ Lion(a))
     . not (Lion(a) /\ Cheetah(a))
     . not isFaster(a, a)
     . isFaster(a, b) /\ isFaster(b, c) => isFaster(a, c)
end
```

One can easily see that there is in fact a signature inclusion morphism from `Cats` to `MoreCats`. To create a proof obligation that covers the fact that `MoreCats` might be a refinement of `Cats`, a heterogeneous view in CASL is introduced:

```
logic CASL
view cview : Cats to MoreCats
```

Again, we can easily see that all models of `MoreCats` are models of `Cats` if we reduce the signature to forget the `Lion`. By Def. 4, `MoreCats` is a heterogeneous refinement of `Cats`, while `Cats` is a sub-ontology of `MoreCats`. In HETS, this refinement is displayed as in Fig. 8 on the right.

6 Discussion and Future Work

We have introduced an abstract framework for the study of structured heterogeneous ontologies, allowing for a systematic analysis of conceptual and algorithmic problems in heterogeneous environments that were previously considered rather disparate. We have pointed out a way for ontology designers to build their ontologies in a heterogeneous and structured fashion, splitting it up in several meaningful modules and plugging them together to make up the overall ontology. With this heterogeneous approach it is possible to define parts of an ontology in several logics depending on the needed expressivity. The mantra of this approach is: as simple as possible, as expressive as needed.

We have given a notion of heterogeneous refinement providing a very strong relation between two ontologies, and shown that it is directly supported within our framework. With this notion, we can determine if an ontology is a sub-ontology of another, larger or more complex one and which might be specified in a different formalism. This provides a convenient tool for the comparison of ontologies.

Unlike related approaches like Common Logic [27], our approach provides explicit structuring mechanisms, and logic translations are treated as first-class citizens. Of course, it is also possible without too much effort to add Common Logic as another logic to the HETS logic graph.

The structured reasoning support that our approach allows has already been used for answering questions that 'standard' automated reasoning can not tackle: the consistency of the first-order version of the foundational ontology DOLCE (reformulated as a HET-CASL specification) can be verified by model-checking a view into a finite specification of a model for DOLCE, and the structuring techniques built into HETS also support the modular construction of models for large first-order ontologies such as DOLCE [22]. We also work on determining the exact logical relationship between different versions of DOLCE, formalised in various DLs as well as first-order logic: here, only partial heterogeneous refinements can be established as the DL versions are hand-made approximations of the first-order version.

Currently, we are working on integrating a tool for the discovery of theory morphisms into the Heterogeneous Tool Set as well as on integrating modularisation algorithms as developed in [21; 8]. These techniques would allow (semi)-automatic structuring of ontologies and the discovery of ontology overlaps modulo alignment mappings.

Acknowledgements

Work on this paper has been supported by the DFG-funded collaborative research centre SFB/TR 8 Spatial Cognition and by the German Federal Ministry of Education and Research (Project 01 IW 07002 FormalSafe). We thank John Bateman, Mihai Codescu, Alexander Garcia Castro, Joana Hois, and Lutz Schröder for fruitful discussions, the anonymous reviewers for constructive comments, and Erwin R. Catesbeiana for insights into conservativity issues.

Bibliography

[1] A. Artale and E. Franconi. A survey of temporal extensions of description logics. *Annals of Mathematics and Artificial Intelligence*, 30(1-4):171–210, 2000.

[2] A. Artale, R. Kontchakov, C. Lutz, F. Wolter, and M. Zakharyaschev. Temporalising tractable description logics. In *Proc. of the 14th Int. Symposium on Temporal Representation and Reasoning (TIME)*, pages 11–22, Washington, DC, USA, 2007. IEEE.

[3] J. Bateman, T. Tenbrink, and S. Farrar. The Role of Conceptual and Linguistic Ontologies in Discourse. *Discourse Processes*, 44(3):175–213, 2007.

[4] M. Bidoit and P. D. Mosses. CASL *User Manual*. LNCS Vol. 2900. Springer, 2004.

[5] D. Calvanese, D. Lembo, M. Lenzerini, and R. Rosati. Epistemic first-order queries over description logic knowledge bases. In *In Proc. DL 2006, Lake District, UK, May 30June*, page 2006.

[6] CoFI (The Common Framework Initiative). CASL *Reference Manual*. LNCS Vol. 2960. Springer, 2004.

[7] B. Cuenca Grau, V. Honavar, A. Schlicht, and F. Wolter, editors. *2nd International Workshop on Modular Ontologies (WoMO-07)*, volume 315, (K-CAP) Whistler, BC, Canada, 2007. CEUR Workshop Proceedings.

[8] B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Modular Reuse of Ontologies: Theory and Practice. *J. of Artificial Intelligence Research (JAIR)*, 31, 2008.
[9] R. Diaconescu. Grothendieck Institutions. *Applied Categorical Structures*, 10:383–402, 2002.
[10] F. M. Donini, M. Lenzerini, D. Nardi, W. Nutt, and A. Schaerf. An epistemic operator for description logics. *Artif. Intell.*, 100(1-2):225–274, 1998.
[11] H. Ehrig and H.-J. Kreowski. Refinement and implementation. In E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, editors, *Algebraic Foundations of Systems Specifications*, pages 201–242. Springer Verlag, 1999.
[12] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L . Schneider. Sweetening Ontologies with DOLCE. In *Proc. of EKAW 2002*, LNCS Vol. 2473, pages 166–181. Springer, 2002.
[13] J. A. Goguen and R. M. Burstall. Institutions: Abstract Model Theory for Specification and Programming. *Journal of the ACM*, 39:95–146, 1992.
[14] J. A. Goguen and G. Roşu. Institution morphisms. *Formal Aspects of Computing*, 13:274–307, 2002.
[15] P. Haase, V. Honavar, O. Kutz, Y. Sure, and A. Tamilin, editors. *1st Int. Workshop on Modular Ontologies (WoMO-06)*, volume 232, (ISWC) Athens, Georgia, USA, 2006. CEUR Vol. 232.
[16] M. Horridge, N. Drummond, J. Goodwin, A. Rector, R. Stevens, and H. Wang. The Manchester OWL Syntax. In *OWL: Experiences and Directions*, 2006.
[17] M. Horridge and P. Patel-Schneider. Manchester Syntax for OWL 1.1. In *OWL: Experiences and Directions*, Washington, DC, 2008.
[18] I. Horrocks, O. Kutz, and U. Sattler. The Even More Irresistible $\mathcal{SROIQ}$. In *Proc. of KR*, pages 57–67, 2006.
[19] A. Kalyanpur, B. Parsia, M. Horridge, and E. Sirin. Finding all Justifications of OWL DL Entailments. In *Proc. of ISWC/ASWC2007*, LNCS Vol. 4825, pages 267–280. Springer, 2007.
[20] B. Konev, C. Lutz, D. Walther, and F. Wolter. Formal properties of modularization. In H. Stuckenschmidt and S. Spaccapietra, editors, *Ontology Modularization*. Springer, 2008.
[21] B. Konev, C. Lutz, D. Walther, and F. Wolter. Semantic Modularity and Module Extraction in Description Logics. In *18th European Conf. on Artificial Intelligence (ECAI-08)*, 2008.
[22] O. Kutz, D. Lücke, and T. Mossakowski. Modular Construction of Models—Towards a Consistency Proof for the Foundational Ontology DOLCE. In *1st Int. Workshop on Computer Science as Logic-Related*, ICTAC 2008, Istanbul, Turkey, 2008.
[23] O. Kutz, D. Lücke, T. Mossakowski, and I. Normann. The OWL in the CASL. Technical report, University of Bremen, Bremen, Germany `http://www.informatik.uni-bremen.de/~okutz/OWLinCASL-TR.pdf`, 2008.
[24] O. Kutz, C. Lutz, F. Wolter, and M. Zakharyaschev. E-connections of abstract description systems. *Artificial Intelligence*, 156(1):1–73, 2004.
[25] O. Kutz and T. Mossakowski. Conservativity in Structured Ontologies. In *18th European Conf. on Artificial Intelligence (ECAI-08)*. IOS Press, 2008.
[26] J. Lehmann and P. Hitzler. Foundations of refinement operators for description logics. In *Proc. of the 17th Int. Conf. on Inductive Logic Programming (ILP)*, volume 4894 of *LNCS*, pages 161–174. Springer, 2008.
[27] Common Logic. `http://common-logic.org/`.
[28] C. Lutz, D. Walther, and F. Wolter. Conservative Extensions in Expressive Description Logics. In *Proceedings of IJCAI-07*, pages 453–458. AAAI Press, 2007.
[29] T. Mossakowski. Comorphism-based Grothendieck logics. In *Mathematical Foundations of Computer Science*, volume 2420 of *LNCS*, pages 593–604. Springer, 2002.
[30] T. Mossakowski, S. Autexier, and D. Hutter. Development Graphs—Proof Management for Structured Specifications. *J. of Logic and Algebraic Programming*, 67(1–2):114–145, 2006.
[31] U. Sattler and A. Tamilin, editors. *Workshop on Ontologies: Reasoning and Modularity (WORM-08)*, volume 348, ESWC, Tenerife, Spain, 2008. CEUR Workshop Proceedings.
[32] M. Schorlemmer and Y. Kalfoglou. Institutionalising Ontology-Based Semantic Integration. *Journal of Applied Ontology.*, 2008. To appear.
[33] N. Sioutos, S. de Coronado, M. W. Haber, F. W. Hartel, W.-L. Shaiu, and L. W. Wright. NCI Thesaurus: A semantic model integrating cancer-related clinical and molecular information. *Journal of Biomedical Informatics*, 40(1):30–43, 2007.