Engineering Ontologies using Semantic Patterns

Steffen Staab, Michael Erdmann
Institute AIFB, University of Karlsruhe
D-76128 Karlsruhe, Germany
www.aifb.uni-karlsruhe.de/WBS/
Ontoprise GmbH, Haid-und-Neu-Strasse 7
D–76131 Karlsruhe, Germany
www.ontoprise.com

Alexander Maedche
FZI Research Center
for Information Technologies
Haid-und-Neu-Strasse 10-14
D–76131 Karlsruhe, Germany
www.fzi.de/wim

Abstract

Interoperability is one of the major design objectives when building applications for B2B and Semantic Web applications. In this paper, we present a methodology for engineering semantic knowledge such that these semantic structures are easier reusable when switching between several representation languages. For this purpose, we reconsider the commonalities of representation languages and their usage in actual applications. Out of this consideration we derive *semantic patterns* as a means to communicate knowledge at an epistemological level of representation and as a means for (partial) execution by any particular implementation of any representation language. The underlying method we propose combines the advantages of formal specification methods (where feasible) with informal, natural language explanations such as used in software engineering for design patterns.

1 Introduction

The Web tremendously changed the way companies do their business, because it provides cheap, easy and widely available transport for information. Now, the Semantic Web is about to let the Web mature from a technical platform that allows for the transportation of syntactic information to the communication of knowledge. The prime format for the latter is RDF (Resource Description Framework) and RDFS (RDF-Schema). RDF [25] was designed by its developers in the Web way, *i.e.* as a smallest common denominator that a lot of people can easily adhere to, only representing a light-weight object model (cf., e.g., [3]) with URIs and reification. *RDFS* [8] adds an additional layer to integrate some simple notions of classes, class inheritance, properties and property inheritance. While RDF(S)[1] certainly goes an important step into the direction of the "Semantic Web", it only provides a very lightweight, and thus extremely restricted, semantic language. Therefore, a number of proposals for languages and language extensions on top of RDF and RDFS are currently under development (cf. [14; 2; 11], which describe some of them). Given the large variety of logics in use in many systems nowadays and given experiences from knowledge representation and reasoning[2] that have shown the necessity of this multitude of languages, the variety of these proposals gives only a first impression of the Babel of languages which will come up in the Semantic Web. This Babel, however, is counterproductive to semantic interoperability which lies at the heart of doing smart B2B on the Semantic Web (e.g., for exchanging knowledge about catalogues or about resource availability). This paper is about engineering machine-processable knowledge in a way such that it is reusable across different Semantic Web languages and across different styles of modeling.

Even before the wide-spread usage of the Web, there have been efforts to find one representation level for all languages (cf., KIF [20; 19]) and to automatically translate between different languages (cf., OntoLingua [22]), but both approaches heavily suffered from the fact that the *meaning* of representations, *i.e.* their semantic entailments, could not be adequately represented in a single *lingua franca*. In order to allow for reuse of semantic information and a multitude of underlying representation languages, we approach the problem from a different angle, an engineering point of view, by considering differences and commonalities of various languages at an explicitly modeled *epistemological level* (cf. [7]). We opt for, first, building on RDF(S) and, second, by constructing *semantic patterns* that capture the intended semantic entailments.

While RDF(S) allows for a frame model that virtu-

[1]We use "RDF(S)" to refer to the combined technologies of RDF and RDF-Schema.

[2]Various applications request different types of languages and reasoning systems, ranging from description logics systems (e.g., for data warehouse quality [17]), over — tractable — non-monotonic reasoning systems (e.g., non-monotonic inheritance for insurance help desk [27]), or systems that include temporal reasoning (e.g., for corporate history analysis [4]).

ally everyone may agree one, what really distinguishes and what is common to any two representation languages are the differences and commonalities of the semantic entailments expressible there. We show in this paper how to model commonalities in, what we call, *semantic patterns*. Semantic patterns are used for communication between Semantic Web developers on the one hand, but also for mapping and reuse to different target languages on the other hand, thus bridging between different representations and different ways of modeling knowledge. Developing the semantic patterns, we do not invent the wheel from scratch, but we pick insights from software engineering and knowledge representation research and integrate them for use in the Semantic Web.

By sheer principle, we cannot produce an exhaustive list of possible semantic patterns or show how the epistemological level should look like given any set of representation languages. Hence, we substantiate the claims we make with a case study considering as target representation systems OIL/FaCT [14], currently the most prominent semantic layer on top of RDF(S), and SiLRi [13], an F-Logic-based [24] representation system.

Outline of the paper. In the following, we start with our model for semantic patterns, their underlying rationale as well as their formal and informal components (Section 2). Then, we show how this model fits into the Semantic Web (Section 3), *i.e.* how it can be realized in RDF and from which point it has to evolve now. Subsequently, we illustrate our approach with a case study. We describe an application scenario, where semantics are brought to bear in a target representation independent way.

2 Semantic Patterns

The rationale and conceptual model of our approach is explained in this section. In order to give the broad view necessary to understand the problem of modeling knowledge for a variety of representation languages through semantic patterns, this section

1. analyses the abstract properties of our problem, thereby recollecting the most relevant related work in this area;
2. characterizes the high-level solution to the problem, which leads to informal means for communicating a semantic pattern together with formal descriptions of consistency constraints; and, finally,
3. defines these two aspects of semantic patterns exemplifying them by one extremely simple and useful pattern, for which no modeling primitive exists in most modeling languages and frameworks.

2.1 The Problem and some of its History

When one tries to reuse semantics across boundaries stemming from the usage of different representation languages, different actual representation tasks and their correspondingly different formal models, one may recognize characteristics of the semantic models that remain constant. Striving for semantic interchangability the crucial point lies in capturing these characteristics. This is difficult, because:

- Different language definitions come with different formal models. In general, the models of two different languages are not comparable at all. Thus, when one defines a translation there may not exist a criterion to evaluate the correctness of the translation.
- There may be several semantically equivalent statements in one language. Their equivalence is, in general (e.g., for first-order predicate logic), undecidable. Hence, their fully automatic translation is, in general, not feasible.
- Some choices for representation are not semantically motivated, but are made in order to generate some particular behaviour of the actual system.

Therefore, direct translations from one representation language into the next do not seem to yield a viable way. As a way around this dilemma, we consider the engineering task of constructing a particular representation. Rather than working hard to implement intended semantic entailments in statements of one particular — for the purpose of reuse and translation even arbitrary, language — the engineer may decide to explicitly model semantic entailments at a meta-level, instantiate the meta-level description, and compile the final representation into the one or the other target language.

In fact, those semantic entailments that are most widely agreed upon, such as necessary inheritance conditions, directly show up in common representation languages, such as `rdfs:subclass` and `rdf:type` in RDF(S). This also is the reason that subsequently we may easily assume that ground RDF facts are translatable into virtually all target languages.

Then, there is another medium size set of such semantic characteristics that are widely deemed interesting, that can be modeled independently from particular target languages in a number of systems and that can also be mapped into a wide range of languages. They are widely known from object-oriented databases. Gruber defined a set of primitives that captures them in his *Frame Ontology* [22] for use of comparatively simple translation between several representation languages (e.g., transitivity of relations, database joins, or disjointness of concepts).[3]

[3]Subsets of them have also been discussed/are under discussion for usage in description logics languages like OIL.

For characteristics more sophisticated than those mentioned above, there exists no comprehensive concept for engineering semantics in a way that is really reusable across several languages.

By its very nature, the problem of describing formal model characteristics for all representation languages is an open one that cannot be solved by producing a closed list of modeling primitives like the ones in Gruber's Frame Ontology. Hence, there is a need for a technique of describing new semantic primitives at a higher level of abstraction.

Again looking back into history, Brachman [7] and others have captured particular model characteristics, i.e. *semantic entailments*, in *axiom schemata* for the purpose of easier engineering of large sets of axioms. Axiom schemata provide an abstraction of actual axioms and particular axiom schemata were categorized and named. Doing so, Brachman introduced the name *epistemological level* for this layer of description. The results of his efforts were a set of epistemological primitives for description logics. Unlike our purpose, his goal was not the reuse of semantics across represenation languages, but rather the reuse of engineering efforts in one language.

2.2 The High-level Solution to the Problem

While axiom schemata already go into the direction of abstracting from formal model characteristics, by definition they are developed for one language only. Hence, one part of our high-level idea was to allow for (an open list of) **new epistemological primitives** that can be instantiated in different representation languages for modeling particular semantic entailments and which are, thus, similar to named axiom schemata working in one language.

However, one needs a more flexible paradigm better suited to apply to a larger range of representation languages and more able to abstract from particular formal models. As described above, the general problem does not allow to come up with a completely formal and ubiquitously translatable specification of semantics. Hence, the other part of our high-level idea is to require extra efforts from Semantic Web developers. To support them in their efforts, it appeared to be a prerequisite that they could communicate more efficiently about these new epistemological primitives — similar to the way that software engineers talk about recurring software designs.

Design Patterns and Semantic Patterns. Design patterns have been conceived for object-oriented software development to provide *(i)* a common design vocabulary, *(ii)* a documentation and learning aid, and *(iii)* support for reorganizing software. Likewise to the naming and cataloguing of algorithms and data structures by computer scientists, design patterns are used by software engineers to communicate, document and explore design alternatives by using a common design vocabulary or a design pattern catalog. By this way, they also decrease the complexity of developing and understanding of software systems. Additionally, design patterns offer solutions to common problems, help a novice "acting" more like an expert and facilitate the reverse-engineering of existing systems.

Though bridging between formal representations seems to be a formal task only, very often quite the contrary becomes true. When not everything, but only relevant aspects of knowledge can or need to be captured, when not all inferences, but only certain strains of semantic entailments can or need to be transferred, the development of new semantic primitives should not only allude to the formal definition of translations into target languages, but also to informal explanations. Therefore a semantic pattern does not only comprise new epistemological primitives, but likewise to design patterns, it also serves as a means for communication, cataloguing, reverse-engineering, and problem-solving. Thus, it may contribute to a more efficient exploitation of Semantic Web techniques.

Semantic Patterns and Consistency. Experiences in the related field of problem solving methods (cf. Section 6) have shown that there are as many interpretations of natural language descriptions as there are readers [15]. Given the preliminary that we do not want to subscribe to any particular, extremely powerful, and hence undecidable, specification language, we nevertheless need some means to describe consistency conditions that an implementation of a semantic pattern must adhere to.

The basic idea here is the following: A semantic pattern enforces semantic entailments on ground facts. A semantic pattern is implemented by translating its instantiated epistemological primitives into the target language. Thus, if one gives an instantiation of a semantic pattern together with some example ground facts related to the pattern, the implementation (i.e., the translation together with the target system) may derive semantic consequences. A translation may be considered consistent, if it derives those consequences out of the example ground facts that the developers of the semantic patterns wanted to be derived (i.e. the positive examples) and not those that they explicitly excluded (i.e. the negative examples).

This definition of *consistency of translations* is easy to realize, since it only builds on premises that are already given within the semantic patterns framework sketched so far. In particular, the translation of ground RDF facts into the target language is sometimes trivially done by an identity function (e.g., for OIL or DAML-Ont). Otherwise it is not overly complicated, because the RDF model already is a kind of least common denominator for the representation languages we consider.[4] The reader may note that this notion of con-

[4]The only counterexamples we could come up with were

sistency may not completely prevent misuse or misunderstanding. For instance, translations that map everything to the empty set always incur consistency without doing any good.

A complete description of semantic patterns including a formal specification of consistency will be given in the following.

2.3 Two Complementary Aspects of Semantic Patterns

This subsection puts the high-level rationale outlined above into a concrete perspective. We start with the informal description of the template structure of semantic patterns. Subsequently, we specify the formal parts of semantic patterns including consistency conditions for the translation into target representations.

Informal Description of Semantic Patterns. A Semantic Pattern consists of two major parts. The first part describes core elements that are completely independent from any actual implementation. The second part specifies example implementations, including descriptions about target system/language-specific behaviour.

The first part consists of the following eight core elements (also cf. Example Pattern part 1 given in Table 1):

1. **Pattern Name**: describes in few words the semantic problem, its solutions and consequences. Naming extends the pattern catalog and extends the semantic pattern vocabulary.
2. **Intent**: is a short statement describing what the pattern does and what its rationale and intent are.
3. **Also Known as**: enumerates other well-known names of this patterns (synonym list).
4. **Motivation**: describes a scenario that illustrates the semantic problem and elucidates how the semantic pattern may help in making implicit knowledge explicit.
5. **Structure**: represents the pattern. In particular, it gives the defining namespace, lists the relevant (new) epistemic primitive(s) and describes their signature.
6. **Known Uses**: shows examples of the pattern found in real Semantic Web applications.
7. **Related Patterns**: lists a number of closely related patterns (e.g. generalization hierarchy of patterns) and describes how they are related
8. **Constraints**: lists tuples of RDF facts and instantiated epistemic primitives ($C_{i,\text{in}}$, $C_{i,\text{out}}$, $C^{opt}_{i,\text{out}}$, $C_{i,\text{notout}}$, $C^{opt}_{i,\text{notout}}$). Their intended meaning is that for all i, given $C_{i,\text{in}}$, and thus using the new epistemic primitives of the pattern, $C_{i,\text{out}}$ *must* be derived in any given implementation and $C_{i,\text{notout}}$ *must not* be derived in any given implementation. In addition, one may include sets $C^{opt}_{i,\text{out}}$ and $C^{opt}_{i,\text{notout}}$ that, correspondingly, *should* and *should not* be derived in any given implementation.

rather esoteric schemes, like monadic predicate logics.

The second part deals with implementation aspects of a Semantic Pattern. It consists of an arbitrary number of descriptions that relate the semantic pattern to particular target languages/systems. Each singleton entry (referring to one target language/system) should include the following five template elements (also cf. Example Pattern part 2 described in Table 2):

1. **Name of target language/system**: refers to the actual language specification. Because various system implementations may even incur different behavior (ranging from response time to various degrees of covering a specification), we also allow to specify system implementations rather than just language versions.
2. **Applicability**: The applicability of a semantic pattern in an actual language/system may be restricted. Example restrictions may necessitate the generation of new symbols in a particular target system or they may restrict the semantic entailments generated from $C_{i,\text{in}}$ to some subset of $C_{i,\text{out}}$ to mention but two example restrictions.
3. **Translation result of input constraints** $C_{i,\text{in}}$: shows the representation of an example fact base in the target language. Thus, the user of the Semantic Pattern sees an explicit example result of the translation process.
4. **Translation — Sample Specification**: This specification describes the translation of instantiated epistemic primitives of the given pattern into the target language. The specification may be given in logics, pseudo code or a real programming language. In some target languages (e.g., F-Logic) it is reasonable to specify the translation in the target language itself. If the translation is given by a formal specification it can be considered to represent the translation T_i, which is referred to in the following.
5. **Comment**: Additional comments on particularities of the translation and the translation results.

The reader may note in the example templates that the various fields of the semantic patterns are not required. For ease of presentation, we have used italics in the running example to abbreviate redundant syntactic descriptions.

[5]For better readability we here mostly use a PL1-style of denotation (without quantifiers) that can be easily mapped to RDF.

Table 1: Example Semantic Pattern — Part 1

Semantic Pattern	
Pattern Name	`Locally Inverse Relation`
Intent	Allows to define inverses the scope of which is restricted to particular concepts.
Also Known As	Restricted inverse relation
Motivation	Often the definition of global inverses is too generic and yields overly general inferences. For example, one may have ontology definitions that every MOVIE ISSHOWN in a THEATRE and every PLAY ISGIVEN in a THEATRE and THEATRE HOST EVENT. Now, the local inverse of ISSHOWN is HOST restricted to the range MOVIE and the local inverse of ISGIVEN is HOST restricted to the range PLAY. A global inverse might lead to unwanted consequences. For this reason this pattern allows the definition of inverse properties restricting their domain and range concepts. This notion of locality is naturally given in OO systems, where properties are defined locally in classes. It is not given in RDF(S) where properties are first-class citizens and exist independent of classes.
Structure	
Namespace	`http://ontobroker.aifb.uni-karlsruhe.de/schema/LocalInverse.rdf`
Epistemic Primitive(s)	LOCALINVERSE
Signature	LOCALINVERSE(r_1, c_1, r_2, c_2) with r_1, r_2 denoting binary relations and c_1, c_2 denoting their corresponding ranges
Known Uses	`http://ka2portal.aifb.uni-karlsruhe.de`
Related Patterns	The pattern "globally inverse relation" subsumes "locally inverse relation" when applied to the same relations
Constraints[5]	
$C_{1,\text{in}}$	LOCALINVERSE(ISSHOWN, THEATRE, HOST, MOVIE), DOMAIN(ISSHOWN, MOVIE), RANGE(ISSHOWN, THEATRE), DOMAIN(HOST, THEATRE), RANGE(HOST, EVENT), TYPE(Lassie, MOVIE), HOST(Schauburg, Lassie)
$C_{1,\text{out}}$	ISSHOWN(Lassie, Schauburg)
$C^{\text{opt}}_{1,\text{out}}$	TYPE(Schauburg, THEATRE)
$C_{1,\text{notout}}$	ISGIVEN(Lassie, Schauburg), TYPE(Lassie, PLAY)

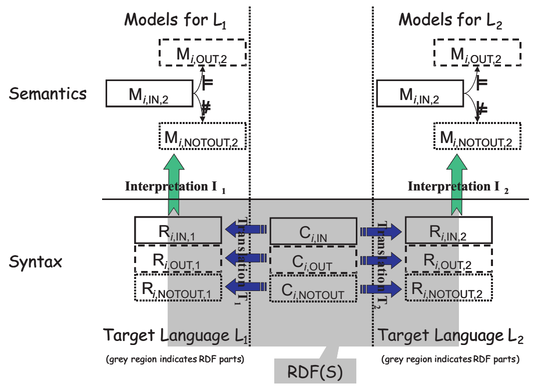


Figure 1: Checking for consistency

Formal Consistency Conditions for Semantic Patterns. Combining our considerations on consistency with our actual specification of semantic patterns we may now describe the overall setting in formal terms (also cf. Figure 1).

We base our consistency checking on facts in RDF (cf. $C_{1,\text{in}}, C_{1,\text{out}}, C_{1,\text{notout}}$ in Figure 1), which may include some of the new epistemic primitives. Each translation maps this RDF based representation into a target language (or system) L_j, resulting in target representations $R_{1,\text{in},j}, R_{1,\text{out},j}, R_{1,\text{notout},j}$ ($j = 1, 2$). From any consistent translation T_j the interpretation of output facts $R_{1,\text{out},j}$ must and the interpretations of facts $R_{1,\text{notout},j}$ must not be semantically entailed by the corresponding interpretation of input constraints $R_{1,\text{in},j}$. Depending on the actual system, semantic entailment ($\models$) or not-entailment ($\not\models$) may be replaced by syntactic derivation ($\vdash$) or not-derivation ($\nvdash$).

To describe this intuition precisely, we specify:

Definition 1 (Translation Mapping) *A translation mapping is any function $T_i : 2^{\text{sentences(RDF)}} \rightarrow 2^{\text{sentences}(L_i)}$, where sentences($X$) stands for all legal sentences of language X, L_i ($i = 1 \ldots n$) are representation languages, and $2^{\text{sentences}(X)}$ describes the set of all subsets of all legal statements of language X.*

This simply boils down to: a translation mapping for target language L_i is able to translate every possible RDF representation into this target language.

In this definition we assume that new epistemological primitives are defined by statements in RDF (also cf. Section 3). Given such a translation mapping, we

Table 2: Example Semantic Pattern — Part 2

Semantic Pattern Implementations for Locally Inverse Relations	
Language/System	OIL/FaCT
Applicability	Requires the creation of artificial relations for this type of modeling.
Translation (Sample Code)	copy RDF literally, create two new subproperties with specialized range restrictions and declare appropriate INVERSE relation
Translation Result for $C_{1,\text{in}}$	*literal copy of the statements in $C_{1,in}$ plus RDF equivalent of ...*
	slot-def *host1*
	subslot-of *host*
	inverse *isShown*
	slot-def *host2*
	subslot-of *host*
	inverse *isGiven*
Comment	The reader may note that in contrast to `rdfs:subPropertyOf` OIL's **subslot-of** allows for cycles.
Language/System	F-Logic/SiLRi
Applicability	Applicable.
Translation (Sample Code)	translate RDF syntactically and add two meta-rules (see below)
Translation Result for $C_{1,\text{in}}$	*Syntactic translation of statements in $C_{1,in}$ plus ...*
	$FORALL\ C1, C2, R1, R2, O1, O2$
	$O2[R2 \twoheadrightarrow O1]\ and\ O1 : C1 \leftarrow$
	$\textsc{localInverse}(R1, C1, R2, C2)\ and\ O1[R1 \twoheadrightarrow O2]\ and\ O2 : C2.$
	$FORALL\ C1, C2, R1, R2, O1, O2$
	$O1[R1 \twoheadrightarrow O2]\ and\ O2 : C2 \leftarrow$
	$\textsc{localInverse}(R1, C1, R2, C2)\ and\ O2[R2 \twoheadrightarrow O1]\ and\ O1 : C1.$
Language/System	Predicate Logic
Applicability	Applicable.
Translation (Sample Code)	*add the following PL2 Specification*
	$FORALL\ C1, C2, R1, R2, O1, O2$
	$R2(O2, O1) \wedge \textsc{type}(O1, C1) \leftarrow$
	$\textsc{localInverse}(R1, C1, R2, C2) \wedge \textsc{type}(O2, C2) \wedge R1(O1, O2)$
	$FORALL\ C1, C2, R1, R2, O1, O2$
	$R1(O1, O2) \wedge \textsc{type}(O2, C2) \leftarrow$
	$\textsc{localInverse}(R1, C1, R2, C2) \wedge \textsc{type}(O1, C1) \wedge R2(O2, O1)$

can evaluate to which degree it is consistent with regard to the constraint specification of the given semantic pattern.

Definition 2 *Let the semantic pattern S include the constraints ($C_{i,in}$, $C_{i,out}$, $C^{opt}_{i,out}$, $C_{i,notout}$, $C^{opt}_{i,notout}$) for $i := 1 \ldots m$. A translation mapping T_j is called consistent with the Semantic Pattern S iff forall $i := 1 \ldots m : T_j(C_{i,in}) \models T_j(C_{i,out})$ and $T_j(C_{i,in}) \not\models T_j(C_{i,notout})$.*

Definition 3 *Let the semantic pattern S include the constraints ($C_{i,in}$, $C_{i,out}$, $C^{opt}_{i,out}$, $C_{i,notout}$, $C^{opt}_{i,notout}$) for $i := 1 \ldots m$. A translation mapping T_j is called strongly consistent with the Semantic Pattern S iff T_j is consistent with S and forall $i := 1 \ldots m : T_j(C_{i,in}) \models T_j(C^{opt}_{i,out})$ and $T_j(C_{i,in}) \not\models T_j(C^{opt}_{i,notout})$.*

3 Semantic Patterns for the Semantic Web

Building on the rational and methodology outlined above, the basic idea of semantic patterns on the Web has two major dimensions: First, there is the dimension of technical representation and, second, there is the social process of establishing Semantic Pattern libraries on the Web.

3.1 Representing Semantic Patterns in RDF

Semantic patterns are used for communicating some information to human developers and some information to computer systems. Hence, RDF is also the ideal format for the representation of the Semantic Pattern itself.

We have provided a RDF-Schema description for semantic patterns available at `http://ontoserver.aifb.uni-karlsruhe.de/schema/PatternSchema.rdf` which describes RDF resources of `rdf:type` *Pattern*.

We refer to this schema with the namespace prefix `ps` for Pattern Schema. An actual instantiation of this schema, *viz.* our running example, `locally inverse relation`, is shown at
`http://ontoserver.aifb.uni-karlsruhe.de/schema/LocalInverse.rdf`.
For easier presentation we refer to it in the following outline of these RDF structures by the namespace prefix `pa` for Pattern Application.

A semantic pattern is a `rdfs:Resource` and can be associated with other resources by a number of defined properties. The properties `ps:patternName`, `ps:intent`, `ps:alsoKnownAs`, and `ps:motivation` have been described in Section 2 and associate a pattern object with literal, textual values (`rdf:parseType="Literal"`). The property `ps:relatedPattern` links a pattern to related ones. The structure of a pattern is modeled by the two properties `ps:epistemicPrimitive` and `ps:signature`. The former represents a simple literal while the latter associates a pattern with several named properties (e.g. `pa:class1` or `pa:rel2`). These properties define the parameters of the pattern and must define the pattern itself as their `rdf:domain`. The ranges of the parameter properties represent the parameter types for the pattern. The example pattern has four parameters: `pa:class1` and `pa:class2` of type `rdfs:Class`, and `pa:rel1` and `pa:rel2` of type `rdf:Property`.

All mentioned information becomes a part of the actual pattern description, i.e. the RDF model. Applications can ask for the signature of a pattern by querying this model, esp. the `ps:signature` and `ps:epistemicPrimitive` properties of the pattern. This formal part of the model can directly be exploited for further processing, e.g. for building GUIs for instantiating a particular semantic pattern, e.g. for instantiating the locally inverse relations-pattern with HOST, MOVIE, ISSHOWN, and THEATRE.

The constraints ($C_{i,\text{in}}$, $C_{i,\text{out}}$ etc.) used for checking consistency of actual implementations represent partial models that are only true within their consistency checking context, but not on a global scale. The means of RDF for representing contextual information is *reification*. Therefore the different constraint sets are modeled via reification. Each set of constraint statements is retrievable from a pattern-resource by querying one of the constraint-properties. Each such property relates pattern-resources with `rdf:Statements`. Translation functions (cf. Definition 1) may access these sets and translate the reified statements into the target language.

The second part of semantic patterns as described in Section 2 defines target language/system-specific information about the pattern, of possible implementations, and expected results of translation functions. The description of the name of the target language and system are modeled as literal values of the properties `ps:language` and `ps:system`, respectively. The translation code may be stored within another RDF-literal accessible via the `ps:code` property of the `ps:Implementation` resource. Results of applying this code to the sample given in the constraint set $C_{i,\text{in}}$ can be represented in the RDF-model of the implementation as well. Since languages exist that directly operate on the RDF-model, it is possible to store reified RDF-statements reachable via the `ps:C_in_rdf`-property. Applications that do not understand RDF syntax may retrieve the transformation of the statements $C_{i,\text{in}}$ from `ps:C_in_literal`.

It is our general policy to allow developers a lot of leeway. Currently, all mentioned properties are optional and in the typical case either `ps:C_in_rdf` or `ps:C_in_literal` is given but not both.

The reader may note that the formal description (in RDF) of formal parts allows for direct digestion of constraints and signatures for aims such as code generation, consistency checking, and user interface construction.

3.2 Semantic Pattern Libraries

Eventually, the need for particular semantic patterns is driven by Semantic Web developers. With the engineering of ontologies on the Web (cf., e.g., [1]) new ideas will come up about what type of inferencing shall be supported and, hence, made interchangable between representation systems.

Since this development is in its infancy right now, we have started to collect a number of semantic patterns that seem widely applicable:

- Gruber's Frame Ontology includes a set of over 60 primitives, some of which are found in core RDF(S), e.g. `rdf:type`, and some of which are somewhat more sophisticated, e.g. symmetry of relations or composition (database joins).
- Medical knowledge processing often relies on the engineering of *part-whole reasoning* schemes such as appear or do not appear when we consider the following examples: *(i)*, the appendix is part of the intestine. Therefore, an appendix perforation is an intestinal perforation. And, *(ii)*, the appendix is part of the intestine, **but** an inflammation of the appendix (appendicitis) is not an inflammation of the intestine (enteritis).

  We have described how to represent structures that allow for expressing (for *(i)*) and preventing (for *(ii)*) these semantic entailments in RDF in [29] — in a preliminary version of the semantic patterns framework.
- Inheritance with exception is a semantic pattern that is very often useful. Its application and its tractable, even efficient, technical reasoning part has been described, e.g., in [27]. The core idea is that one considers the inheritance of properties, allows for the non-inheritance

of certain properties, and uses a particular, unambiguous strategy for resolving conflicts between paths of inheriting and non-inheriting a particular property. A simple example is that a PATIENT's treatment may be covered by medical insurance, a NON-COMPLIANT PATIENT's treatment may not be covered, but a NON-COMPLIANT, MENTALLY DISTURBED PATIENT's treatment will be paid by the insurance company. Hence, coverage of treatment is typically inherited, e.g. by almost all subclasses of patient, but not by ones like NON-COMPLIANT PATIENTs.

Note that often there is no translation into particular target languages for this pattern. For instance, it can be realized in Prolog or F-Logic, but not in the standard description logics systems.

- A number of patterns may be derived from object-oriented or description logics systems, e.g. *local range restrictions* are very often useful. A simple example is that the *parentOf* a HUMAN is restricted to HUMAN, the *parentOf* a FOX is restricted to FOX, while the range restriction of *parentOf* may be ANIMAL in general.

A more complete elaboration of these and other patterns is currently under development. In particular, we investigate how software engineering methodology about modeling and code generation from an evolving library of semantic patterns can be brought to bear within our modeling environment (cf. Section 5).

4 Using Semantic Patterns — A Case Study

In [28] we have described how "Semantic Community Web Portals" using ontologies can be built. The ontology acts as a semantic backbone for accessing information on the portal, for contributing information, as well as for developing and maintaining the portal. We discussed a comprehensive and flexible strategy for building and maintaining a high-value community web portal, where the development and maintenance process consists of the stages *requirements elicitation*, *web site design*, *ontology engineering* and *query formulation*. The reasoning service for the portal was provided by SiLRI [13], which is essentially based on F-Logic [24; 12], only ground facts may alternatively be provided in RDF syntax. F-Logic fits nicely with the structures proposed for RDF and RDFS, however, F-Logic does not offer any support for interoperability of representation mechanisms, *i.e.* axioms written in F-Logic and the implicit knowledge that comes from applying them to the fact base are extremely hard to reuse in other representation frameworks. In this section we show how our approach fits with a recent proposal for representing knowledge on the web, namely OIL, the ontology inference layer [14]. OIL, which is in several semantic respects "orthogonal" to F-Logic, offers inferencing [23] on a semantic layer on top of RDF(S).

In the following case study we show the usage of semantic patterns for meeting the needs of an actual application, while allowing for the engineering of semantics on a level that is transportable to OIL *and* F-Logic (and many other representation schemes). The case study described here relies on the tools and techniques we employed for building "Semantic Community Web Portals". We here consider a **Cultural Event Portal**, that integrates distributed information from movie databases and cinema programs and offers semantic access to the information provided.

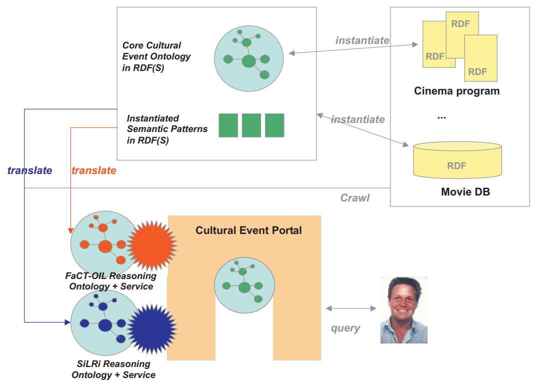

Core Cultural Event Ontology in RDF(S)
instantiate
RDF
RDF
RDF
Cinema program
...
Instantiated Semantic Patterns in RDF(S)
instantiate
RDF
Movie DB
translate
translate
Crawl
Cultural Event Portal
FaCT-OIL Reasoning Ontology + Service
query
SiLRi Reasoning Ontology + Service

Figure 2: Case Study — Building a Semantic Cultural Event Portal

Figure 2 depicts the overall framework. Based on the core ontology, actual facts are generated at the information provider side. For building a semantic cultural event portal with sophisticated reasoning we additionally instantiate semantic patterns on top of the core ontology. The core ontology with instantiated semantic patterns may be translated into OIL and/or F-Logic. Facts are crawled from the information providers and given to the reasoning services. The portal accesses the underlying reasoning services and provides comprehensive information on cultural events.

In the following we give some examples how the core ontology looks like, show how actual semantic patterns are defined and translated into OIL and/or F-Logic.

4.1 Modeling the Core Ontology

We use our Ontology Engineering Environment OntoEdit (cf. Section 5 and Figure 3) for engineering class and property definitions in RDF(S) with graphical means. Parts of the core ontology are given as follows:

```
<rdfs:Class rdf:ID="Event"/>
<rdfs:Class rdf:ID="Movie">
 <rdfs:subClassOf rdf:resource="Event"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Play">
 <rdfs:subClassOf rdf:resource="Event"/>
</rdfs:Class>
 <rdfs:Class rdf:ID="Theatre"/>
<rdf:Property rdf:ID="host">
 <rdfs:domain rdf:resource="Theatre"/>
 <rdfs:range rdf:resource="Events"/>
</rdf:Property>
<rdf:Property rdf:ID="isShown"/>
 <rdfs:domain rdf:resource="Movie"/>
 <rdfs:range rdf:resource="Theatre"/>
</rdf:Property>
<rdf:Property rdf:ID="isGiven"/>
 <rdfs:domain rdf:resource="Play"/>
 <rdfs:range rdf:resource="Theatre"/>
</rdf:Property>
```

The ontology defines the conceptual backbone for generating RDF metadata on the information provider side, as for example given through the following statements:

```
<cultev:Movie rdf:ID="movie:Lassie">
 <cultev:name>Lassie</cultev:name>
 <cultev:hasActor
   rdf:resource="actor:RoddyMcDowall"/>
</cultev:Movie>
```

4.2 Generating an OIL ontology with Semantic Patterns

An OIL ontology is built on top of the core RDF(S) data model and contains descriptions of classes, slots and individuals [14]. Classes are unary predicates and may be related to other classes by stating that one is a subclass of another. Slots are binary relations, they may also be related to each other via the notion of subslots.

In our example application, Cultural Event Portal, additional reasoning on top of core RDF(S) is required. We therefore instantiate some patterns on top of the core RDF(S) ontology to enforce semantic constraints and then translate them into the more powerful OIL. In our scenario we use two patterns, namely the *local range restriction* and the *locally inverse relations* pattern. The pattern *local range restriction* (cf. Section 3.2) adds to the class definition of MOVIE with respect to the property ISSHOWN the range restriction THEATRE. The following statements are added within the concept definition of MOVIE:

```
<oil:hasSlotConstraint>
 <oil:ValueType>
  <oil:hasProperty rdf:resource="isShown"/>
  <oil:hasClass rdf:resource="Theatre"/>
 </oil:ValueType>
</oil:hasSlotConstraint>
```

In Section 2 our mechanism for defining semantic patterns has been introduced using the example of *locally inverse relations* patterns. OIL offers the definition of global inverses, that is often too generic and yields overly general inferences. In our example, we defined in our ontology that every MOVIE ISSHOWN in a THEATRE and every PLAY ISGIVEN in a THEATRE and THEATRE HOST EVENT. Now, the local inverse of ISSHOWN is HOST restricted to the range MOVIE and the local inverse of ISGIVEN is HOST restricted to the range PLAY.

As OIL does not directly support locally inverse relations, the creation of artificial relations is required. The translation into OIL is given through the following statements:

```
<rdf:Property rdf:ID="host1">
 <oil:subSlotOf rdf:ID="host"/>
 <rdfs:domain rdf:resource="Theatre"/>
 <rdfs:range rdf:resource="Movie"/>
</rdf:Property> <rdf:Property rdf:ID="host2">
 <oil:subSlotOf rdf:ID="host"/>
 <rdfs:domain rdf:resource="Theatre"/>
 <rdfs:range rdf:resource="Play"/>
</rdf:Property>
<rdf:Property rdf:ID="isShown">
 <oil:inverseRelationOf rdf:resource="host1"/>
</rdf:Property>
```

We introduce two new properties HOST1 and HOST2 as subslots of HOST [6]. The range of property HOST1 is restricted to the MOVIE class, the range of property HOST2 is restricted to the PLAY class. Additionally we use the `inverseRelationOf` construct of OIL to denote that the property ISSHOWN is inverse to the property HOST1.

We also give the translation of `locally inverse relation` pattern to Frame-Logic. The pattern is applicable and is generated via the F-Logic statements we have seen before:

$\textsc{LocalInverse}(\textsc{isShown}, \textsc{Movie}, \textsc{HOST}, \textsc{Theatre}).$

$FORALL\ C1, C2, R1, R2, O1, O2 \quad O2[R2 \twoheadrightarrow O1]\ and\ O1 : C1 \leftarrow$
$\quad \textsc{localInverse}(R1, C1, R2, C2)\ and\ O1[R1 \twoheadrightarrow O2]\ and\ O2 : C2.$

$FORALL\ C1, C2, R1, R2, O1, O2 \quad O1[R1 \twoheadrightarrow O2]\ and\ O2 : C2 \leftarrow$
$\quad \textsc{localInverse}(R1, C1, R2, C2)\ and\ O2[R2 \twoheadrightarrow O1]\ and\ O1 : C1.$

Essentially, this version was used for the Community Web Portal, but it could not be communicated to outsiders of F-Logic.

5 OntoEdit

Our general approach for engineering ontologies in conjunction with developing and using semantic patterns has been or is currently being implemented in ONTOEDIT [30], an ontology engineering workbench for building web ontologies[7]. In this section we give an outline of how an ontology engineering environment is augmented by components for realizing semantic patterns.

The modeling of the core ontologies builds on RDF(S) primitives. The process is started by collecting terms for classes and organizing them hierarchically; in

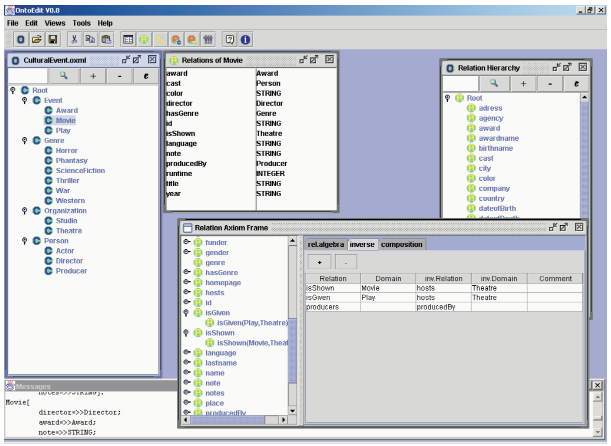
[6]We use the oil:subSlotOf component as defined in the denotational semantics of standard OIL available at http://www.cs.man.ac.uk/ horrocks/OIL/Semantics/oil-standard.html.

[7]More detailed information can be obtained at http://ontoserver.aifb.uni-karlsruhe.de/ontoedit.

parallel one may add properties to the ontology. Several different views for building the ontology are offered to the user. Figure 3 depicts the graphical user interface of ONTOEDIT: On the left hand side of Figure 3 the class hierarchy of our cultural event ontology is depicted. The class-property view offers the user the possibility to attach properties to classes. Properties may also be defined globally and organized hierarchically.

ONTOEDIT offers a number of predefined semantic patterns. On the lower right part of figure 3 the interface for instantiating global inverseness and locally restricted inverseness is depicted. The user selects properties and defines their (local) inverses explicitly. If the user also restricts domain and range of the properties the semantic pattern `locally inverse relations` is instantiated. The text descriptions of the semantic patterns are available in ONTOEDIT's help.

Once conceptual modeling is completed, one may use ONTOEDIT to explore the defined ontology including the newly instantiated semantic patterns. For this purpose, one may crawl example RDF facts, translate the semantic patterns into F-Logic or OIL and then explore ontology and facts by querying the test examples.



award	Award
cast	Person
color	STRING
director	Director
hasGenre	Genre
id	STRING
isShown	Theatre
language	STRING
note	STRING
producedBy	Producer
runtime	INTEGER
title	STRING
year	STRING

Relation	Domain	inv.Relation	inv.Domain	Comment
isShown	Movie	hosts	Theatre	
isGiven	Play	hosts	Theatre	
producers		producedBy		

```
Movie[
        director=>>Director;
        award=>>Award;
        note=>>STRING;
```

Figure 3: Snapshot of OntoEdit Web Ontology Workbench

6 Related Work

This paper is motivated by the need to share and exchange semantic knowledge on the Web (cf., e.g., [9] for general motivation or [28] for an actual application). This need comprises the integration of various sources on the content level as well as on the representation level, i.e. integrating knowledge from various basic representation mechanisms available (like [8]) or on the rise (like [14; 2]).

We started out from the area of *ontology engineering* aiming at conceptual models that could be used in multiple underlying representations (cf. [29]). Doing so, we extended related work in the field of knowledge representation using *axiom schemata*.Because our goal was not only to formally represent, but to allow for rich communication between developers who create actual implementation based on various representation systems, we looked into software and knowledge engineering dealing with *design and knowledge patterns and problem-solving methods*.

6.1 Ontology Engineering & RDFS

In our earlier proposals [30] we have discussed how to push the engineering of ontological axioms from the *symbol level* onto the *knowledge level* — following and extending the general arguments made for ODE [6] and Ontolingua [16]. Also similar to our RDF(S)-based ontology engineering tool ONTOEDIT is Protégé [21], which provides comprehensive support for editing RDFS and RDF, but lacks any support for axiom modeling and inferencing. In contrast to all of these approaches, we aim also at *partial* descriptions of semantic entailments such as very often necessary when switching from one to the other representation paradigm.

6.2 Axiom Schemata

The usage of axiom schemata in various paradigms has been a major motivation for our approach (cf. Subsection 2.3). In particular, we have relied on experiences with engineering axiom schemata in F-Logic and on related work that exploits axiom schemata in various description logics dialects.

F-Logic. The logical model of F-Logic, essentially a rich model for datalog, fits nicely with the structures proposed for RDF and RDFS. This also led to the first implementation of an inference engine for RDF (SiLRi [13]). SiLRi provides many inferencing possibilities one wants to have in RDF and, hence, has provided an excellent start for many RDF applications. In fact, it even allows to use axioms in restricted second-order logic, but these axioms may not be denoted in RDF, but only directly in F-Logic.

Description Logics. Description logics has been derived from efforts for specifying the axiom schemata that are most relevant for terminological engineering. Hence, its development provides valuable input for relevant semantic patterns such as the ones exploited in our case study (cf. Section 4). To speak more precisely, description logics constitutes not a single, but a set of similar languages. A large amount of research has been undertaken to explore the effects of adding additional syntactic and semantic features to existing versions of description logics. However, their efforts remain very far from bridging between mutually incompatible representation paradigms, which is the goal of our approach.

A web-compatible version of description logics has been presented with OIL [14]. OIL is intended as a common core language that is more powerful than RDF, but is intended to provide a basic layer rather than a language "all singing all dancing". As our case study also illustrates, even OIL does not suffice for all potential needs, but semantic patterns may also be used on top of OIL — rather than "only" on top of RDF.

Combinations of Different Logics. Obviously, there has been the need for interoperability between F-Logic and Description Logics and, hence, Levy and Rousset [26] proposed an integration of a (simple) Description Logics approach with horn rules. In the end, however, neither one of them nor their integration will be sufficient for all possible purposes and applications of the future Semantic Web. A similar statement holds for current combinations of modal logics; in fact, the field as a whole is very young and can be exploited for practical purposes only to very limited extent in the near future (cf. the excellent survey paper [5]). Along similar lines KIF [19] was invented, but was most often only used at the syntactical rather than at the semantic level of knowledge transportation. Building on KIF, Gruber [22] has investigated the translation between languages using the frame ontology as its interlingua. Though the frame ontology is very useful (essentially it catches the primitives used in object-oriented database systems), the language is too restricted in general.

We have shown in this paper, how to use semantic patterns with OIL, a Web-compatible description logics framework and F-Logic, a language that had been intensively used for Semantic Web applications [28]. Thereby, our semantic patterns are not restricted to either of these paradigms or their integration.

6.3 Patterns and Problem Solving Methods

Design patterns [18] — and their knowledge engineering counterparts [10] — have proved extremely successful in describing characteristics of the *contents* that are to be described (algorithmic structures or knowledge structures). We in contrast have focused on the description of *language characteristics* in order to bridge between different representation languages, thus applying the paradigms of patterns at the meta-level.

A similar contrast holds between semantic patterns and problem solving methods. "Problem solving methods describe domain-independent reasoning components" [15]. They come at various levels of abstractions, from informal text, over few lines of pseudocode up to implementations in a particular language. Problem solving methods can be thought of as a variety of search methods with heuristics that benefit from domain specific knowledge where the heuristic is built into the problem solving method itself.

While on the very high level problem solving methods may appear similar to semantic patterns, there are several major distinctions, only two of which we want to mention here: First, semantic patterns only describe *what* needs to be inferenced they do not specify how semantic entailments are actually derived in a particular representation, which is the domain of problem solving methods that describe *how* to do things. Second, the domain proper of semantic patterns and problem solving methods is rather dissimilar. Typical problem solving method libraries include, e.g., "propose and revise", or "heuristic classification", while semantic patterns such as we propose abstract from language characteristics to include, e.g., "part-whole reasoning", "local inverses", or "inheritance with exceptions".

7 Conclusion

We have shown a new methodology, *viz. semantic patterns*, for engineering semantics on the Web in a way that makes it easier to reuse in a wide range of existing representation systems and easier to communicate between different Semantic Web developers. Semantic patterns are used to describe intended semantic entailments and, thus, allow a higher level of abstraction above existing Semantic Web languages — similarly as software design patterns allow to abstract from actual applications.

With this approach, there comes now the possibility to bridge between various paradigms for representation. By semantic patterns, the social process of designing new and communicating previously successful semantic patterns may now be started. The reader, however, may bear in mind that semantic patterns only provide a ground of discourse for man and machine. Which actual patterns will eventually turn out to be successful for which purpose will have to be shown over time by the Web community.

References

[1] Daml ontology library. http://www.daml.org/ontologies/, observed 2000.

[2] Daml-ont initial release. http://www.daml.org/2000/10/daml-ont.html Observed at October 12, 2000, 2000.

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The lorel query language for semi-structured data. *Journal of Digital Libraries*, 1(1):68–88, 1997.

[4] J. Angele, H.-P. Schnurr, S. Staab, and R. Studer. The times they are a-changin' — the corporate history analyzer. In D. Mahling and U. Reimer, editors, *Proceedings of PAKM-2000. Basel, Switzerland, October 30-31, 2000*, 2000.

[5] B. Bennett, C. Dixon, M. Fisher, E. Franconi, I. Horrocks, U. Hustadt, and M. de Rijke. Combinations of modal logics. submitted for publication, 2000.

[6] M. Blázquez, M. Fernández, J. M. García-Pinar, and A. Gómez-Pérez. Building ontologies at the knowledge level using the ontology design environment. In *In Proceedings of KAW-98, Banff, Canada, 1998*, 1998.

[7] R. Brachman. On the epistomological status of semantic networks. *Associative Networks*, pages 3–50, 1979.

[8] D. Brickley and R.V. Guha. Resource description framework (RDF) schema specification. Technical report, W3C, 1999. W3C Proposed Recommendation. http://www.w3.org/TR/PR-rdf-schema/.

[9] V. Christophides and D. Plexousakis, editors. *Proceedings of the ECDL-2000 Workshop — Semantic Web: Models, Architectures and Management*, 2000.

[10] P. Clark, J. Thompson, and B. Porter. Knowledge patterns. In A. Cohn, F. Giunchiglia, and B. Selman, editors, *In Proc. of KR-2000, Breckenridge, CO, USA, 12-15 April 2000*, pages 591–600, San Mateo, CA, 2000. Morgan Kaufmann.

[11] O. Corby, R. Dieng, and C. Hebert. A conceptual graph model for w3c resource description framework. In *In Proceedings of ICCS-2000. Darmstadt, Germany, August 2000*, LNAI. Springer, 2000.

[12] S. Decker. On domain-specific declarative knowledge representation and database languages. In *Proc. of the 5th International KRDB Workshop*, pages 9.1–9.7, 1998.

[13] S. Decker, D. Brickley, J. Saarela, and J. Angele. A query and inference service for RDF. In *QL'98 - The Query Languages Workshop*. W3C, 1998. http://www.w3.org/TandS/QL/QL98/.

[14] S. Decker, D. Fensel, F. van Harmelen, I. Horrocks, S. Melnik, M. Klein, and J. Broekstra. Knowledge representation on the web. In *Proceedings of the DL-2000, Aachen, Germany*, 2000.

[15] D. Fensel and E. Motta. Structured development of problem solving methods. *IEEE Transactions on Knowledge and Data Engineering*, to appear.

[16] R. Fikes, A. Farquhar, and J. Rice. Tools for assembling modular ontologies in Ontolingua. In *Proc. of AAAI 97*, pages 436–441, 1997.

[17] E. Franconi and G. Ng. The i.com tool for Intelligent Conceptual Modeling. In *Proceedings of 7th International KRDB Workshop. Berlin*. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-29/.

[18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns — Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[19] M. R. Genesereth. Knowledge interchange format. draft proposed american national standard (dpans). ncits.t2/98-004. http://logic.stanford.edu/kif/dpans.html seen at Sep 7, 2000, 1998.

[20] M. Ginsberg. Knowledge interchange format: the KIF of death. *AI Magazine*, 5(63), 1991.

[21] E. Grosso, H. Eriksson, R. W. Fergerson, S. W. Tu, and M. M. Musen. Knowledge modeling at the millennium — the design and evolution of Protégé-2000. In *In Proceedings of KAW-99, Banff, Canada, 1999*, 1999.

[22] T. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

[23] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of KR-98*, pages 636–647, 1998.

[24] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42, 1995.

[25] O. Lassila and R. Swick. Resource description framework (RDF). model and syntax specification. Technical report, W3C, 1999. W3C Recommendation. http://www.w3.org/TR/REC-rdf-syntax.

[26] A. Y. Levy and M.-C. Rousset. Combining horn rules and description logics in CARIN. *Artificial Intelligence*, 104(1-2):165–209, 1998.

[27] L. Morgenstern. Inheritance comes of age: Applying nonmonotonic techniques to problems in industry. *Artificial Intelligence*, 103:1–34, 1998.

[28] S. Staab, J. Angele, S. Decker, M. Erdmann, A. Hotho, A. Maedche, H.-P. Schnurr, R. Studer, and Y. Sure. Semantic community web portals. In *In Proc. of WWW9, Amsterdam, The Netherlands, May, 15-19, 2000*. Elsevier, 2000.

[29] S. Staab, M. Erdmann, A. Maedche, and S. Decker. An extensible approach for modeling ontologies in RDF(S). In Christophides and Plexousakis [9].

[30] S. Staab and A. Maedche. Ontology engineering beyond the modeling of concepts and relations. In *In Proceedings of the ECAI-2000 Workshop on Ontologies and Problem-Solving Methods. Berlin, August 21-22, 2000*, 2000.