Chapter XI
Semantic Annotation of Objects

Petr Křemen
Czech Technical University in Prague, Czech Republic

Miroslav Blaško
Czech Technical University in Prague, Czech Republic

Zdeněk Kouba
Czech Technical University in Prague, Czech Republic

ABSTRACT

Compared to traditional ways of annotating multimedia resources (textual documents, photographs, audio/video clips etc.) by keywords in form of text fragments, semantic annotations are based on tagging such multimedia resources with meaning of objects (like cultural/historical artifacts) the resource is dealing with. The search for multimedia resources stored in a repository enriched with semantic annotations makes use of an appropriate reasoning algorithm. Knowledge management and Semantic Web communities have developed a number of relevant formalisms and methods. This chapter is motivated by practical experience with authoring of semantic annotations of cultural heritage related resources/objects. Keeping this experience in mind, the chapter compares various knowledge representation techniques, like frame-based formalisms, RDF(S), and description logics based formalisms from the viewpoint of their appropriateness for resource annotations and their ability to automatically support the semantic annotation process through advanced inference services, like error explanations and expressive construct modeling, namely n-ary relations.

INTRODUCTION

This paper has been motivated by experience gained by the authors in the course of solving the EU project CIPHER (5th Framework Programme 2001-2004). The main output of the project was the Story Fountain (Mulholland et al., 2004)—a software tool providing intelligent support for story research and exploration. Story Fountain is based on a collection of semantically annotated stories and supports the user in exploring these stories, e.g. looking for similarities, finding a
chain of stories, which logically connects two stories etc.

The developed methodology and tools have been tested on two story collections. The first one was devoted to the Bletchley Park. Bletchley Park is located in Bletchley (now integral part of the city of Milton Keynes, UK). It is the place where the British Government’s Code and Cipher School was located during the Second World War. Nowadays, it is a museum of the code breaking work done there. The exhibition located in Bletchley Park emphasizes the influence it has had on contemporary communication and computing technology. The collection consisted of several hundreds of stories, mainly interviews with people having worked there. The other collection consisted of about 50 historical stories related to castles in Southern Bohemia, Czech Republic.

Story Fountain accepts semantic annotations expressed in a subset of Operational Conceptual Modeling Language—OCML (Motta, 1999). OCML is a frame based conceptual modeling language. See next chapter for more information on frames. A frame represents basically an n-ary relation between individuals. Hence, the modeling process is more straightforward in comparison with formalism based on description logics including at that time emerging standard OWL (W3C, 2004), which provide only binary relations and the annotation process is thus much more tedious work than in case of using frame based formalisms. This was the main reason, why OWL was not chosen as the formalism for semantic annotations at the beginning of the project.

Nowadays, OWL is a well established standard with a large community of users and developers. There exist a number of public domain reasoners and other supporting tools. For example we were missing a tool for semantic comparison of two ontologies in OCML—a kind of a semantic diff tool. In OWL environment, the availability of reasoners makes the development of such a semantic diff tool much easier. Another big problem in CIPHER was debugging of semantic annotations. If there was an inconsistency in the annotation, discovering its root was a hard problem for the author of the annotation in Story Fountain. On the other hand, description logics provide a well defined mathematical background, which makes possible to develop effective algorithms for detecting the smallest sets of inconsistent axioms.

The restriction to binary relations seems to be the only one important weak point of OWL as a formalism for authoring semantic annotations. Fortunately, there exists decidable description logic with possibility to express n-ary relations called DLR (Calvanese, 1998). DLR motivated the authors of this chapter to explore the possibility of using DLR as a primary formalism for semantic annotations with subsequent translation to OWL—a key opening the doors to the wealth of publicly available semantic web tools. The rest of this chapter deals with basic aspects of a new methodology for authoring semantic annotations using DLR and OWL based semantic web tools.

**STATE OF THE ART**

**Semantic Networks and Frames**

One of the well known knowledge representation formalisms are semantic networks. There are many particular implementations with different graphical notations and different semantics. In principal, a semantic network is a directed or undirected graph, where nodes represent concepts or individuals (instances of concepts) and edges represent semantic relations between various concepts (individuals). Typical relations are: (i) A is-a B—concept A is a specialization of concept B or an individual A is an instance of concept B, (ii) A is-part-of B meaning that B has A as part of itself, (iii) various types of associations. An interesting extension to this basic semantic network paradigm are conceptual graphs (Sowa, 1992). A simplified version of conceptual graphs
Semantic Annotation of Objects

was used as a basis for a graphical tool for semantic annotations (Uhlíř et al., 2005) in the CIPHER project. The user drew a conceptual graph, which was later transferred to the frame based OCML transcription. Experiments have shown that users intuitively understand better conceptual graphs than the structured expressions in a conceptual modeling language even if visualized by means of a form-based editor.

Frame-based conceptual languages mentioned above are based on the concept of frames introduced by Marvin Minsky (Minsky, 1975) as a framework for knowledge representation. According to him, a frame is “… a data-structure for representing a stereotyped situation, like being in a certain kind of living room, or going to a child’s birthday party. Attached to each frame are several kinds of information. Some of this information is about how to use the frame. Some is about what one can expect to happen next. Some is about what to do if these expectations are not confirmed. We can think of a frame as a network of nodes and relations. The “top levels” of a frame are fixed, and represent things that are always true about the supposed situation. The lower levels have many terminals—“slots” that must be filled by specific instances or data. Each terminal can specify conditions its assignments must meet. … Simple conditions are specified by markers that might require a terminal assignment to be a person, an object of sufficient value, or a pointer to a sub-frame of a certain type. More complex conditions can specify relations among the things assigned to several terminals.”

The concept of frames is similar to the object-oriented paradigm. Frames correspond to classes well-known in the OO world. Slots correspond to properties of classes (data members in OO programming). Slots may contain values. The value can be (i) again a frame (corresponds to meta-class—class relationship), (ii) an individual (instance of a class) or (iii) simple data value (non-object data types like integer). The relation of frame generalization/specialization corresponds to class inheritance forming a subsumption hierarchy.

Very often it is necessary to express restrictions on slots. That is why slots have so called facets (restrictions on slots or properties of slots). Facets determine e.g. (i) type of the slot, (ii) its cardinality, (iii) if the value of this slot is obligatory or not, etc. Type facet restricts the set of all possible values of the slot to the set of values compatible with defined slot type(s). Cardinality facet determines how many values can be filled in the slot simultaneously (typically one or many).

A set of mutually interrelated classes forms a formal explicit representation of concepts in a (specific) domain of discourse. An ontology together with a set of class instances (individuals) is often called a knowledge base.

A number of particular frame representation systems aimed at expressing knowledge bases were developed in recent years like Loom (see http://www.isi.edu/isd/LOOM/LOOM-HOME.html) or Ontolingua (see http://www-ksl-svc.stanford.edu:5915/). In order to achieve interoperability of various knowledge representation systems, there were attempts to develop generic protocols allowing portability of application between any particular knowledge representation systems compliant with the particular generic protocol. The most famous generic protocols are GFP (Generic Frame Protocol—see http://www.ai.sri.com/~gfp/) and its successor OKBC (Open Knowledge Base Connectivity—see http://www.ai.sri.com/~okbc/).

There exist a number of tools allowing for ontology authoring. Probably the most famous is the Protégé platform (see http://protege.stanford.edu/overview/index.html) supporting two main formalisms—frames and OWL. Protégé-Frames is an ontology editor aimed at authoring OKBC compliant ontologies. There exist plug-ins for main OKBC compliant knowledge representation systems and it is possible to create own plug-ins for not yet supported OKBC compliant ones.
Logic-Based Languages

During the past decade the attention of knowledge representation community has turned from frame-based systems and semantic networks to logic oriented languages. Simultaneously, introduction of the idea of the semantic web in 2001 by Tim Berners-Lee catalyzed research efforts in the area of semantic annotation and knowledge representation the web. First significant results, Resource Description Framework (RDF) and RDF-Schema (RDFS), have been developed and standardized as a W3C Recommendation in 2004.

RDF is essentially a graph representation of some domain knowledge. Figure 1 shows an example graph, each link of which represents one RDF statement in the form (subject: S, predicate: P, object: O). Subjects are labeled either by resources (anything that can be identified by an URI), or blank nodes (existentially quantified variables), predicates are labeled by resources and objects are labeled either by resources, blank nodes, or literals. From the RDF point of view, the graph in the Figure 1 is interpreted as 7 statements corresponding to its links, where the links labeled rdf:type have a special meaning. For example the leftmost link represents the triple (:b rdf:type :X) (all examples of RDF graphs will be given in the N3 notation) and says that a resource http://example.com/myknowledgebase#b is an instance, or a realization of a resource denoted by http://example.com/myknowledgebase#X.

RDFS complements RDF with the possibility to model simple statements about the domain structure, like class/property hierarchy and property domains and ranges. The RDF graph representation of RDF+RDFS (denoted as RDF(S)) restricts the form of described structural relations in the domain to just binary ones. Central in RDF(S) is the notion of a class (denoting a set of domain objects) and a property (denoting a binary relation over domain objects). The graph in Figure 1, viewed as an RDF(S) document, describes :X to be a subclass (i.e. subset) of an anonymous class _:q and _:q to be a subproperty (i.e. subrelation) of :p. This allows an RDF(S) reasoner to deduce that :a is related with :d also through the relation :p. In addition to these (rather natural) features, RDF(S) allows to model higher order statements for example by using one resource both as a class and as an instance. Although such modeling constructs might be useful in various applications, they compromise the possibility of translation such an RDF(S) document into the first order predicate calculus. While RDF(S) is a simple language, it is a wide-spread choice for representing simple taxonomies and semantic annotations.

Together with the evolution of the semantic web and of the description logics (Baader, 2003) a family of more expressive logic based languages has been studied. First of them, DAML, was developed by the DARPA project and was subsequently extended to DAML+OIL and to OWL, which became a W3C Recommendation in 2004. The OWL language is layered into three variants according to the expressivity. The OWL-full variant extends RDF(S) with various class/property constructs well-known from description logics. Most important OWL class and property constructors (in N3 notation) involve:

- **intersection, union, complement:** “class of women students” (Woman and Student)
  `_w owl:intersectionOf (Woman :Student) .

- **existential/universal quant:** “class of objects that have a son” (hasChild some Man)
  `_w rdf:type owl:Restriction .
  `_w owl:onProperty :hasChild .
  `_w owl:someValuesFrom :Man .

- **min/max/exact cardinality restrictions:**
  “class of those objects that have exactly 3 children” (hasChild exactly 3)
  `_w rdf:type owl:Restriction .
  `_w owl:onProperty :hasChild .
  `_w owl:cardinality 3 .
Semantic Annotation of Objects

Figure 1. An example RDF graph. URIs in node/link labels are shortened as usual; prefix : corresponds to the namespace http://example.com/myknowledgebase#, prefix rdf: to the RDF vocabulary namespace http://www.w3.org/1999/02/22-rdf-syntax-ns#, prefix rdfs: to the RDF Schema namespace http://www.w3.org/2000/01/rdf-schema#, and prefix owl: to the OWL vocabulary http://www.w3.org/2002/07/owl#

- **nominals**: “class containing exactly two object—John and Paul” (\{John, Paul\})
  \[\text{_:w} \text{ owl:oneOf} \ (\text{John} : \text{Paul})\ .\]
- **inverse properties**: “:hasChild is inverse of :hasParent”. \(\text{hasChild} \text{ subPropertyOf} \text{ hasParent}\)
  \(\text{:hasChild} \text{ owl:inverseOf} \text{:hasParent}\ .\)

Due to the correspondence to the logical calculus, each OWL knowledge base \(K\) consists of axioms, some of which are already known from RDF(S). Each axiom is represented as an atomic statement of the following form:

1. **property assertion**: \("a\ is related to \(d\ through \(q\" \ (q(a,d))\)
   \(\text{:a} \text{ q} \text{:d}\ .\)
2. **class assertion**: \("b\ is of type \(X\" \ (X(b))\)
   \(\text{:b} \text{ rdf:type} \text{:X}\ .\)

- **class subsumption, equivalence, complement**: \("X\ is subclass of the class of elements that are related with \(q\ to some element of \(Y\" \ (X \text{ subClassOf} q \text{ some} Y)\)
   \(\text{:X} \text{ rdfs:subClassOf} \text{_:q} \ .\)
   \(\text{_:q} \text{ rdf:type} \text{owl:Restriction}\ .\)
   \(\text{_:q} \text{ owl:onProperty} \text{:q}\ .\)
   \(\text{:q} \text{ rdfs:subPropertyOf} \text{:Y}\ .\)
- **property subsumption**: \("q\ is a subproperty of \(p\" \ (q \text{ subPropertyOf} p)\)
   \(\text{:q} \text{ rdfs:subPropertyOf} \text{:p}\ .\)
- **transitive, functional, inverse functional, symmetric properties**: “property :hasSpouse is symmetric”. \(\text{hasSpouse} \text{ subPropertyOf hasSpouse}\)
   \(\text{:hasSpouse} \text{ rdf:type} \text{owl:SymmetricProperty}\ .\)
- **property domains/ranges, instance identification, and other**, see (McGuinness 2004) for the full description of the OWL language.
The RDF(S) basis of OWL-full causes its undecidability due to possibility to model higher order constructs. To provide a decidable fragment of OWL-full, the OWL-DL variant of OWL language has been introduced. To stay decidable and to allow translation into first order predicate calculus, OWL-DL enforces some syntactic restrictions on an OWL document, like disallowing a resource to be used both as a class and as an instance. OWL-DL has direct correspondence to a subset of first order logic—the description logic $SHOIN(D)$. Evaluating the RDF graph in Figure 1 using the OWL-DL semantics, it can be inferred, in addition to the RDFS inferences mentioned above, that $b$ is related through the property $q$ to some (unspecified) instance of $X$.

The least expressive variant in the OWL language family is the OWL-Lite, that disallows using some of the above mentioned OWL constructs, like nominals and general form of number restrictions trading them for more efficient reasoning (consistency checking is EXPTime complete vs. NEXPTime complete for OWL-DL). However, optimizations implemented in most state of the art OWL reasoners (see below) seem to narrow the gap between OWL-Lite and OWL-DL for real world ontologies and leave OWL-DL the most popular variant of the OWL language, at present.

Although the expressivity of the OWL language is sufficient for many applications, it still lacks some practically useful constructs that do not compromise decidability and practical tractability, like qualified cardinality restrictions and more expressive role constructors. These extensions will be present in the upcoming extension of the OWL language, called OWL 2. In addition to the enhanced expressivity, several tractable fragments are specified within OWL2 to support more efficient, and even polynomial, reasoning for specific applications.

Languages, such as OWL 2, based on traditional DLs do not support modeling relations of arity greater than two. If a need emerges during the domain modeling to express an n-ary relation within OWL 2, it has to be expressed using binary roles. This transformation usually loses significant information about the n-ary relations and overwhelms the semantic annotation author with a bunch of rather unintuitive concepts and relations.

These problems show the importance of the information loss analysis and of finding ways to hide this binary representation from the user as much as possible. An n-ary description logic DLR that allows for expressing relations of arbitrary arity has been introduced in (Calvanese et al., 1998). In addition to its convenience for modeling of n-ary relations in complex domains, the expressive power of DLR allows for establishing a mapping from conceptual modeling languages such as UML, EER, and ORM2 to description logics (Berardi et al., 2005; Calvanese et al., 1998; Keet, 2007).

Basic building blocks of DLR are atomic concepts, atomic relations of arbitrary arity and individuals. Furthermore, there are several constructors (similar to the OWL ones) for creating complex concepts and complex relations. Comparing to the OWL, complex concept/role constructs in DLR it is necessary to point to the arguments of an n-ary relation. In the examples below we use binary relations $hasChild$, $hasDaughter$ and $hasSon$ and a ternary relation $isBetween$ with their natural meaning. Since there is no known correspondence between DLR and RDF(S), we provide just the abstract syntax for DLR statements and constructs:

- **class intersection, union, complement:**
  
  "class of woman students"
  
  $\text{Woman} \cap \text{Student}$.

- **existential quantification:** "class of individuals that have a child"
  
  $\exists I \text{ some hasChild}$.

- **min/max/exact cardinality restrictions:** "class of individuals that are at most between 3 pairs of individuals"
  
  $\forall I \text{ max 3 isBetween}$.
Relation constructors allow for construct complex relations from simpler ones:

- **intersection, union, complement (all arguments have to be relations of the same arity):** “binary relation of tuples between a parent and a son or daughter”
  
  \[ \text{hasSon or hasDaughter} \]

- **projection:** “ternary relation of tuples the second argument of which is of type Person”
  
  \[ (\text{Person} : 2/3) \]

Similarly to OWL, a DLR knowledge base consists of set of axioms of the following form:

- **concept inclusion axiom:** “class Man is a subclass of the class Person”
  
  \[ \text{Man subClassOf Person} \]

- **relation inclusion axiom:** “binary relation hasSon is a subrelation of the relation hasChild” (both relations have to be of the same arity)
  
  \[ \text{hasSon subRelationOf hasChild} \]

- **concept assertion:** “John is a Person”
  
  \[ \text{Person(John)} \]

- **relation assertion:** “individual y is between individual x1 and individual x2”
  
  \[ \text{isBetween(x1, y, x2)} \]

Use of relation inclusion axiom and relation assertion is restricted to only relations and tuples of the same arity.

**Tools for Semantic Annotation Authoring**

A number of semantic authoring tool has been developed since the late 90’s of the 20th century. First interactive semantic annotation authoring tool, Knowledge Annotator, was developed in 2001 at the University of Maryland. This tool allowed the user to annotate a HTML document using a very simple annotation language SHOE. An extended version of this system, called SMORE, allowed more expressive annotation languages, like RDF, RDFS and DAML+OIL. Both the Knowledge Annotator and SMORE system were supposed to support semantic annotations of HTML documents with no support for semi-automatic annotation authoring and inferences. Simultaneously, in 2001, two systems were developed at the University of Karlsruhe: OntoMat-Annotizer and OntoAnnotate. Both of these tools allowed to annotate web pages in the DAML+OIL or OWL format and offered to the user simple support for knowledge base consistency checking, without the possibility to explain modeling errors and of (semi-) automatic information extraction. In 2001, several other systems supporting using text-mining and machine learning methods for pattern extraction were introduced. Lockheed Martin Corporation developed the AeroDAML that uses text-mining methods to generate semantic annotations into predefined linguistic DAML+OIL ontologies automatically. Another tool, MnM, was developed at the Knowledge Media Institute, Open University. Comparing to the other mentioned systems, it allows to generate and store semantic annotations directly in the annotated document. Since 2001, a bunch of project emerged, namely WebKB, Anotea, CREAM, OntoBroker and other. For an in-depth survey of semantic annotation tools, see (Gómez-Pérez et al., 2005).

The most wide-spread system for ontology development and semantic annotation authoring is the above mentioned Protégé tool (Protégé, 2008). It is available in two versions, a frame-based version, and an OWL version. For both of these variants a bunch of plugins is available, ranging in functionality from inference support to visualization. The OWL version of the Protégé tool can be integrated with an OWL-DL reasoner.

To facilitate inference support for creating annotations many of the semantic annotation
tools are backed by an inference engine for the underlying language. We will survey shortly the best known semantic web inference engines used in connection with annotation tools. The Jena system provides an RDF/RDFS/OWL management system, which employs rule-based inference service for reasoning with RDF/RDFS ontologies and for incomplete reasoning with OWL. The most popular systems that provide a complete support for the OWL language are Fact++, Pellet and RacerPro. They are based on the highly optimized tableau-based decision procedures. Both Fact++ and Pellet could be integrated into the Protégé system, thus providing Protégé users with consistency checking, error explanations, classification and other inference services. An alternative to the tableau reasoning for a subset SHIQ of OWL2 has been introduced by the system KAON2, which translates the knowledge base to a disjunctive datalog program.

INFERENCe SUPPORT FOR CREATING SEMANTIC ANNOTATIONS

The problem of creating semantic annotations in any of the languages mentioned in the previous sections is a very tedious and time-consuming task. During the Cipher project (cipher, 2004) several dozens of historical narratives were annotated to facilitate an efficient search through various aspects of the historical domain, like important historical events, actors of these events and their locations. All of these concepts have different granularities (for example, each father is a special kind of man) and different attributes (for example, each historical event has some date).

To navigate the annotation creator through this (rather complex) domain structure, as well as through (typically much larger) set of annotations, it seems crucial to provide some form of automated support. Unfortunately, fully automated creation of semantic annotations of natural language documents and all types of multimedia is a complicated task that requires natural language processing and speech recognition techniques. While both of these areas are rapidly evolving during last years, they are still far from being useful without human interaction. Thus, the natural language processing and speech recognition techniques might be used just as preprocessing of the documents/multimedia resources to produce a start-up knowledge base that has to be reviewed by human. Keeping this in mind, in the rest of the paper we will deal only with semi-automatic annotation creation, i.e. with inference services of the chosen modeling language that provide significant and useful help for manual creation of semantic annotation creation.

Next paragraphs describe several convenient services, each of which addresses a different problem that occurs when creating semantic annotations. First paragraph focuses on the problem of detecting and explaining inconsistencies in the semantic annotation repository. When authoring/managing large annotation knowledge bases it occurs frequently, that some assertions in such a knowledge base are inconsistent. Unfortunately, even if such an inconsistency is detected, the user can hardly get rid of it without knowing what the causes of it are.

As presented in the previous sections modeling language families differ in their expressivity and, in particular, in their support for modeling relations of arbitrary arity—besides binary ones. Unfortunately, current state-of-the-art languages for knowledge representation in the semantic web, like RDFS and OWL, support just binary relations. The second paragraph discusses this problem and proposes several approaches for modeling relations of arbitrary arity in such cases.

Modeling Error Explanations

Informing the author of semantic annotations about inconsistencies and about reasons why these inconsistencies occur in the repository of semantic annotations is of crucial importance, especially
for large repositories. Although the notion of inconsistency can be theoretically studied for frame-based systems and other formalisms that do not have foundations in logics, tracking back sources of an inconsistency in these languages is usually rather trivial and thus, we will stick our attention to explaining inconsistencies in logic-based formalisms, although some of the discussed methods will have much broader application area.

Most of the methods that deal with explaining modeling errors in an inconsistent logic knowledge base $K$ focus on finding the smallest subset $K'$ of $K$ that is inconsistent. Providing such $K'$ (usually several orders smaller than $K$) to the user helps him/her to localize the problem and solve it, e.g. by removing one of the axioms in $K'$ from $K$. In general, there might be more smallest subsets $K'$, $K''$, … of $K$ (also known as minimal unsatisfiable subsets, or MUSes). In this case it is necessary to make each of these subsets consistent, e.g. by removing an axiom from all of them.

Nowadays, there are three general approaches to find the subsets $K'$, $K''$, … of $K$ that differ in the level of their integration to the underlying consistency checking service. The black-box methods take an arbitrary knowledge base $K$ and an arbitrary inference engine for checking consistency of the language under consideration. There is a variety of black box methods, both well-known from other research areas and original ones. Here, we will briefly describe only some of them, referencing interested readers to (Křemen and Kouba, 2008) for more details.

A well-known method (De La Banda, et al. 2003) for error explanations uses the notion of so called CS-tree. A CS-tree represents a search tree of all subsets of a given set of axioms. Each node in a CS-tree is of the form $D$, $P$, where $D$ are axioms that belong to all MUSes represented by the subtree of the current node, and $P$ of axioms that might, but need not belong to some of the MUSes. Each of its $|D|+|P|$ children lacks one axiom from either $D$ or $P$. The algorithm starts with all axioms in the P part of the node, leaving D empty. The algorithm traverses the CS-tree in the depth-first manner, performing one consistency check at each explored node. If the explored node is consistent, then all of its children are pruned (consistent knowledge base has only consistent subsets). Whenever an inconsistent node without children (or with only pruned ones) is reached, it corresponds to a MUS and the search backtracks. CS-tree algorithms might be augmented with additional pruning heuristics, constraint set partitioning and eliminating always satisfiable constraints, see (De La Banda et al., 2003) for more details. As an example let’s take an OWL-DL knowledge base containing 3 axioms (written in DL-notation):

1. Car subClassOf Vehicle and (hasWheel exactly 4)
2. (Car and (hasWheel exactly 3))(myVelorex)
3. (Car and hasWheel only Nothing)(justCar-Body)

The algorithm to find these MUSes automatically traverses the CS-tree as depicted in Figure 2. The inconsistency of the knowledge base is represented by the root node. Then, 3 possibilities are checked (3 successors of the root): either axiom 1 is certainly not in the MUS (left branch), or axiom 2 certainly does not belong to the MUS, while axiom 1 certainly does (central branch), or both axioms 1 and 2 certainly belong to the MUS, and axiom 3 certainly does not (right branch). In the right branch, the set of axioms $\{2,3\}$ turns out to be consistent and subsequent branches (shown in gray color) need not be tested. In the very same manner the rest of the tree is traversed and the MUS $\{1, 2\}$ is found (there is one more MUS $\{2, 3\}$ for this knowledge base).

Although various optimizations and heurstics could be used to prune the search space, the problem of computing all MUSes has exponential complexity, in the worst case.
On the other hand, users of annotating tools might not need all MUSes at a moment. Instead, a single MUS, that localizes one of the modeling errors, may be sufficient. The reparation procedure is then incremental—a single MUS is found, the user fixes it and if the resulting KB is still inconsistent, the procedure is repeated. There is a simple method (Schlobach, 2005) for computing a single MUS with polynomial complexity. In the first phase, this algorithm starts with an empty set $S$ and fills it with all available axioms one by one until it becomes inconsistent. In the second phase, each axiom is conditionally removed from $S$. If the new $S$ turns satisfiable, the axiom is put back. The resulting set of axioms is a MUS. For more details, see (Schlobach and Huang, 2005).

More detailed survey of applications of black-box methods to error explanations is given in (De La Banda et al., 2003). Although their wide reusability, black-box methods typically suffer, especially when used with expensive consistency checking procedures, with poor scalability, see (Křemen and Kouba, 2008) for more details.

While black-box methods do not need to know any details about internals of the underlying consistency checking procedure, incremental methods for error explanation additionally need access to the consistency checking algorithm state and restoring the originally saved state. This slightly refined interface to the reasoned allows us to avoid repeated generation of the internal state structure, for example a completion graph in case of a tableau algorithm. An interesting approach to compute all MUSes is presented in (De La Banda et al., 2003). It proposes to search through a tree, that is a generalization of the CS-tree described above. In addition to the $D$ and $P$ sets, each node contains also a set $T$ that represents axioms from $D \cup P$, such that $D \cup T$ is satisfiable, and $sD$, $sT$—states of the reasoner corresponding to the sets of axioms $D$, resp. $T$. To find all MUSes, this tree is searched in the depth-first manner, performing several incremental tests at each node, stopping whenever an inconsistency occurs. In an analogous way to the black-box methods, (Křemen and Kouba, 2008) presents an optimization of this method and a method for computing a single MUS using the incremental techniques. All incremental methods require, in general, more interactions with the reasoner than the black-box ones, but these interactions/consistency checks are usually much cheaper. Thus, the performance is typically several times better (Křemen and Kouba, 2008) than the performance of black-box methods.

The most efficient type of error explanation techniques are glass-box methods, i.e. methods that are fully integrated into the consistency
checking service itself. Currently, to the best of authors’ knowledge, there is a lack of glass-box techniques for most of the expressive logic-based languages (including OWL). Let’s briefly introduce a glass-box technique (Schlobach and Huang, 2005) for the description logic \textit{ALC} (Baader et al., 2003). Consistency of an ALC knowledge base is typically checked with a tableau algorithm implementation. The explanation extension of the consistency checking procedure keeps the history of the generated candidate models (completion graphs) to track dependencies of the generated assertion to the axioms. Whenever a clash occurs in a completion graph, all axioms corresponding to the clashing concepts are output as the explanation for the clash. Direct extension of this idea (Kalyanpur, 2006) to the language OWL-DL is incomplete, due to the interaction of the tracked dependencies with tableau expansion rules for OWL-DL, so that it can be used only as a (usually quite efficient) preprocessing step. After the preprocessing the resulting axioms have to be handled using one of the incremental/black-box techniques mentioned above.

\section*{Expressive Modeling Constructs}

In July 2004, W3C addressed the OWL shortcoming of expressing relationships of higher arity in a Working Group Note (Noy and Rector, 2006). The document describes two different ontology design patterns to represent n-ary relations in semantic web languages RDF(S) and OWL. First of the techniques is useful in situations where n-ary relations have fixed number of arguments, while the second one describes relations that are much like lists or sequences of arguments. In this paragraph, we will concentrate on the first case - relations of fixed arity. We show that there are cases for which the rather light-weight solution suggested in (Noy and Rector, 2006) is not sufficient and propose a correct solution based on reification of DLR first introduced in (Calvanese, 1998). At the end we show how this transformation can be used in semantic annotation authoring tools.

The document proposes to model n-ary relations with fixed number of arguments by so-called \textit{reification}. The main idea behind \textit{reification} is that n-ary relations are represented by fresh atomic concepts and as many roles as is the arity of the relation. A tuple of the relation is represented as an instance of the fresh concept and it is linked via dedicated roles to individuals representing the components of the tuple.

There are three different use cases of the reification pattern defined in the document. Each of them explains how dedicated roles should be connected to reified relation and what additional conditions should be applied along these roles. First two use cases address situations in which there is one argument of the relationship that acts as a subject or an “owner” of the relation. The third use case describes a situation in which all participants of the n-ary relation are equally significant. Let’s consider an example about the person Deidre Capron who has worked on an assignment in Bletchley Park for 5 months. Such knowledge can be modeled by introducing a new concept \textit{Assignment} that represents the reified version of the relation work and new roles \textit{hasParticipant}, \textit{hasLocation}, and \textit{hasDuration} that connect instances of the \textit{Assignment} concept to the components of the relation, as depicted in the Figure 2. For the sake of this example, we specify that each assignment has exactly one participant, location and duration.

Properties of the reified relation \textit{work} are captured by the definition of concept \textit{Assignment}, i.e. it is stated that each individual of the concept is connected to exactly one individual for each of three roles (e.g. for role \textit{hasLocation} it could be expressed by axioms \textit{Assignment subClassOf hasLocation some Place}, \textit{Assignment subClassOf hasLocation max 1}).

In order to complete our goal to represent n-ary relation \textit{work} correctly, without an in-
formation loss, there is one missing constraint that concept Assignment has to satisfy. In the Figure 3 individual assignment_0 represents tuple <DeidreCapron, BletchleyPark, 9Months>. However, there is no constraint that would rule out another individual (e.g. assignment_1) different from assignment_0 to represent same tuple. This problem of reified relation is referred to as tuple-admissibility problem. We say that reified relation is tuple-admissible if it is interpreted as set of tuples and thus the reification concept contains only one representative for each instance of a tuple. Solution to tuple-admissibility problem goes beyond the scope of the W3C document for defining n-ary relations.

Another interesting work on reification of n-ary relations can be found in (Calvanese, 1997). In the work, Calvanese used DLR to solve query containment problem (QCP). Part of the solution of QCP was satisfiability problem of DLR which was translated and delegated to satisfiability problem of CIQ DL knowledge base. Since there was no implementation of reasoner based on CIQ DL, Calvanese et al. solution did not lead to practical decidability. To overcome this problem, (Horrocks et al., 2000) introduced slightly different mapping of QCP to DLR and translated it to satisfiability problem of the description logic SHIQ knowledge base that is a strict subset of the OWL 2 language. This tuple-admissible transformation works as follows:

For each arity $i$ of relations occurring in original DLR knowledge base (KB), a concept $T_i$ is defined that corresponds to all reified tuples of the arity $i$. $T_i$ corresponds to concept of all simple individuals. Next, new functional roles $f_i$ are introduced to represent $i$-th component of reified tuples ($f_i$ functional). The interpretation domain of reified counterpart is defined as a union of all concepts $T_i$ ($T_i$ subClassOf $T_i$ or $T_{i+1}$ or ... or $T_{max}$) that are mutually disjoint by its definitions. Each $T_i$ contains only individuals that are connected to exactly $i$ individuals (components of a tuple) of concept $T_1$, each by a dedicated functional role $f_i$:

$$T_i \text{ equivalentClass } f_i \text{ some } T_j \text{ and } ... \text{ and } f_i \text{ some } T_j \text{ and } f_{i+1} \text{ only Nothing, } f_i \text{ only Nothing } \text{ subClassOf } f_{i+1} \text{ only Nothing.}$$

In addition, it is stated that each reified counterpart of an atomic concept and an atomic relation of arity $i$ is subconcept of $T_j$ and $T_i$ respectively.
With such a structure of binary knowledge base it is proved that each DLR inclusion axiom can be straightforwardly transformed to equivalent axiom of reified counterpart without compromising tuple-admissibility.

From the semantic annotation point of view, it is interesting to pinpoint, how DLR assertion axioms are transformed to the binary representation. For each asserted tuple occurring in original DLR knowledge base a new individual is introduced along with an axiom that ensures uniqueness of this tuple representative and thus supply tuple-admissibility of reified relation. DLR concept assertions are transformed directly to concept assertions of reified counterpart. DLR relation assertions of arity $i$ are transformed to a concept assertion, which states that tuple representative belongs to reified counterpart, and $i$ role assertions, which state that tuple representative is connected to its components.

For demonstration of the transformation we extend our example n-ary knowledge base about Deidre Capron to contain binary relation hasAge and a fact that Diedre Capron is 18 years old. The structure of reified counterpart according to Horrocks is shown in Figure 4. In the figure, super concept of all concepts ($T$) is divided into three disjoint subconcepts $T_1$, $T_2$, and $T_3$. DLR relations hasAge and work are reified to concepts $A_{hasAge}$ and $A_{work}$ that are subconcepts of $T_2$ and $T_3$ respectively. For the triple <DeidreCapron, BletchleyPark, 9Months> a new individual $t_{work_0}$ is introduced and linked through roles $f_1$, $f_2$, $f_3$ to the corresponding components of the triple. In much the same way a new individual $t_{hasAge_0}$ is linked to DeidreCapron and 18Years by roles $f_1$ and $f_3$ respectively.

There is a limited support for expressing n-ary relations in OWL in most semantic annotation tools. The well known ontology editor (Protege, 2008) contains a wizard for creating n-ary relations according to the W3C design patterns. The main benefit of the wizard is that it unifies common tasks in creation of n-ary pattern and helps user to specify similar names for some components of the pattern by specifying prefixes and suffixes.

Figure 4. Reification of DLR knowledge base about Deidre Capron (according to Horrocks et al.)
of new OWL entities. Major disadvantage of this approach is that after completing the wizard, the ontology editor produces the corresponding OWL axioms directly and throws away the information about the correspondence between an n-ary relation and its reified counterpart.

Thus, there is no way to change already created ontology pattern in a systematic manner (e.g. change all prefixes of similar components without changing each of them separately) or provide other extended functionality to the editor that makes use of the n-ary nature of the knowledge (e.g. in definition of an n-ary relation assertion the ontology editor could provide slots for all individuals of the assertion without having to define tuple representative or its relationships to all individuals of the assertion).

Our current research in n-ary relations focuses on the use of a simplified version of DLR for semantic annotations of documents. Consistency of the annotations can be checked in OWL 2 after performing the reification of n-ary relation assertions, as explaining earlier, which facilitates the use of other, more advanced, inference services that can be reduced to the consistency checking problem of a DL knowledge base.

One of such advanced services is error explanations, as described in the previous section. The explanation of inconsistencies in a DLR knowledge base is a bit more complicated than in the binary case. DLR can be again reified to OWL ontology, however, after finding MUSes in the ontology, we need to transform the result back into DLR to present it to the user. For this purpose we developed transformation from reified OWL ontology back to DLR knowledge base. This can be done thanks to advanced annotation capabilities of OWL2. In OWL2 it is possible to annotate each OWL entity (i.e. class, property or individual) using a rich annotation system. When transforming DLR to OWL, the annotations can be used to keep extra information the reification process that is used back in the reverse transformation. Note that added annotations does not alter semantics of OWL knowledge base and that using the transformation from OWL to DLR it is possible to keep all annotations in the OWL ontology. This technique could also be a direction for future work and integration within ontology editors like Protege.

**CONCLUSION AND FUTURE WORK**

This chapter discusses some of the most important problems that occur when creating semantic annotations of multimedia/textual resources during the CIPHER project. First, relevant formalisms, like frames, semantic networks and description logics, are introduced. The above discussion shows why the latter is considered the most advantageous choice for creating semantic annotations. In spite of the difficulties of description logics to model n-ary relations, they provide a well-defined semantics that opens the door to advanced and useful services, like modeling error explanations. The subsequent sections present the state of the art in error explanation techniques for OWL Web Ontology Language and the possibility of solving the problems with modeling n-ary relations in OWL.

The problematic of creating semantic annotations is very complex, thus leaving this chapter necessarily incomplete. Practical tools that support creating annotations will certainly exploit the data mining and pattern extraction techniques to guide the user during the annotation process, based on the annotation patterns the user has created so far. Another important point is the support for macros—n-ary relation transformation presented above can be considered as a special case of such macros. Last but not least, it might be useful to preprocess the multimedia resources/documents with some application specific techniques. For example, the natural language processing techniques might provide a first starting set of annotations that is later edited by the user.
ACKNOWLEDGMENT

This work has been supported by the grant No. MSM 6840770038 Decision Making and Control for Manufacturing III of the Ministry of Education, Youth and Sports of the Czech Republic.

REFERENCES


**KEY TERMS AND DEFINITIONS**

**Description Logics:** A family of decidable logic based knowledge representation formalisms that is a basis for the current semantic web technologies.

**Error Explanation:** A complex inference service, that provides the user of a semantic annotation tool with a concise explanation of given modeling problem, like the knowledge base inconsistency.

**Frames:** A knowledge representation paradigm that allows to group knowledge into logically interrelated pieces—frames. It is close to the object-oriented technology.

**Knowledge Base:** A common repository of semantic annotations to facilitate a fast and efficient search in the given set of resources. Binding a knowledge base to an ontology gives the semantic meaning to the annotations according to the domain structure described in the ontology.

**Ontology:** A conceptual model of the domain under consideration. It defines the structure of the domain, by the specification of the relevant concepts and structural relations between them.

**Reification:** A process of converting an n-ary knowledge into the binary one.

**Semantic Annotation of a Resource:** The process of enriching a resource (text, multimedia) with tags, the meaning of which is specified in the chosen ontology. The main purpose of this process is to provide a structure to—formerly unstructured or semi-structured data—to enable an efficient searching in the resource repository.