

# Interpretation Logics

[OPAALS Conference 2007]

Ossi Nykänen

Institute of Mathematics, Hypermedia Laboratory  
Tampere University of Technology  
Tampere, Finland  
ossi.nykanen@tut.fi

**Abstract**—Integration of incompatible axiomatic theories is a concrete obstacle in developing logic-based applications. This challenge is faced, e.g., when integrating legacy databases, rule-based knowledge systems, and domain-specific ontologies. Further, many applications do not exploit the full range of properties of the underlying logic languages that typically leads into unnecessary complex, implicit overhead in reasoning. We propose a generic framework called interpretation logics for integrating mismatching knowledge bases, and explicitly designing the related interpretation system. We claim that the integration of formal theories based on a linguistic transformation and application-specific, asserted interpretation axioms, with respect to interpretation logic, is not only feasible but also desirable, due to the common challenges of application development. We outline the conceptual framework, demonstrate its applicability with an example, and discuss implementations with respect to Semantic Web technologies.

*Interpretation logics, Knowledge transformation, Semantic interoperability*

## I. INTRODUCTION

Logic provides a system for mechanically working with knowledge. The usefulness of logic builds upon the two fundamental tasks of mathematics: the descriptive task and the deductive task [10]. The abstract knowledge engineering process captures the motivation of computerized knowledge systems well: decide what to talk about; decide on a vocabulary of predicates, functions, and constants; encode general knowledge about the domain; encode a description of the specific problem instance; and pose queries to the inference procedure and get answers [21]. While this is straightforward in principle, serious problems usually arise in applications, at least when different systems are to be integrated.

Despite the evident hype, logic-based models are today commonly found in mainstream applications such as software engineering, configuration, medicine, digital libraries, and Web-based information systems [2]. Prominent new areas of work include applications for web portals, multimedia collections, corporate web sites, design documentation, agents and services, and ubiquitous computing [8]. Visions of modern applications range from publishing business rules to discover the potential of enterprise collaborations to aggregating the family photo albums [19][23].

Indeed, the standardization of the representation languages and the ease of publishing and accessing data over the Internet,

have made it possible to compile and semantically integrate logical data and exploit it in a variety of local applications. However, strict integration of different axiomatic and slightly incompatible theories soon becomes an obstacle in practical application development. This happens, e.g., when integrating legacy systems, or when combining rule-based knowledge systems and domain-specific ontologies.

In this article, we present a generic framework called interpretation logics for integrating a series of available, mismatching knowledge bases, and explicitly designing the related interpretation system. Intuitively, interpretation logics may be perceived as an abstract design pattern, with logic-specific characteristics of more general interfaces and proxy design patterns (see, e.g., [6]). Due to its popularity in software engineering, the study of design patterns is emerging in several research topics, including ontology design (see, e.g., [1]).

The basic idea is that instead of mixing the available logical data within a single "rich enough" logical framework, applications should be based on explicit design decisions of what formulas to include and with what logical axioms. Thus, the interpretation theory is not simply a "passive" mixture of the theories of the available knowledge bases, but an "active" interpretation of them, involving application-specific choices.

The main motivation for our work stems from the research needs of a large European Network of Excellence Project OPAALS (Open Philosophies for Associative Autopoietic Digital Ecosystems; see <http://www.opaals.org/>) coordinated by London School of Economics and Political Science. The overall objectives of the OPAALS project are to build a sustainable interdisciplinary research community in the emerging area of Digital Ecosystems (DE, <http://www.digital-ecosystems.org/>) and to develop an integrated theoretical foundation for DE research. In the project, the problem of integrating mismatching knowledge bases is faced in two major research activities: language evolution and visualization. In the former research activity, interpretation logics provides a method for working with bottom-up ontologies, i.e., interpretation theories built with the contextual organization of the available knowledge bases. In addition, interpretation logics provide a concrete framework for studying the integration of domain ontologies and business rules. In the latter activity, interpretation ontologies (or theories) are used for mapping the available knowledge bases onto visualization models [18]. This allows abstracting the concepts of the few general-purpose

visualization models from the numerous specific-purpose knowledge models.

We claim that working with interpretation logics not only provides a concrete, easy-to-deploy framework for integrating logical models (because tools can be chosen more freely), but also significantly clarifies application development. This is due to the fact that designers are allowed to make transparent design decisions about the reasoning process in local applications. This in turn helps making designs more understandable, at least by providing an intermediate venue for pinpointing the potential misunderstandings.

In principle, one can easily argue against this approach by saying that interpretations based on transformations are not needed since there are several good and standardized general-purpose knowledge representation languages. However, a short tour on the Semantic Web applications (see, e.g., [9][3]) of Resource Description Framework (RDF), Web Ontology Language (OWL) and Semantic Web Rule Language (SWRL), reveals that there are known practical problems with the "one size fits them all" approach. For instance:

1. Useful knowledge representation languages have often been developed in parallel, without a tight integration of the theory or the tools. For instance, the interplay of rule systems and ontology systems in reasoning applications is often defined only ad hoc. Consider, e.g., the management of the asserted and the inferred knowledge in Protégé with a DIG-compliant reasoner (for an overview of Protégé and OWL, see, e.g., [13][11]).
2. Integrating legacy data with ontologies is often challenging, due to the fact that the ontology languages seriously limit the use of expression of the language. For instance, OWL DL ontologies do not allow user-defined properties between individuals and classes [22].
3. Standard logics may include axioms that are not needed for every application. For instance, the Jena reasoner provides an explicit option of omitting some of the usually non-needed but expensive consequences of reasoning [20].
4. There are design errors and inconsistencies in the application theories. For instance, the RDF Schema is still commonly misused for "checking predicates' constraints" rather than asserting the valid consequences of their usage.
5. The availability of correctly working reasoning tools is a concrete problem; the available tools typically support only few language/theory/logic combinations. For instance, at the time of writing, the standardization efforts for a Semantic Web rule language are still underway [7].
6. In many cases, the modeling decisions of the legacy systems are fundamentally incompatible which means that explicit and constructive transformation processes are needed at any case. For instance, integrating local

models typically requires actions related to the global naming of resources.

The above kinds of restrictions have led into introducing documented workarounds. For instance, the relatively simple restriction 2 above, which follows from the fact that OWL DL models classes as sets, is explicitly dealt as a Semantic Web best practice on representing classes as property values [16].

As suspected, the notion of interpretation logics is not tied to any particular logic system. However, for our purposes, implementing interpretation logics as Prolog programs has turned out to be a good compromise. There are several established tools available for various platforms and embedding Prolog interpreters to applications is a common practice. The need for using, e.g., Prolog tools explicitly may well change as the standard Semantic Web tools mature. Nevertheless, the availability of tools is an important issue in application development.

The interplay of ontologies and rule systems also seems to support this design decision. While it is generally agreed that, e.g., ontology systems and rule systems genuinely complement each others (neither subsumes the other as a logic unless we choose "unusually rich" ontology or rule systems), Prolog seems sufficiently expressive for capturing many useful OWL (DL) ontologies (see, e.g., [24]).

Because of its expressive power and availability, Prolog has also been suggested as a implementation strategy for the SWRL rules [15]. Popular Prolog systems, such as the SWI-Prolog, also include readily-available packages for managing Semantic Web data [25]. Wielemaker's and his colleagues work also includes, e.g. implementations of Semantic Web query languages and ontology editors [26][27]. As such, SWI-Prolog has been used for implementing Semantic Web reasoning in successful large-scale projects such as the MuseumFinland [12]. The specific challenge of managing heterogeneous ontologies has been extensively studied, e.g., in terms of ranked mappings of descriptions of resources onto OWL DL ontologies with appropriate complexity [4].

Finally, while implementing declarative logic programs is relatively straightforward, the computational complexity of logical queries is typically very high. Indeed, while the notion of interpretation logics might be perceived more as a design pattern than a theory, application developers face the practical problem of writing both expressive and sufficiently fast logic programs. Because of the theoretical and practical significance, the complexity and expressive power of logic programming and description logics has been extensively studied (see, e.g., [2][5]). The results indicate that while tractability and completeness can be achieved with suitable restrictions, the complexity of practical applications tends to be non-polynomial. Intuitively this means that large applications slow down very quickly. While interpretation logics can not obviously solve this problem, their explicit nature enables clarifying the complexity of applications. The basic method is including only the particular deductions (whose expected complexity is known and accepted) that are explicitly required by the target application.

The rest of this article is organized as follows: Section 2 describes the basic concepts and general definitions in more detail, by depicting interpretation logics in a relationship with a series of given initial knowledge bases. The basic idea of applications is then illustrated with an example in Section 3, with respect to specific Semantic Web technologies. Section 4 finally concludes the article with technical remarks and discussion.

## II. BASIC CONCEPTS

In order to be able to discuss interpretation logics, we need to establish two main concepts: logic and theory. Interpretation logics and theories build upon these concepts by providing an explicit venue of language transformations, asserted interpretation axioms, interpretation systems, and finally concrete implementations.

### A. Characterization of Logic Systems

We call a system  $L$  a logic (system) if it introduces and associates the following three meaningful components:

1.  $L$  establishes an object language of well-formed formulas (wff), denoted by  $F(L)$ .
2.  $L$  introduces an interpretation system  $R(L)$ .
3.  $L$  explains its concepts and reasoning process in terms of a meta language  $M(L)$ .

The language  $F(L)$  serves as the principal medium of capturing asserted knowledge with respect to  $L$ . If  $F(L)$  is a formal language we may consider  $F(L)$  as a potentially infinite set of strings over a finite alphabet, usually defined in terms of a finite grammar. Examples of wffs include ground wffs such as facts and rules, and schematic wffs, typically intuitively simple string-rewriting functions. Schematic wffs are usually operated in the meta language, conditionally outputting ground wffs "upon request", based on valid parameters. (In implementations, schematic wffs might be implemented in terms of macros or pre-processing systems.)

The interpretation system  $R(L)$  essentially provides a way of reasoning, i.e. finding out whether a set of wffs  $B$  intrinsically follows from another set of wffs  $A$  (often denoted  $A \rightarrow B$  or  $A \supset B$ ). The reasoning system typically includes an algebraic component that allows equating and thus simplifying wffs that appear as different words in the language but have the same meaning with respect to the interpretation system. When  $R(L)$  is expressed formally, it typically introduces a set-theoretic or truth-functional interpretation theory which aims justifying, e.g., the soundness and completeness of the inference method. Further, the tractability and costs of inference may be explained with respect to worst-case or expected value estimates of computational complexity.

The meta language  $M(L)$  is needed for humans for communicating and analyzing the former two components in a sufficiently clear and understandable way. In most cases, meta language is some natural language such as English, equipped with some mathematical notation.

Note that according to the above characterization, a system may be considered "logical" even if it is not "correct", "consistent", etc.

We are now ready to define our second main concept, theory. A theory  $T_L$  with respect to a particular logic  $L$ , is simply a collection of wffs,  $T_L \subset F(L)$ . Further, when a theory  $T_L$  is interpreted, we may think that the underlying logic  $L$  establishes a subsumed (and often potentially infinite) set of implied wffs that are not explicitly asserted by  $T_L$ . (Note that interpretation systems typically explain this behavior in terms of models of potential worlds rather than linguistically with the implied wffs.) Any changes in the theory or in the interpretation system usually change the set of implied wffs of the theory.

In short, a theory provides a logical justification for both the explicitly asserted wffs (which can be usually read from the set  $T_L$  when the theory is finite) and the implied wffs (whose existence needs to be verified with proofs). While the set of wffs is in principle arbitrary, useful theories are usually used to describe complex applications with few and understandable wffs, providing a concise and consistent system for making rigorous but rich deductions. While the concept "truth" is often found in the semantics, a theory does not really need to make particular ontological commitments. A theory simply captures the aspects of the application the designers have found useful.

A logic and hence a theory may be considered formal or informal. The notion of formality typically accounts to the level of detail and consistency in describing the object language and the method of inference. Examples of well-known formal logics include propositional logic, first-order predicate calculus (and the related logics such as pure and full predicate logic, modal logic, temporal logic, fuzzy logic, and probabilistic logic), RDF Schema, and OWL (and its sublanguages OWL Lite, OWL DL, and OWL Full) (for a list of core Semantic Web references, see [9]). Explicit rule languages that could be accounted as logics include e.g., abstract Horn Clause Logic and concrete systems such as Datalog, Prolog, SWRL, and their numerous relatives (see, e.g., [5] [15]).

Not every useful logic need to be considered formal, though. The best known examples of useful informal logics include common-sense reasoning and, e.g., semi-formal systems that operate with formally defined wffs with a reasoning procedure but without a detailed or an written out interpretation theory. Note that semi-formal and formal systems may in effect provide similar kinds of reasoning services, e.g., subsumption and satisfiability, even if the "quality" of reasoning might not be explicitly known in the case of the semi-formal systems. In practice, many kinds of algebraic structures, string-rewriting systems, and automata may be considered semi-formal logics in this sense. With some limitations, we may also conceptualize e.g. mathematical proofs as semi-formal logic systems, even if the object language is not usually fully formalized and reasoning depends heavily on structures and mathematical intuition expressed only on the level of the meta language.

The reasons for working with semi-formal logics vary. Perhaps the most common motivation for not aiming for a

completely formal theory is the complexity of the task, the estimated cost/gain ratio of the effort, and the fact that the occasionally resulting inconsistencies due to poor design may be tolerable in applications. As a consequence, formalization often results only after the informal approach has for some reason been found insufficient in application development.

For purposes of this article, it is interesting to record two well-known observations.

1. Theories do not usually exploit all of the available structures of the object language, nor require all the aspects of the interpretation system.
2. Applications that claim to work with a particular logic, are sometimes in fact working with a (slightly) modified logic, perhaps adding or removing some wffs considered as axioms.

The first point implies that in some cases, theories could be captured with a simpler logic. For instance, certain applications of full predicate logic may be expressed in suitable description logic. The second point implies that in some cases this might also be unintentional, e.g., due to misunderstandings. For instance, a logical theory may include a contradiction or convenient extra axioms whose rigorous implications (usually nullifying the usefulness of the theory in a strict sense) are neglected due to the lack of systematic consistency checking (tools).

We claim that both of these issues are faced commonly in application development. While these observations might simply be considered as poor modeling decisions (using an overly complex modeling language), or errors in reasoning (not conforming to the claimed logic), they hint the existence of useful interpretation logics, in which these selections are made knowingly, as active design decisions.

### B. Interpretation Logics and Interpretation Theories

Assume a list semi-formal or formal theories  $\langle T_i \rangle$  with the associated logics  $L_i$ . Let us call these the initial knowledge bases.

Define an interpretation theory  $T_i$  (with respect to an interpretation logic  $L_i$ ) as follows:

1. Assume  $L_i$  is a suitable logic.
2. Establish a list of language transformations  $g_i$  that filter and adapt the interesting formulas from the theories of  $\langle L_i \rangle$  to the object language of  $L_i$ .
3. Assert a suitable set of wffs denoted by  $D$ , and set interpretation theory  $T_i = \cup_i g_i(\langle T_{L_i} \rangle) \cup D$ . The set  $D \subset T_i$  is called the set of interpretation axioms of  $T_i$ .

An interpretation logic  $L_i$  is defined in association with an interpretation theory, by establishing  $R(L_i)$  and  $M(R_i)$  so that  $L_i$  appropriately captures the intended reasoning application.

In applications, the interpretation logic is typically selected first with a rough idea of its reasoning capabilities. Then, an interpretation theory is compiled and enriched with appropriate axioms. Finally, the interpretation of the interpretation logics is refined to match the needs of the target application.

The usefulness of the above definitions is based on the idea that useful knowledge bases include wffs that can provide a rough basement for the application-specific formulas of  $L_i$ . According to this design stance, the user may choose to utilize the formal linguistic constructions as a part of his or hers own logic system freely, adding interpretation axioms when convenient.

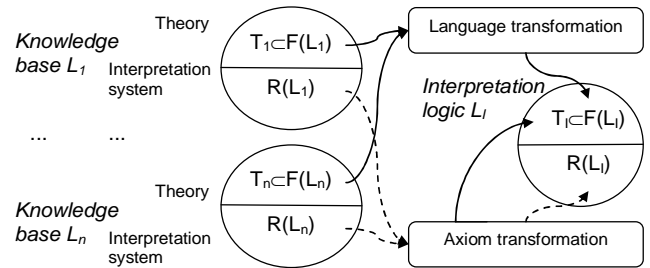


Figure 1. An outline process for working with interpretation logics

The basic setting of interpretation logics discussed above is illustrated in Figure 1. The role of the meta language(s) is not explicitly depicted since we may think that it is implicitly present in the whole setting. Further, the process of asserting the interpretation axioms is symmetrically depicted as a kind of transformation (represented by dotted lines in the figure), even if it typically appears asymmetrically in applications, with a somewhat less mechanical flavor.

Assuming the knowledge bases  $\langle L_i \rangle$  are compatible with matching languages and interpretation systems, constructing the interpretation logic may be nearly trivial. For instance, assuming  $\langle L_i \rangle = \langle L_1, L_2, L_3 \rangle$  where  $L_1$  and  $L_2$  are propositional logics and  $L_3$  is a pure predicate logic, with the associated theories  $T_1 = \{A, A \wedge B\}$ ,  $T_2 = \{A \vee B\}$ ,  $T_3 = \{\exists x: C(x) \rightarrow A\}$ , we easily get a meaningful propositional logic  $L_1$  with a theory  $T_1 = \{A, A \wedge B, A \vee B, \exists x: C(x) \rightarrow A\}$  with some familiar interpretation theory and method of inference (which might allow simplifying the theory into some shorter, and "simpler", but semantically equivalent form, such as  $\{A \wedge B, \exists x: C(x) \rightarrow A\}$ ).

When the language and the method of inference of the interpretation logic are sufficient for completely capturing the knowledge bases, we say that the interpretation logic is conformant. Completely conformant interpretation logics (as above), when applicable, are usually uninteresting in the sense that they simply aim introducing a maximally rich logic for capturing the knowledge bases. As expected, most useful interpretation logics are non-conformant, e.g., modifying the language and/or the interpretation system. This means that the designer establishing the interpretation actively selects and modifies the formulas for the language of the interpretation logic, and decides which axioms are included in the reasoning process.

An important special case of interpretation logics is provided by interpretation ontologies. Here the term ontology refers to a suitable ontology language, typically a restriction of some predicate logic. Interpretation ontologies are important simply because of the pivotal role of OWL (DL) in many concrete applications.

Note that the interpretation system of the interpretation logic may also be weaker than the interpretation system of a knowledge base, and still be able to meaningfully capture the key applications semantics. This is usually due to the fact that all theories do not fully exploit the properties of the underlying logics. However, the language transforming process may also map a wff in  $F(L_i)$  into multiple wffs in  $F(L_i)$ , perhaps "writing out" some significant deductions. This is particularly the case when language transformations are allowed to process schematic wffs. In other words, the language transformation process effectively includes information about the intended reasoning process of the interpretation logic. (Of course, in most cases, e.g., writing out all deductions as ground wffs is impossible due to the large or potentially infinite number of them etc.)

Clearly, non-conformant interpretation logics actually change something in the language and/or in the interpretation of it. The (iterative) process of constructing useful non-faithful interpretation logics must be managed with care, requiring substantial understanding of the knowledge bases and the target application.

### III. EXAMPLE

Let us next consider a simple example that aims for combining three knowledge bases,  $K_1$ ,  $K_2$ , and  $K_3$  in terms of an interpretation theory in Prolog. To get a flavor of interpretation logics of useful applications, the wffs of initial knowledge bases include statements about individuals, ontology, and if-then rules using Semantic Web technologies. Using Semantic Web technologies simplifies the task of language transformations and the selection of the interpretation logics since the languages share the principal modeling characteristics.

#### A. Initial Knowledge Bases

Our target application involves capturing (research) interests of different people, based on data provided by registered users of a knowledge-sharing system. To keep our example straightforward, we consider only a very simple use case. We would like to provide a search mechanism for finding out people based on their interests (and vice versa), and to discover potential "weak associates", i.e., people working with similar topics, perhaps without knowing about each others' work.

Our first knowledge base  $K_1$  includes two statements about individuals in RDFS (see, e.g., [14]) written in the Turtle syntax:

```
<http://www.foo.org/person_Paul>
  <http://xmlns.com/foaf/0.1/topic_interest>
    <http://archive.astro.umd.edu/ont/Science.owl#Math>.
<http://www.foo.org/person_Susan>
  <http://xmlns.com/foaf/0.1/topic_interest>
    <http://archive.astro.umd.edu/ont/Science.owl#Algebra>.
```

Strictly speaking, the first statement roughly says "there is something called (...Paul) that exists with a relationship called (...topic\_interest) with something called (...Math)". A natural interpretation might be that Paul is interested in mathematics.

The second statement is similar, with a natural interpretation that Susan is interested in algebra.

If we really would like to interpret the statements in RDFS (which is too expressive since RDFS-specific constructs are not actually used), there are more points to be made. In particular, according to RDF Semantics there is, e.g., no assumption that this is all there is nothing more to be known about the resources, nor that the names would actually refer to different things. Further, the interpretation system states that by default, each knowledge base of RDFS statements implicitly includes the so-called RDF and RDFS axiomatic triples. In practice, these include not only ground wffs such as `rdf:type rdfs:domain rdfs:Resource.`, but also schematic wffs, for instance `rdf:<<N>> rdfs:domain rdfs:Resource.` where `<<N>>` stands for an arbitrary identifier of a membership property, e.g. `_1`.

Our second knowledge base  $K_2$  (that is an excerpt of a real ontology by E. Shaya (see <http://archive.astro.umd.edu/ont/index.html>)) includes statements in OWL DL, written in the RDF/XML syntax (see, e.g., [22]):

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-
rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/
01/rdf-schema#" xmlns:owl="http://www.w3.org/2002/
07/owl#"><owl:Ontology rdf:about="http://archive.
astro.umd.edu/ont/Science.owl#" />
  <owl:Class rdf:ID="Algebra">
    <rdfs:subClassOf rdf:resource="#Math" />
  </owl:Class>
<owl:Class rdf:ID="Math" />
</rdf:RDF>
```

Intuitively, these statements assert that the given ontology includes a class Algebra that is a subclass of the class Math. Again, strictly speaking, we have not asserted that the two classes are actually different things (but in principle OWL would allow us to do so), a set of axiomatic triples is implicitly assumed, etc. Further, the knowledge base  $K_2$  accounts as an OWL Lite ontology since it only exploits a restricted set of the OWL language. (Actually declaring that the classes are disjoint would have resulted an OWL DL ontology.)

Finally, our third knowledge base  $K_3$  includes a rule in SWRL, written in the "human readable" SWRL syntax (see [15]):

```
hasInterest(?p1, ?s) ^ hasInterest(?p2, ?s) ->
hasWeakAssociate(?p1, ?p2)
```

Intuitively, this rule asserts that whenever two persons share a common interest, they become weak associates. Again, strictly speaking, the rule simply asserts that whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold, a set of rule axioms is assumed, etc.

#### B. Issues of interpretation

We would now like to establish an interpretation theory  $K_1$  in Prolog for purpose of an application that wishes to integrate the knowledge bases  $K_1$ ,  $K_2$ , and  $K_3$  which are theories in RDF Schema, OWL DL, and SWRL logics. Intuitively, we would like to use the knowledge bases for reasoning about the

interests of Paul and Susan, perhaps to find out whether the interests of Paul and Susan overlap.

However, certain known issues exist that make simply aggregating the knowledge bases problematic. For instance:

1. The statements in  $K_1$  relate Paul and Susan with resources that are classes according to  $K_2$ . Relating individuals with classes is not allowed for an object property in OWL DL.
2. The names in the rule of  $K_3$  do not match with the names found in  $K_1$  and  $K_2$ .
3. The knowledge bases include lots of implicit axioms, but assert, e.g., no disjointness relations.

In a concrete application, we might have a priori knowledge about these things that we would like exploit in the interpretation. In particular, for purposes of our particular application, we would perhaps like to give up the open world assumption and drop several unnecessary wffs, to make reasoning more straightforward, discarding uninteresting deductions along the way.

Indeed, while with certain assumptions, we might be able to capture the knowledge bases  $K_1$ ,  $K_2$ , and  $K_3$  with sufficiently rich conformant theory and interpretation logic (something like "OWL Full + SWRL" logic), we would not usually wish to do so. At least we would like to glue the useful names together (e.g. the properties `topic_interest` and `hasInterest` are equivalent), add disjointness axioms (Paul and Susan are names of different things), and ensure the tractability of our system, thus effectively ending up with a new theory.

Note that we could in principle solve these problems by redesigning the knowledge bases and harmonizing the modeling. However, in practice, the knowledge bases might not be under our control, or they might include other, more important use cases.

With these issues in mind, we will next compile our interpretation theory  $K_1$  in terms of a Prolog program (some other logic system might be applicable as well). After deciding basic modeling strategy, the work involves making a series of language transformations and asserting appropriate interpretation axioms.

### C. An interpretation logic in Prolog

Let us first decide encoding for the knowledge base  $K_1$ . A general and a well-known approach is encoding RDF statements as triples (we omit the URL prefixes for brevity; in a concrete application the name 'Paul' would be represented as 'http://www.foo.org/person\_Paul', some literals might be associated with datatypes, the knowledge base might include anonymous terms [nodes], etc.):

```
t('Paul', 'hasInterest', 'Math').
t('Susan', 'hasInterest', 'Algebra').
```

Note that we changed the name of the predicate, due meta-reasoning and selection (`topic_interest` and `hasInterest` are equivalent but we are only interested in deductions involving the name `hasInterest`).

Then, let us assert the terminological information related to the second knowledge base  $K_2$ :

```
t('Algebra', 'subClassOf', 'Math').
```

Note that we, e.g., discarded the information about the existence of an ontology and that Algebra and Math are classes since we have no use for this piece of information in our application. Further, we have not included any RDF(S) axiomatic triples so far because our application does not have any use for them.

Finally, we transform the third knowledge base  $K_3$  into a Prolog rule:

```
t(Vp1, 'hasWeakAssociate', Vp2) :-
t(Vp1, 'hasInterest', Vs), t(Vp2, 'hasInterest', Vs).
```

In this case, we have again discarded axioms of the SWRL logic from our interpretation theory.

Now it is time to assert some interpretation axioms of our own. The objective is of course a definition of a useful system that captures the aspects of interest from the initial knowledge bases, for purposes of the target application (a real application would obviously need a more elaborate description in the meta language).

Let us first assert a procedure that essentially states that the `subClassOf` predicate is transitional. We can achieve this e.g. by introducing a support procedure `transitiveSubClassOf`:

```
t(A, 'transitiveSubClassOf', B) :-
t(A, 'subClassOf', B).
t(A, 'transitiveSubClassOf', B) :-
t(S, 'subClassOf', B), t(A, 'transitiveSubClassOf', S).
```

In general, formulations of transitive properties typically include potential pitfalls in logic programming, resulting to endless queries. Of course, writing support procedures in triples format is not strictly necessary.

We shall next assert that being interested in something particular implies interest in the topic in general. For this, we exploit the transitive definition of the subclass property:

```
t(Vp, 'hasInterest', Vsuper) :-
t(Vsub, 'transitiveSubClassOf', Vsuper), t(Vp, 'hasInterest', Vsub).
```

This concludes the definition of our interpretation theory  $K_1$ .

We may now use our interpretation logic system for working with the interpreted knowledge. In the case of a Prolog interpreter, the basic tool is making queries. For instance, we might be interested in who is interested in what. The query `?- t(X, 'hasInterest', Y). now` returns:

```
X = 'Paul' Y = 'Math'; X = 'Susan' Y = 'Algebra'; X = 'Susan' Y = 'Math'; No
```

Of course, we can ask more explicit questions such as "What is Paul interested in?"

We may also ask that who have weak associations based on mutual interests. The query `?- t(X, 'hasWeakAssociate', Y).` seemingly does the trick. It returns:

```
X = 'Paul' Y = 'Paul'; X = 'Paul' Y = 'Susan'; X =
'Susan' Y = 'Susan'; X = 'Susan' Y = 'Paul'; X =
'Susan' Y = 'Susan'; No
```

In other words, we get all the conceivable combinations of persons that are interested in same things, either based on a direct fact, or transitive rule based on the available taxonomic knowledge of the topics of interest.

A nice thing about our application is that it includes only the reasoning potential more or less explicitly stated by the interpretation theory  $K_I$  and modifying the interpretation is easy. For instance, if we would like to further refine our interpretation by discarding symmetric properties, we could simply rewrite `hasWeakAssociate` as follows:

```
t(Vp1, 'hasWeakAssociate', Vp2) :-
t(Vp1, 'hasInterest', Vs), t(Vp2, 'hasInterest', Vs), Vp1 \
=Vp2.
```

Finally, from applications' perspective, it is sometimes beneficial to map the deduced results further onto some other interpretation logics. For instance, should the deductions outlined above be integrated with Semantic Web applications, we might assert the Prolog query results in terms of a suitable RDF vocabulary. When the queries include only concatenated triple terms  $\wedge_j t_i(p_{j1}, p_{j2}, p_{j3})$  as in `?- t_1(Vp_11, Vp_12, Vp_13), t_2(Vp_21, Vp_22, Vp_23), ..., t_n(Vp_n1, Vp_n2, Vp_n3)`. we may trivially write out the retrieved variable configurations in some RDF format. For instance, a result configuration `X = 'Paul' Y = 'Susan'` of the query `?- t(X, 'hasWeakAssociate', Y)` yields `Paul hasWeakAssociate Susan`. (with the exception that applications would of course use global URI names). Transforming this into, e.g., RDF/XML, is straightforward.

#### D. From convenience methods to knowledge interfaces

Interpretation logics do not need to follow the linguistic or interpretative conventions of the initial knowledge bases. For instance, we may assert convenience methods as support procedures, e.g., classifying names in order to be able to easily list all the known persons and their properties with a query `?- t(X, 'type', 'Person')`. (Note that in this case, asserting the required wffs can be achieved either with the language transformations or with the interpretation axioms.)

Various kinds of convenience methods may be used to encapsulate the internal complexity of the initial knowledge bases. This design perspective allows using interpretation theories as knowledge interfaces, enabling implementing knowledge views.

We may also exploit the language components of the initial knowledge bases in a setting of non-standard logics, e.g., probabilistic and fuzzy logics. It is important to realize that this not only enables integrating implied wffs according to non-conformant crisp reasoning, but also induced wffs, due to statistical or imprecise reasoning. Effectively, this allows integrating logical theories with predictions (e.g. via manually designed probabilistic models or machine learning algorithms) and wffs due fuzzy reasoning (e.g. classifications and deductions associated with fuzzy models etc.).

For instance, a natural application for fuzzy logic exists in our example: simply saying that because Susan is interested in Algebra she is also interested in Math is too vague. It seems natural to think that weak associations between people, based on their interests, may have varying degrees, according to the closeness of the topics of interest. In particular, assuming a tree-like taxonomy of interest, a person that asserts being interested in the top concept (now Math) would participate in every weak research group according to  $R_I$ !

This observation leads into a notion for computing degrees for weak associations, e.g., based on the (shortest) distance of topics in a topic graph (a taxonomy tree in our example). Of course, the intuition of the degree of weak association may well be captured with different kinds of definitions of fuzzy membership.

Coming up with extensions like this does not necessary mean giving up Prolog etc. For instance, one may use the language of Prolog to capture wffs associated with fuzzy models, and implement reasoning procedures of fuzzy logic [17]. However, the interpretation system might need re-evaluating, giving explanations of the newly computed results in the meta language. In fact, this interplay of interpretation system and meta language is already present in our simple example, e.g., when considering whether Susan and Paul are different individuals and how this should be interpreted in the application.

## IV. CONCLUSION

In this article, we have presented a generic framework called interpretation logics for integrating a series of available, mismatching knowledge bases and explicitly designing the related interpretation system. We conclude the article with few notes about implementation and applications.

When working with RDF/XML data and Prolog, relatively natural implementation architecture exists. The knowledge bases are first identified as data sources and mapped onto RDF/XML, e.g. with the help of GRDDL techniques. This raw data is then queried and filtered into an appropriate intermediate RDF/XML form using query and transform processors, effectively performing the language transformation step (with the help of, e.g. SPARQL, XQuery 1.0 and XSLT 2.0). The transformed RDF/XML data is serialized into a Prolog program (as triples), and associated with the interpretation axioms, e.g. again in terms of XSLT processing. The Prolog interpreter is then consulted for interesting queries. Finally, the results are applied as such, or transformed back into the RDF/XML format, perhaps to be integrated with the original knowledge bases or data sources. (About the referenced acronyms and technologies, see, e.g., <http://www.w3.org/TR/>.) In applications, practical challenges include user interface design, Unicode tool support, modeling issues and ensuring the speed of queries, error management and understandable reporting, and applications-specific issues such as the logical interpretation of inferred data in applications.

To support these activities, we have implemented a system called *tinker* that is capable of reading RDF/XML data sources,

carrying out the reasoning subtask with asserted axioms, and outputting conclusions in RDF/XML, as described above. Considering our OPAALS visualization use case, tinker can be fitted into the visualization pipeline architecture (see [18]), as a part of a complete series of pre-processing, query, reasoning, and transforming components. In practice, this also requires implementing an additional wrapper layer on top of the tasks outlined above.

Finally, while the framework of interpretation logics does not solve the issues of, e.g., transparency, expressiveness, and non-polynomial complexity in logical modeling, it should help clarifying the reasoning component in applications. In particular, the introduction of interpretation logics makes the abstract dialogue of theories in different logics in applications fairly concrete. Consider, e.g., the question (\*): What is the relationship between and the consequence of two theories  $T_1$  and  $T_2$  (e.g. an SVBR rule system and an OWL DL ontology)? The answer ( $T_1 = T_2$ ,  $T_1 \subset T_2$ ,  $T_1 \supset T_2$ , or some partial inclusion, resulting inconsistency, etc.) depends upon the properties of the theories and the underlying logics and the interpretation on the level of the meta language(s). However, in applications, the question (\*) typically appears in a form: What is the relationship of two theories  $T_1$  and  $T_2$  *with respect to a particular application*? Since few applications need every aspect of the theories, a useful amalgam might be constructed in terms of a suitable interpretation logic system (e.g. an easily accessible Prolog interpreter), by explicitly escaping the pathologies of the theories, without the need of introducing highly specialized reasoning tools in every application.

#### REFERENCES

- [1] M.E. Aranguren, Ontology design patterns for the formalisation of biological ontologies, Ph.D. Thesis, University of Manchester, faculty of engineering and physical sciences, 2005. Available at <http://www.gong.manchester.ac.uk/docs/MPhilThesis.pdf>
- [2] F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F., and Patel-Schneider, Description Logic Handbook, Cambridge University Press, 2002.
- [3] D. Beckett, Dave Beckett's Resource Description Framework (RDF) Resource Guide, Planet RDF, 2007. Available at <http://planetrdf.com/guide/>
- [4] C.F. Da Silva, L. M'edini, S.A. Ghafour, P. Hoffmann, P., and P. Ghodous, Semantic Interoperability of Heterogeneous Semantic Resources, In Electronic Notes in Theoretical Computer Science 150, 2006, pp. 71-85.
- [5] E. Dantsin, T. Eiter, G. Gottlob, A., and Voronkov, Complexity and Expressive Power of Logic Programming, ACM Computing Surveys, Vol. 33, No.3, September 2001, pp. 374-425.
- [6] E. Gamma, R. Helm, R. Johnson. J., and Vlissides, Design patterns: elements of reusable object-oriented software, Addison-Wesley, USA, 1994.
- [7] S. Hawke (Ed.), Rule Interchange Format Working Group Charter, World Wide Web Consortium, 2007. Available at <http://www.w3.org/2005/rules/wg/charter.html>.
- [8] J. Heflin (Ed.), OWL Web Ontology Language Use Cases and Requirements, W3C Recommendation, 10 February 2004, World Wide Web Consortium, Available at <http://www.w3.org/TR/webont-req/>.
- [9] I. Herman (Ed.), W3C Semantic Web Activity, World Wide Web Consortium, 2007. Available at <http://www.w3.org/2005/rules/wg/charter.html>.
- [10] J. Hintikka, What Do Logic and Computer Science do with Each Other? In Logic, Mathematics and the Computer, Publications of the Finnish Artificial Intelligence Society, No. 14, 1996, pp. 85-88.
- [11] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroel, A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools (Edition 1.0), The University Of Manchester, 2004. Available at <http://www.co-ode.org/resources/tutorials/ProtegeOWLTutorial.pdf>.
- [12] E. Hyvönen, M. Junnila, S. Kettula, E. Mäkelä, S. Saarela, M. Salminen, A. Syreeni, A. Valo, and K. Viljanen, Finnish Museums on the Semantic Web: The user's Perspective on MuseumFinland. Proceedings of the Museums and the Web 2004, Toronto, Canada. Available at <http://www.archimuse.com/mw2004/papers/hyvonen/hyvonen.html>
- [13] H. Knublauch, O. Dameron, and M.A. Musen, Weaving the Biomedical Semantic Web with the Protégé OWL Plugin, Proceedings of the First International Workshop on Formal Biomedical Knowledge Representation (KRMed04), Whistler, Canada, 2004. Available at <http://protege.stanford.edu/plugins/owl/publications/KRMed2004-protege-owl.pdf>.
- [14] F. Manola, E. Miller, and B. McBride (Eds.), RDF Primer, W3C Recommendation, 10 February 2004, World Wide Web Consortium, Available at <http://www.w3.org/TR/2004/REC-rdf-primer-20040210/>.
- [15] G. Newton, J. Pollock, D.L. McGuinness, Submission request to the World Wide Web Consortium: Semantic Web Rule Language (SWRL), World Wide Web Consortium, 2004. Available at <http://www.w3.org/Submission/2004/03/>.
- [16] N. Noy (Ed.), Representing Classes As Property Values on the Semantic Web. W3C Working Group Note 5 April 2005, World Wide Web Consortium, Available at <http://www.w3.org/TR/swbp-classes-as-values/>.
- [17] O. Nykänen, An Approach to Logic Programming with Type-1 Fuzzy Models Using Prolog. Proceedings of the IADIS International Conference of Applied Computing, San Sebastian, Spain, 2006, pp. 201-208.
- [18] O. Nykänen, M. Mannio, J. Huhtamäki, and J. Salonen, A Socio-Technical Framework for Visualising an Open Knowledge Space, In Proceedings of the WWW/Internet 2007, Portugal, to appear.
- [19] OMG. Semantics of Business Vocabulary and Business Rules (SBVR), Second SBVR Interim Specification without change bars dtc/06-08-05. Object Management Group, 2006. Available at <http://www.omg.org/docs/dtc/06-08-05.pdf>
- [20] D. Reynolds, Jena 2 Inference support. Sourceforge documents by the Hewlett-Packard Development Company, 2007. Available at <http://jena.sourceforge.net/inference/index.html>.
- [21] S. Russel, and P. Norvig, Artificial Intelligence; A Modern Approach, Prentice Hall, USA, 1995.
- [22] M.K. Smith, C. Welty, and D.L. McGuinness (Eds.), OWL Web Ontology Language Guide, W3C Recommendation 10 February 2004, World Wide Web Consortium, Available at <http://www.w3.org/TR/owl-guide/>.
- [23] R. Troncy, J. van Ossenbruggen, J.Z. Pan, J.Z., and G. Stamou (Eds.), Image Annotation on the Semantic Web, W3C Incubator Group Report 14 August 2007, World Wide Web Consortium, Available at <http://www.w3.org/2005/Incubator/mmssem/XGR-image-annotation/>.
- [24] V. Vassiliadis, Thea: A Web Ontology Language - OWL Library for [SWI] Prolog, SemanticWeb.gr, Greece, 2007. Available at <http://www.semanticweb.gr/TheaOWLLib/Thea%20OWL%20Lib.pdf>
- [25] J. Wielemaker, G. Schreiber, and B. Wielinga, Prolog-based infrastructure for RDF: performance and scalability. In Proceedings ISWC'03, Florida. USA, 2003.
- [26] J. Wielemaker, An optimised Semantic Web query language implementation in Prolog. Proceedings of the ICLP 2005, Barcelona, Spain.
- [27] J. Wielemaker, G. Schreiber, and B. Wielinga, Using triples for implementation: the Triple20 ontology-manipulation tool. Proceedings of the ISWC2005, Galway, Ireland.