Specification of ontologies in CASL *

Klaus Lüttich, Till Mossakowski
{luettich,till}@tzi.de
University of Bremen, Germany

Abstract. This paper proposes to use CASL (Common Algebraic Specification Language; designed by CoFI – Common Framework Initiative) for formalising ontologies in FOL. The major advantage of CASL over other specification techniques is its static strong typing and subtyping. Namely, using sorts (types) instead of unary predicates for the ontological categories gives the possibility to find unintended application of n-ary predicates during the type checking of CASL. Another advantage of CASL is its structuring facilities that provide renaming and hiding of symbols used in specifications and the instantiation of parametrised specifications. All this is supported by an evolving tool set for syntax and type analysis, called HETS (Heterogeneous Tool Set), that provides the connection to various provers (interactive and automatic). The tool set allows the combination and detection of various different logics and sublogics. There is a further specialisation of CASL related to the Semantic Web ontology language OWL-DL called CASL-DL (CASL-Description Logic). It gives, as a sublanguage of CASL, the ability to use automatic reasoners like Racer for inferences while using HETS for type checking of the predicates. A consistent view of the same ontology in CASL (for expressiveness) and CASL-DL (for easier automated reasoning with tools like Fact and Racer) is provided by using HETS. Moreover, the first-order axioms can be kept when using a first-order prover. It is well known that there are first-order provers that are as fast as Fact or Racer on description logic fragments, thus combining the best of both worlds. We use HETS as a tool to exploit this possibility.

Introduction

The aim of our proposal is to use CASL as a development language for ontologies. Starting from a very basic and loose foundational ontology up to highly optimised applied ontologies, everything can be developed within one consistent framework. All this work is supported by a whole set of tools integrated via HETS (Heterogeneous Tool Set). The tool set keeps track of all the proofs and also provides some versioning of specifications. This approach brings together the advantages of having both expressive ontologies and tool support. Via the detection of certain sublogics even very effective reasoners could be used with FOL ontologies.

This paper is structured as follows: first we describe the language CASL and its background. The next section shows some advantages of the typed and readable language CASL over some other specification formalisms. A further section gives a fairly long example to illustrate the usage of CASL as an ontology specification language. The following two sections outline the language CASL-DL, a CASL restricted to Description Logic, and its correspondence to OWL-DL. Finally, the Heterogeneous Tool Set is discussed and an application of the theorem prover Isabelle [1] to the example is shown.

*This paper has been supported by Deutsche Forschungsgemeinschaft in the SFB/TR8 ”Spatial Cognition”.

1 CASL

CASL, the *Common Algebraic Specification Language* [2], has been designed by COFI, the *Common Framework Initiative* for algebraic specification and development. It has been designed by a large number of experts from different groups, and serves as a de-facto standard. The design of CASL has been approved by the IFIP WG 1.3 "Foundations of Systems Specifications".

CASL consists of several major *levels*, which are quite independent and may be understood (and used) separately:

Basic specifications are written in many-sorted, first-order logic. Subsorts (interpreted as injective embeddings) increase flexibility, while retaining a strong type system. Partial functions with possibly undefined values are allowed as well. Finally, the CASL basic specifications provide powerful and concise constructs for specifying datatypes, which in the presence of subsorts also can be used to specify disjoint, non-disjoint and exhaustive unions of sorts.

Structured specifications allow translation, reduction, union, and extension of specifications. A simple form of generic (parametrised) specifications is provided, allowing specifications to be re-used in different contexts.

Architectural specifications define how models of a specification may be constructed out of models of simpler specifications.

Libraries allow the distributed storage and retrieval of (particular versions of) named specifications.

Major libraries of validated CASL specifications are freely available on the Internet, and the specifications can be reused simply by referring to their names. Tools are provided to support practical use of CASL: checking the correctness of specifications and proving facts about them.

While CASL was originally designed for specifying requirements and design for software, in this work we show that CASL can also perfectly suit as a language for formalizing ontologies.

2 Advantages of CASL in contrast to other formalisms

This section describes CASL in contrast to untyped FOL written in LaTeX and KIF. The main advantages of CASL are (1) strong typing with subtyping (2) structuring of specifications for reuse and parametrisation and (3) tool support with pretty printing in LaTeX and (4) the connection to various reasoners.

2.1 Correspondence of Unary Predicates and Sorts

A common way of representing and specifying ontologies in first-order logic is to use unary predicates for concepts or categories. This is done, for example, in the DOLCE ontology [3]. In CASL the hierarchy of categories can be achieved by exploiting the subtyping. The devision of categories into (disjoint) subcategories is shown in Fig. 1. We propose to use this feature of CASL to distinguish categories from unary predicates representing a certain feature as shown in Fig. 3. There the **sort** *s* represents a category and the **pred** *At* : *s* represents a feature present for some individuals of *s*.

sorts $C < A; D, E < B$	**%%** subsumption declarations
free type $TOP ::= sort\ A, B$	**%%** disjoint partition of *TOP*
pred $rel : E \times B$	**%%** predicate declaration
op $g : E$	**%%** declaration of an individual
$\forall x: E; z: TOP$	
$\bullet\ rel(x, g)$	**%%** correct application of *rel*
$\bullet\ rel(z, g)$	**%%** application with type error

Figure 1: Examples of Declarations and Definitions

The main advantage of this technique is an early detection of false or unintended predicate application. For example, the second axiom in Fig. 1 has a typing error because the z is of a more general type than the profile of *rel* insists on. If *rel* should be applicable to the sort *TOP* in general, then the profile of *rel* has to be adjusted as well. The type correct application in the example without changing the profile of *rel* would be an explicit cast to the subsort with *z as E*. With this, a projecting function from *TOP* into *E* is called.

2.2 Structuring of Specifications

CASL is equipped with various structuring facilities for specifications: (1) naming, (2) extension and union of specifications, (3) renaming and hiding of symbols and (4) parametrisation of specifications. The last feature and a larger example illustrating the structuring mechanisms are discussed in more detail in Sect. 3.

Naming of specifications can be used to group parts of the ontology into sub-theories or sub-domains. These parts could also include formulas marked as theorems which could be proved only within a certain part of an ontology. Whenever a named specification is used in another context these theorems are treated as proven and can be used in further proofs like axioms.

Extensions and unions of specifications are used to combine basic specifications and instantiated specifications with the CASL keywords **then** and **and**. Basic specifications contain the declarations and definitions of symbols as well as the axioms regarding these symbols. By instantiation of named specifications either parametrised or unparametrised specifications can be reused in different contexts.

Renaming and the hiding of symbols is used to keep symbols separate that otherwise would be identified, because one major principle in CASL is “same name, same thing”. Renaming can also be used to do the converse within the union of separate specifications. Furthermore, unwanted or unneeded symbols like auxiliary predicates can be hidden.

2.3 Readable Syntax and automatic translation to LaTeX

The Lisp notation of KIF [4] is not very readable. CASL provides an input syntax with symbols closely related to mathematical notions, e.g. $\vee$ is input as `\/`. All logical symbols are naturally used in infix or prefix position and have their usual precedences. And HETS, the Heterogeneous Tool Set, provides an automatic translation to LaTeX. Identifiers can be built of various symbols and the arguments of predicates (and functions) can be placed at nearly every

place among or before and after the symbols. These are called generalised mixfix identifiers. For every introduced identifier the user can give a different way how it should be displayed in LaTeX. This leads to very nice looking documents, as can be seen in the examples.

3 Example: First Order Fragment of DOLCE in CASL

This whole section is based on a fragment of DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) [3] used as an example to show how useful and practical it is to write ontologies in CASL. Actually, DOLCE is now being formalized in CASL at the Laboratory of Applied Ontology in Trento, with help of our group in Bremen.

Firstly, we give the declaration of the basic categories. Secondly, we give a definition of a generic parthood which forms a partial order on **sort** *s*. At the right margin labels for the axioms are given for referencing them in proofs.

spec PRIMITIVES =
  %% Basic Categories
  sorts *PD*, *PED*, *S*, *SL*, *T*, *TL* $<$ *PT*
  free type *PT* ::= *sort PD*, *PED*, *S*, *SL*, *T*, *TL*
end

PT	Particular
PD	Perdurant, Occurance
PED	Physical Endurant
S	Space Region
SL	Spatial Location
T	Time Interval
TL	Temporal Location

spec GENPARTHOOD [**sort** *s*] =
  pred $P : s \times s$
  $\forall\, x, y, z$: s
  • $P(x, x)$ **%(**Ad11**)%**
  • $P(x, y) \wedge P(y, x) \Rightarrow x = y$ **%(**Ad12**)%**
  • $P(x, y) \wedge P(y, z) \Rightarrow P(x, z)$ **%(**Ad13**)%**
end

Figure 2: Primitive Categories and Generic Parthood Specification

The specification GENPARTHOOD is now extended to a generic mereology[1] with definitions of proper part (*PP*), overlapping (*O*) and atomic (*At*). These are all predicates typed with the **sort** *s* given as parameter. Lines starting with %% are comments and those starting with % are annotations. The annotation **%implies** given after a **then** lets HETS generate a proof obligation that these theorems are following from the so far given axioms (see section 6). In instantiations of GENMEREOLOGY these theorems are just treated as usual axioms.

After proving the theorems once, they can be used to give a mereology on time interval (*T*), space region (*S*) and perdurant (*PD*) just by instantiation of GENMEREOLOGY with the appropriate argument specifications.

4 CASL-DL – a sublanguage of CASL

CASL-DL is a sublanguage of CASL restricted to the power of $\mathcal{SHOIN}(\mathbf{D})$ [5], the Description Logic underlying OWL-DL. All constructs known from $\mathcal{SHOIN}(\mathbf{D})$ are made available either as FOL formulas and algebraic declarations or via instantiation of predefined specifications. CASL-DL uses sorts as concepts and subsorting for the taxonomy of concepts. For concepts which depend on some role like *Young* either sorts or typed unary predicates can be

[1] In order to make the examples easier to understand, the definitions of binary sum, binary product and binary difference are omitted; for the same reason also various axioms are left out.

spec GENMEREOLOGY [**sort** s] =
GENPARTHOOD [**sort** s]
then
preds $PP(x, y: s) \Leftrightarrow P(x, y) \wedge \neg P(y, x)$; **%(**Dd1_Proper_Part**)%**
$O(x, y: s) \Leftrightarrow \exists z: s \bullet P(z, x) \wedge P(z, y)$; **%(**Dd2_Overlap**)%**
$At(x: s) \Leftrightarrow \neg \exists y: s \bullet PP(y, x)$; **%(**Dd3_Atom**)%**
then
%% Ground Axioms (2)
$\forall x, y: s$
$\bullet \neg P(x, y) \Rightarrow (\exists z: s \bullet P(z, x) \wedge \neg O(z, y))$ **%(**Ad14**)%**
$\bullet \exists z: s \bullet At(z) \wedge P(z, x)$ **%(**Ad18**)%**
then %implies
$\forall x, y, su, su', p, p', d, d': s$
$\bullet (\forall z': s \bullet At(z') \Rightarrow P(z', x) \Rightarrow P(z', y)) \Rightarrow P(x, y)$ **%(**Td1**)%**
$\bullet (\forall z: s \bullet O(z, x) \Leftrightarrow O(z, y)) \Rightarrow x = y$ **%(**Td3**)%**
end

Figure 3: Part of a Generic Mereology Specification

spec MEREOLOGY =
PRIMITIVES
then
%% Ad7, Ad8, Ad9 and Ad10 are generated by instantiation of GenMereology
GENMEREOLOGY [**sort** T]
then
GENMEREOLOGY [**sort** S]
then
GENMEREOLOGY [**sort** PD]
end

Figure 4: Towards a Mereology on Time Intervals, Space Regions and Perdurants

used. Unary predicates are needed for concepts that potentially have no individuals, because the carrier sets of sorts are always assumed as non-empty.

Furthermore, CASL-DL is limited to use only binary predicates, corresponding to roles known from Description Logics [6]. The roles are declared with types in terms of sorts, such that every predicate has a domain and range. Unintended usage of roles is also found during the type checking. Roles can be declared with multiple profiles. This overloading of predicate symbols is interpreted as one role restricted to the union of each arguments sorts. The role hierarchy is formed with implications. (See Fig. 5 and [7])

Complete concept definitions are given as subsort definitions in CASL. So only individuals fulfilling the formula defining the subsort belong to the newly defined concept. Concepts for which all individuals can be enumerated are defined as free types with the individuals given as the only possible constructors. Concepts that have only partial definitions are given as loose subsorts with axioms describing the knowledge about these concepts. Individuals are defined in terms of constant operations declared to be of a certain type. Facts on individuals are just axioms involving only constant operations and no variables.

Unqualified number or cardinality restrictions on roles in definitions of concepts are formed with special predicates derived from the instantiation of a parametrised specification (see Fig. 5). The usage of a fully typed predicate *hasCar* without *Thing* as a filler shows

spec SAMPLE_PERS =
sorts *Person* < *Thing*;
Woman, *Man*
free type *Person* ::= *sorts Woman, Man*
preds *parentOf* : *Person* × *Person*;
childOf : *Person* × *Person*;
motherOf : *Woman* × *Person*
∀ *c*, *p*: *Person*; *m*: *Woman*
• *parentOf*(*p*, *c*) ⇔ *childOf*(*c*, *p*) **%(**inverse roles**)%**
• *motherOf*(*m*, *c*) ⇒ *parentOf*(*m*, *c*) **%(**role subsumption**)%**
sorts *Parent* = {*p*: *Person* • ∃ *c*: *Person* • *parentOf*(*p*, *c*)}; **%(**parent definition**)%**
Mother = {*m*: *Woman* • ∃ *c*: *Person* • *motherOf*(*m*, *c*)} **%(**mother definition**)%**
end

spec SAMPLE_GOODS =
sorts *Goods* < *Thing*;
EdibleGoods, *NonEdibleGoods*;
Car < *NonEdibleGoods*
free type *Goods* ::= *sorts EdibleGoods*, *NonEdibleGoods*
end

spec SAMPLECOMB =
SAMPLE_PERS **and** SAMPLE_GOODS
then
free type *Thing* ::= *sorts Person*, *Goods*
∀ *g1*, *g2*: *Person*; *g'*: *Goods*
• *parentOf*(*g'*, *g2*) **%(**ill typed axiom**)%**
then
GENCARDINALITY [**sorts** *Person*, *Car*
pred *hasCar* : *Person* × *Car*]
then
pred *CarCollector*(*p*: *Person*) ⇔ *minCardinality*[*hasCar*](*p*, *5*) **%(**concept definition**)%**
end

Figure 5: CASL-DL example

that the type safety can be checked independently without increasing the complexity of the reasoning with cardinality restrictions. Thus the translated OWL-DL version of the definition of *CarCollector* does not involve the restriction of *hasCar* to be filled only with *Car*.

Also a limited part of basic data types of CASL are usable within CASL-DL. These include i.e. numeric and string values, and they are separated from the sorts representing concepts. Different specifications of these data types are provided by a library, such that every data type can be instantiated on its own. This library also provides the GENCARDINALITY specification for cardinality constraints.

Translations of CASL *to* CASL*-DL* We want to integrate a projection "function" in HETS which translates ontologies written in CASL to CASL-DL ontologies. The taxonomy will be translated automatically, as well as all binary predicate and unary function declarations. The translation of other predicate and function declarations and axioms will be semi-automatic to aid the development of a light-weight ontology from a given FOL ontology preserving as much of the intended meaning as possible. The user of the translation has several options

depending on the complexity of his axioms:

- n-ary predicate and function declarations, with $n > 2$, can be encoded with several new binary predicates,
- axioms which can be rewritten to meet the CASL-DL constraints are simply kept (in the rewritten form),
- disjunctions and conjunctions where some disjuncts or conjuncts can partly be rewritten are kept while the not translatable ones are dropped, and
- the user can provide a CASL-DL formula marked as simplifying a FOL formula which then has to be proved to be implied by the FOL formula.

All resulting encodings and rewritings and dropped axioms are displayed to the user in CASL or CASL-DL formulas for acceptance. They are clearly marked as CASL or CASL-DL. This results in a script that can be reused in the case changes were made to the underlying FOL ontology. Then only translations where changed axioms or changed declarations are involved have to be reconsidered. This script is also a documentation of the translation useful for customers who want an applied light weight version of a well developed CASL ontology.

5 CASL-DL and OWL-DL

We provide a translation of OWL-DL documents to CASL-DL and vice versa, in a way that the OWL-DL ontology is optimised in the sense of CASL's strong typing (s. Table 1 for a summary). By relying on the entailment of OWL DL in $\mathcal{SHOIN}$(**D**) shown in [8, 9] we have constructed a formal description of the translation in [7]. While OWL-DL is untyped and CASL-DL is typed, we interpret the classes of OWL-DL as types in CASL-DL. Thus class definitions in OWL-DL are translated to sort declarations and axioms in CASL-DL. However, for the case that the user really wants to have a unary predicate we provide an annotation property in OWL-DL that leads to a translation of the class to a unary predicate in CASL-DL. OWL-DL property definitions are translated either to declarations of predicates or partial functions and axioms. Individuals in OWL-DL are constant operations in CASL-DL. All facts present in OWL-DL are translated to axioms involving only constant operations.

$\mathcal{SHOIN}$(**D**)	OWL	CASL-DL
concept	class	sort or unary predicate
subconcept	subclass	subsort
number restriction	cardinality	instantiation of parametrised specification
role	property	binary predicate
role inclusion	subproperty	implication
individual	individual	constant operation
$\top$	class Thing	maximum super sort
datatype	XML datatype	predefined datatypes

Table 1: Correspondences between $\mathcal{SHOIN}$(**D**), CASL-DL and OWL-DL

Names of symbols are very different in OWL-DL and CASL-DL. While the former insists on unique names based on URIs [10], the latter supports overloading of symbols since they can be distinguished by their types. The URIs are maintained in CASL-DL via custom

namespace annotations analogous to XML namespaces [11]. Overloaded binary predicates are translated a bit differently from non-overloaded ones. The argument sorts yield two unions of classes (in $\mathcal{SHOIN}$(**D**): $S_1 \sqcup ... \sqcup S_n$) built from the sorts for the first and second argument. These constructs are then used in OWL-DL property definitions for domain and range attributes. Overloaded unary predicates yield subclass definitions in OWL-DL.

6 Applying the Heterogeneous Tool Set and the Prover Isabelle to Ontologies

The Heterogeneous Tool Set HETS is a tool for analysis and proof management of heterogeneous specifications and theories. That is, it can deal with a multitude of formalisms, such as those introduced above: CASL, CASL-DL, OWL and modal logic. Its architecture is depicted in Fig. 6. HETS includes parsing and static analysis tools for a number of languages, as well as tools for translations among them (like the translations addressed in the previous sections). Moreover, it also provides translations to tool-supported logics and interfaces to respective theorem provers, such as Isabelle/HOL [1].

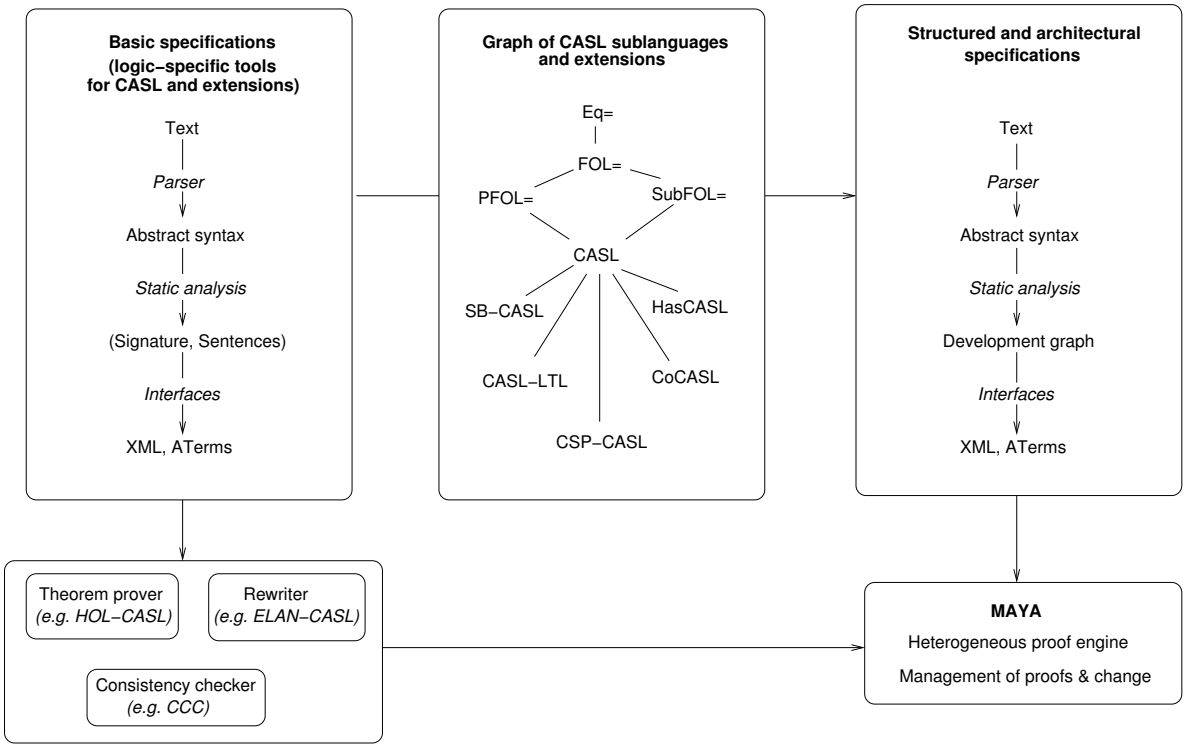

Figure 6: Architecture of the heterogeneous tool set

When (a small part of) the formalization of the DOLCE ontology in CASL is loaded into HETS, a window with a development graph appears (cf. Fig. 7). The dashed arrow starting from the node labelled *GenMereology* to an unlabelled node represents a proof obligation. It can be tackled with the theorem prover Isabelle. Therefore, HETS translates the CASL specification into an Isabelle/HOL theory. We do not show the signature and axioms of this theory here, since it just repeats the CASL declarations. Also, the axiom names used in the proofs stem from the CASL specifications. Finally, the formulas that have to be proved are listed as theorems, but without proof — it is to the user to fill in the proofs.

The first proof (of the intended consequence $Td1$) shows that if every atomic part of x is a part of y, then x is a part of y. This is easily proved by feeding the axioms relevant for the involved concepts into the automatic tactic `best`.

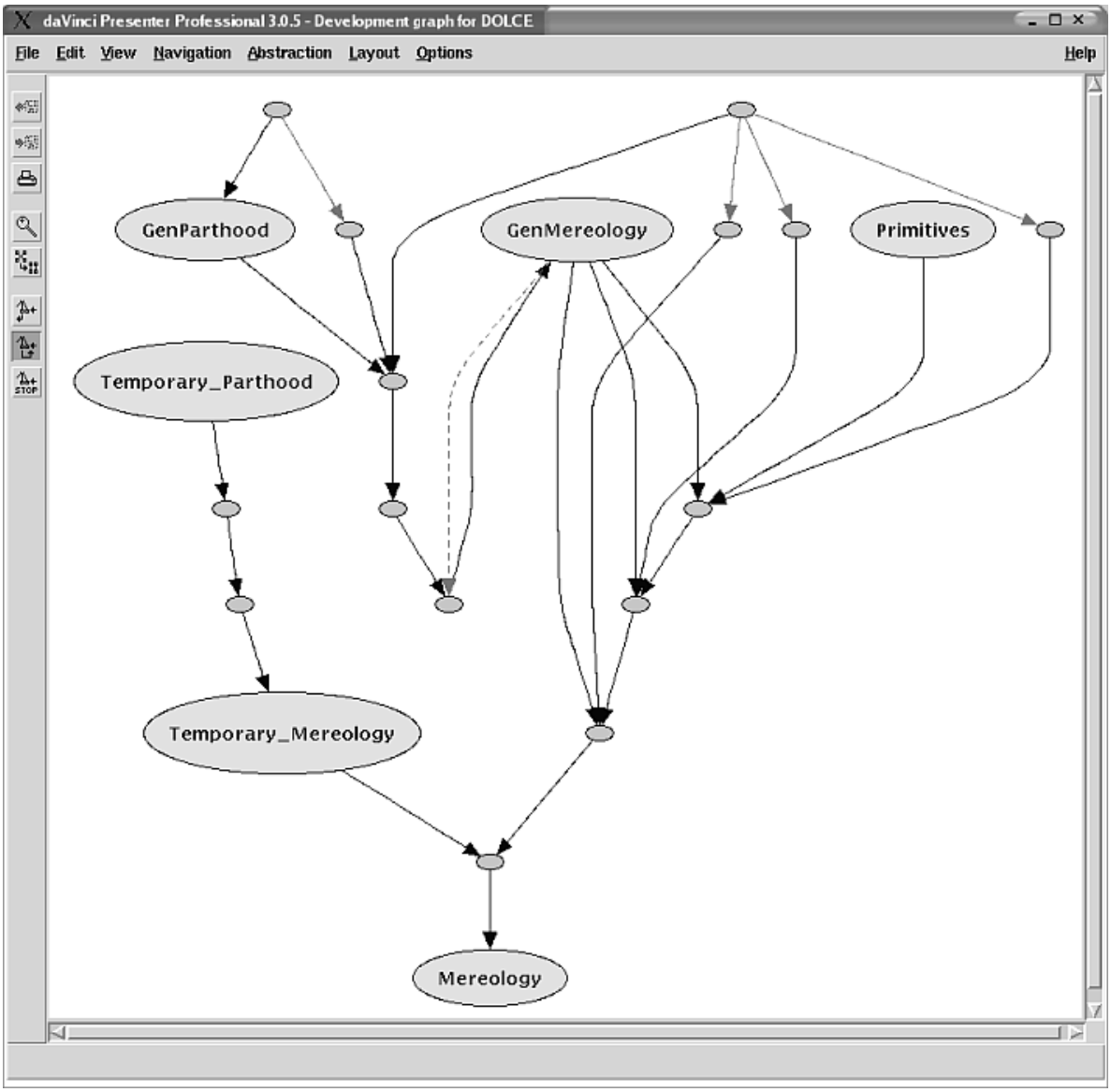

Figure 7: A development graph in the heterogeneous tool set

The second proof (of the intended consequence $Td3$) shows that two objects having the same overlapping parts are equal. The proof is more involved here; it needs a lemma that infers a parthood relation from the equivalence of the (expansion of the definition of) overlapping relation. Actually, we found that Isabelle with the language Isar can be used to formalize proofs in a way that is very close to proofs of the same theorems done with pencil and paper.

```
theorem Td2: "! x :: s . (At x) = (! y' :: s . P y' x --> x = y')"
proof -
  from Ad14 Ad18 Ad13 Dd2_Overlap
  show ?thesis by best
qed
```

```
lemma lemma1:
  assumes aa:"ALL z .(EX u . P u z & P u x)=(EX u . P u z & P u y)"
  shows "P x y"
```

```
proof -
  have h:"ALL v . At v & P v x --> P v y"
  proof
    fix v show "At v & P v x --> P v y"
    proof
      assume ab:"At v & P v x"
      with Ad11 have "EX u . P u v & P u x" by blast
      with aa  have "EX u . P u v & P u y" by simp
      then obtain u where uv: "P u v & P u y" ..
      with ab Dd3_Atom Dd1_Proper_Part Ad12 have "u=v" by blast
      with uv show "P v y" by simp
     qed
  qed
  with Td1 show "P x y" by blast
qed
```

```
theorem Td3: "! x y . (! z . (O z x = O z y)) --> x = y"
proof -
  have "!! x y. (ALL z. O z x = O z y) ==> x = y"
  proof -
    fix x y assume p:"(ALL z. O z x = O z y)"
      with Dd2_Overlap
      have a:"ALL z .(EX u . P u z & P u x)=(EX u . P u z & P u y)"
                by simp
      with lemma1 have "P x y" by blast
      moreover
      from a lemma1 have "P y x" by blast
      ultimately show "x = y" using Ad12 by simp
    qed
    thus ?thesis by blast
qed
```

Conclusion

We have shown how the specification language CASL can be used for the definition and specification of ontologies. The main advantages of this approach are

- strong typing, allowing the static detection of errors while keeping flexibility through the modelling of subconcepts as subsorts,
- powerful structuring constructs, allowing the reuse of definitions in ontologies, and also a more economic way of theorem proving: for a parameterized specification, a theorem needs to be proved only once, but then it carries over to all instantiations,
- and last but not least a readable and standardized input syntax (together with an automatic formatting).

We have interfaced the theorem prover Isabelle with the heterogeneous tool set HETS. The integration of further provers (e.g. tailored towards specific sublogics of CASL) as well

as consistency checkers into HETS is the subject of current research. An interesting open question is whether the decomposition of models into smaller ones given by CASL architectural specifications provides a practically useful way of reducing the consistency problem of larger ontology specifications into that of smaller ones.

References

[1] T. Nipkow, L. C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Springer Verlag, 2002.

[2] CoFI (The Common Framework Initiative). CASL *Reference Manual*. LNCS 2960 (IFIP Series). Springer, 2004.

[3] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening ontologies with dolce. In Gómez-Pérez and Benjamins [12], pages 166–181.

[4] M. R. Genesereth and R. E. Fikes. Knowlegde interchange format version 3.0 reference manual. Stanford Logic Group, Report Logic-92-1 `http://logic.stanford.edu/sharing/papers/kif.ps`, June 1992.

[5] I. Horrocks and P. F. Patel-Schneider. Reducing OWL Entailment to Description Logic Satisfiability. In D. Calvanese, G. De Giacomo, and E. Franconi, editors, *Proceedings of 2003 International Workshop on Description Logics*, volume 81 of *CEUR Workshop Poceedings*, 2003.

[6] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[7] K. Lüttich, T. Mossakowski, and B. Krieg-Brückner. Ontologies for the Semantic Web in CASL. In *Recent Trends in Algebraic Development Techniques*, LNCS. Springer, WADT 2004. submitted.

[8] I. Horrocks and P. F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, number 2870 in Lecture Notes in Computer Science, pages 17–29. Springer, 2003.

[9] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[10] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic syntax. IETF Draft Standard (RFC 2396) `http://www.ietf.org/rfc/rfc2396.txt`, August 1998.

[11] T. Bray, D. Hollander, and A. Layman, editors. Namespaces in XML. W3C Recommendation `http://www.w3.org/TR/REC-xml-names/`, 14 January 1999.

[12] A. Gómez-Pérez and V. R. Benjamins, editors. *Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, 13th International Conference, EKAW 2002, Siguenza, Spain, October 1-4, 2002, Proceedings*, volume 2473 of *LNCS*. Springer, 2002.