Relating Ontology Languages and Web Standards

Dieter Fensel

Abstract

Currently computers are changing from single isolated devices to entry points in a world wide network of information exchange and business transactions called the World Wide Web (WWW). Therefore support in data, information, and knowledge exchange becomes the key issue in current computer technology. Ontologies provide a shared and common understanding of a domain that can be communicated between people and application systems. Therefore, they may play a major role in supporting information exchange processes in various areas. However, in order to develope their full power, the representation languages for ontologies must be compartible with existing data exchange standards in the World Wide Web. In this paper we investigate protoypical examples of ontology languages and web languages. Finally we show how these web standards can be used as a representation languages for ontologies.

1 Introduction

Ontologies are a popular research topic in various communities such as knowledge engineering, natural language processing, cooperative information systems, intelligent information integration, and knowledge management. They provide a *shared and common* understanding of a domain that can be communicated between people and heterogeneous and distributed application systems. They have been developed in Artificial Intelligence to facilitate *knowledge sharing and reuse*.

Ontologies are formal theories about a certain domain of discourse and therefore require a formal logical language to express them. The paper will discuss languages for describing ontologies in Section 2. In Section 3 we will investigate how recent web standards such as XML and RDF can be used as languages for expressing Ontologies, or at least some of their aspects. Finally, in Section 4 we will discuss XOL, a proposal for an XML-based standard for expressing ontologies.

2 Ontology Languages

For our discussion we have chosen:

- CycL and KIF [Gene 1991] as representatives for enriched first-order predicate logic languages.
- Ontolingua [FFR 1997] and Frame Logic [KLW 1995] as representatives for frame-based approaches. Both incorporate frame-based modeling primitives in a first-order logical framework, however they apply very different strategies for this.
- Description Logics that describe knowledge in terms of concepts and role restrictions used to automatically derive classification taxonomies.

2.1 First-order predicate logic languages CycL and KIF

CycL has been developed in the Cyc project [LeGu 1990] for the purpose of specifying the large common-sense ontology that should provide Artificial Intelligence to computers. Far from having attained this goal, Cyc still provides the worldwide largest formalized ontology. CycL is a formal language whose syntax is derived from first-order predicate calculus. However, CycL extends first-order logic through the use of *second* order concepts. Predicates are also treated as constants in expressions. The vocabulary of CycL consists of terms: semantic constants, non-atomic terms, variables, numbers, strings, etc. Terms are combined in CycL expressions, ultimately forming closed CycL sentences (with no free variables). A set of CycL sentences forms a knowledge base. In the following, we will discuss the main concepts of CycL. More details can be found at its homepage[1].

Constants are the vocabulary of the CycL language; more precisely, they are the "words" that are used in writing the axioms (i.e., the closed formulas) that comprise the content of any CycL knowledge base. Constants may denote (1) individuals, (2) collections of other concepts (i.e., sets which correspond to unary predicates), (3) arbitrary predicates that enable the expression of relationships among other constants, and functions. Constants must have unique names.

Predicates express relationships between terms. The *type* of each argument of each predicate must be specified; that is the appropriate formulas must be asserted to be true, i.e. $p\ (A_0)$ with A_0 of type T and $p(c)$ implies that c is element of T.

Variables stand for terms (e.g., constants) or formulas whose identities are not specified. A variable may appear anywhere that a term or formula can appear.

Formulas combine terms into meaningful expressions. Every formula has the structure of a parenthesized list. That is, it starts with a left parenthesis, then there follow a series of objects which are commonly designated A_0, A_1, A_2, etc., and finally there is a matching right parenthesis. The object in the A_0 position may be a predicate, a logical connective, or a quantifier. The remaining arguments may be terms (e.g., constants, non-atomic terms, variables, numbers, strings delimited by double quotes ("...")), or other formulas. Note the recursion (i.e., the second-order syntax) here; A_i in one formula might itself be an entire CycL formula. Every atomic formula must begin with a predicate or a variable in order to be well-formed. The simplest kind of formula is a formula in which the A_0 position is occupied by a predicate, and all the other argument positions are filled with terms (or variables):

(likesAsFriend DougLenat KeithGoolsbey)
(colorOfObject ?CAR ?COLOR)

The first formulas above is called a ground atomic formulas, since none of the terms filling the argument positions are variables. The second formula is not a ground atomic formula; it refers to the variables ?CAR and ?COLOR.

Logical connectives are used to built more complex formulas from atomic formulas (and/or other complex formulas). The most important logical connectives are: *and*, *or*, *not*. New connectives can be introduced simply by asserting a formula to that effect into the knowledge base; thus
(*isa new-connector Connective*).

Complex Formulas. We can compose the above connectives, of course, and have complex expressions such as

1. http://www.cyc.com/cycl.html/

(*and*... (*or* ... (*xor A* (*and* ...))...)...)

Quantification: universal quantification and existential quantification. Universal quantification corresponds to expressions like every, all, always, everyone, and anything, while existential quantification corresponds to expressions like someone, something, and somewhere. CycL contains **one universal quantifier**, *forAll*, and **four existential quantifiers**, *thereExists*, *thereExistAtLeast*, *thereExistAtMost*, and *thereExistExactly*. Additional quantifiers can be introduced by making the appropriate assertions -- declaring the new quantifier to be an instance of Quantifier, and giving a definition of it, probably in terms of existing quantifiers, predicates, and collections. To be considered a closed sentence -- a well-formed formula -- all the variables in an expression need to be *bound* by a quantifier before they are used.

Second-order Quantification: Quantification is also allowed over predicates, functions, arguments, and formulas.

Functions. Like most predicates, most functions have a fixed arity. For each function assertions that specify the type of each argument must be entered into the CycL knowledge base.

Microtheories ([LeGu 1990], [Guha 1993]). A microtheory, or context ([McCa 1993], [Lena, subm.]), is a set of formulas in the knowledge base. Each formula must be asserted in at least one microtheory. Thus a particular formula might be contained in many microtheories, but each particular assertion of that formula was only made into one single microtheory at the time that assertion was made. Microtheories are fully reified objects, and thus they cannot only contain CycL formulas, they can also participate in CycL formulas.

Each formula has an associated **truth value** (in each microtheory). CycL contains five possible non-numeric truth values, of which the most common are default true and monotonically true. The other truth values are default false, monotonically false, and unknown. In addition, CycL accommodates Bayesian probabilities and dependencies, and (separately) fuzzy truth values, attributes and sets. All CycL-compliant system must support at least one “true” and one “false”.

- **Monotonically true** means: true with no exceptions. Assertions which are monotonically true are held to be true in every case, that is, for every possible set of bindings -- not just currently known bindings -- to the universally quantified variables (if any) in the assertion, and cannot be overridden.
- Assertions which are **default true**, in contrast to monotonically true, can have exceptions. They are held to be true only in most cases (usually meaning most of the relevant cases likely to be encountered in the current context), and can be overridden without the need for alerting the user

In a nutshell, CycL uses predicate logic extended by *typing* i.e., functions and predicates are typed, *reification* i.e., predicates and formulas are treated as terms and can be used as expressions within other formulas, and microtheories that define a *context* for the truth of formulas.

The *Knowledge Interchange Format KIF* [GeFi 1992] is a language designed for use in the exchange of knowledge between disparate computer systems (created by different programmers at different times, in different languages, and so forth). Different computer systems can interact with their users in whatever forms are most appropriate to their applications (for example Prolog, conceptual graphs, natural language, and so forth).

Being a language for knowledge interchange KIF can also be used as a language for expressing and exchanging Ontologies. The following categorical features are essential to the design of KIF.

- The language has *declarative semantics*.
- The language is *logically comprehensive* -- at its most general, it provides for the expression of arbitrary logical sentences. In this way, it differs from relational database languages (like SQL) and logic programming languages (like Prolog).
- The language provides *means for the representation of knowledge about knowledge*. This allows the user to make knowledge representation decisions explicit and to introduce new knowledge representation constructs without changing the language.

Semantically, there are four categories of **constants** in KIF -- object constants, function constants, relation constants, and logical constants. Object constants are used to denote individual objects. Function constants denote functions on those objects. Relation constants denote relations. Logical constants express conditions about the world and are either true or false. KIF is unusual among logical languages in that there is no syntactic distinction among these four types of constants; any constant can be used where any other constant can be used. The differences between these categories of constants are entirely semantic. This feature *reifies* second-order features in KIF. It is possible to make statements about statements.

There are three disjoint types of **expressions** in the language: *terms*, *sentences*, and *definitions*. Terms are used to denote objects in the world being described; sentences are used to express facts about the world; and definitions are used to define constants. A *knowledge base* is a finite set of sentences and definitions.

There are six types of **sentences**.

sentence ::= constant | equation | inequality | relsent | logsent | quantsent

We have already mentioned constants. An equation consists of the = operator and two terms. An inequality consist of the /= operator and two terms. An implicit relational sentence consists of a constant and an arbitrary number of argument terms terminated by an optional sequence variable.

The syntax of **logical sentences** depends on the logical operator involved. A sentence involving the not operator is called a negation. A sentence involving the and operator is called a conjunction and the arguments are called conjuncts. A sentence involving the or operator is called a disjunction and the arguments are called disjuncts. A sentence involving the => operator is called an implication; all of its arguments but the last are called antecedents and the last argument is called the consequent. A sentence involving the <= operator is called a reverse implication; its first argument is called the consequent and the remaining arguments are called the antecedents. A sentence involving the <=> operator is called an equivalence.

There are two types of **quantified sentences** -- a universally quantified sentence is signaled by the use of the forall operator, and an existentially quantified sentence is signaled by the use of the exists operator. The first argument in each case is a list of variable specifications. Note that according to these rules it is permissible to write sentences with *free* variables[2], i.e. variables that do not occur within the scope of any enclosing quantifiers.

2. Very different from CycL where free variables are forbidden.

Finally, there are three types of **definitions** -- unrestricted, complete, and partial. Within each type there are four cases, one for each category of constant. For more details see the KIF homepage.[3]

KIF and CycL have features in common. Both languages are oriented on predicate logics. Also, both provide an important extension of first-order logic. They allow the reification of formulas as terms used in other formulas. Therefore, KIF and CycL allow meta-level statements. In addition to this, CycL provides richer modeling primitives than KIF (e.g., various quantifiers and micro theories). This stems from the fact that CycL is a modeling language for ontologies whereas KIF was designed as an exchange format for ontologies. As we will discuss later, both languages are close in spirit to RDF. Second-order elements (i.e., formulas as terms in meta-level formulas) and global scope of properties (i.e., predicates) are common features.

2.2 Frame-based Approaches: Ontolingua and Frame Logic

The central modeling primitive of predicate logic are predicates. *Frame-based* and *object-oriented* approaches take a different point of view. Their central modeling primitive are classes (i.e., frames) with certain properties called attributes. These attributes do not have a global scope but are only applicable to the classes which they are defined for (they are typed), and the “same” attribute (i.e., the same attribute name) may be associated with different range and value restrictions when defined for different classes. In the following we will discuss two frame-oriented approaches: Ontolingua (cf. [Grub 1993], [FFR 1997]) and Frame logic [KLW 1995].

Ontolingua[4] was designed to support the design and specification of ontologies with a clear logical semantics based on KIF. Ontolingua extend KIF using additional syntax to include the intuitive bundling of axioms into definitional forms with ontological significance and a Frame Ontology to define object-oriented and frame-language terms.[5] The set of KIF expressions that Ontolingua allows is defined in an ontology called the Frame Ontology. The Frame Ontology specifies the representation primitives that are often supported with special-purpose syntax and code in object-centered representation systems (e.g., classes, instances, slot constraints, etc.). Ontolingua definitions are Lisp-style forms that associate a symbol with an argument list, a documentation string, and a set of KIF sentences labeled by keywords. An Ontolingua ontology is made up of definitions of classes, relations, functions, distinguished objects, and axioms that relate these terms.

A **relation** is defined with a form like the following:

(define-relation *name* (?A_1 ?A_2)
:def (KIF formula)

The arguments ?A_1 and ?A_2 are universally quantified variables ranging over the items in the tuples of the relation. This example is a binary relation, so each tuple in the relation has two items. Relations of greater arity can also be defined. The sentence after the *:def* keyword is a KIF sentence stating logical constraints over the arguments. Constraints on the value of the first argument of a binary relation are domain restrictions, and those on

3. http://logic.stanford.edu/kif/kif.html

4. http://ontolingua.stanford.edu/

5. The Ontolingua Server as described in [FFR 1997] has extended the original language by providing explicit support for building ontological modules that can be assembled, extended, and refined in a new ontology.

the second argument of a binary relation are range restrictions. There may also be complex expressions stating relationships among the arguments to a relation. The *:def* constraints are necessary conditions, which must hold if the relation holds over some arguments. It is also possible to state sufficient conditions or any combination.

A **class** is defined by a similar form with exactly one argument called the instance variable. In Ontolingua, classes are treated as unary relations to help unify object- and relation-centered representation styles.

A **function** is defined like a relation. A slight variation in syntax moves the final argument outside of the argument list. As in definitions of relations, the arguments to a function are constrained with necessary conditions following the *:def* keyword.

Finally, it is possible to define distinguished **individuals** in an ontology.

The frame ontology is expressed as second-order axioms in Ontolingua. It contains a complete axiomatization of classes and instances, slots and slot constraints, class and relation specialization, relation inverses, relation composition, and class partitions. Each second-order term is defined with KIF axioms. A list of the Frame Ontology vocabulary is given in Figure 1.

Frame logic [KLW 1995] is a language for specifying object-oriented databases, Frame systems, and logical programs. Its main achievement is to integrate conceptual modeling constructs (classes, attributes, domain and range restrictions, inheritance, axioms) into a coherent logical framework. Basically it provides classes, attributes with domain and range definitions, is-a hierarchies with set inclusion of subclasses and multiple attribute inheritance, and logical axioms that can be used to further characterize the relationships between elements of an ontology and its instances.

The alphabet of an F-logic language consists of a set of function symbols and set of variables. A term is a usual first-order term composed of function symbols and variables, as in predicate calculus. A language of F-logic consists of a set of formulae constructed of the alphabet symbols. As in many other logics, formulas are built from simpler formulas by using the usual connectives *not*, *and*, and *or* and the quantifiers *forall* and *exists*. The simplest kind of formulas are called molecular F-formulas. A molecule in F-logic is one of the following statements:

- Assertion of the form *C* :: *D* or of the form *O* : *C*, where *C*, *D*, and *O* are id-terms. The first expression models subclass relationship and the second statement models is-element-of relationship.
- An object molecule of the form *O*[*a* ';'-separated list of method expressions]. A method expression can be either a data expression or a signature expression. *O* is a term denoting an object (which may refer to an instance or a class). “*a*” further specifies properties of this object.
  - Data expressions take one of the following two forms:
    A *scalar* expression *ScalarMethod* @Q_1,...,Q_k -> *T*
    A *set-valued* expression *SetMethod* @ R_1,...,R_l ->> {S_1,...,S_m}
    Data expressions specify that the method *m* applied to the object *O* and the parameter Q_1,...,Q_k deliver the value *T.* The can be either single-valued or may return a set.
  - Signature expressions also take two forms:
    A *scalar* signature expression *ScalarMethod* @V_1,...,V_n => (A_1,...,A_r)

```
class relation (?relation)
class function (?function)
class class (?class)
relation instance-of (?individual ?class)
function all-instances (?class) :->
    ?set-of-instances
function one-of (@instances) :-> ?class
relation subclass-of (?child-class ?parent-class)
relation superclass-of (?parent-class ?child-
class)
relation subrelation-of
    (?child-relation ?parent-relation)
relation direct-instance-of (?individual ?class)
relation direct-subclass-of
    (?child-class ?parent-class)
function arity (?relation) :-> ?n
function exact-domain (?relation) :->
    ?domain-relation
function exact-range (?relation) :-> ?class
relation total-on (?relation ?domain-relation)
relation onto (?relation ?range-class)
class n-ary-relation (?relation)
class unary-relation (?relation)
class binary-relation (?relation)
class unary-function (?function)
relation single-valued (?binary-relation)
function inverse (?binary-relation) :-> ?relation
function projection (?relation ?column)
    :-> ?class
function composition
    (?relation-1 ?relation-2) :-> ?binary-relation
relation composition-of
    (?binary-relation ?list-of-relations)
function compose
    @binary-relations) :-> ?binary-relation
relation alias (?relation-1 ?relation-2)
relation domain (?relation ?class)
relation domain-of
    (?domain-class ?binary-relation)
relation range (?relation ?class)
relation range-of (?class ?relation)
relation nth-domain
    (?relation ?integer ?domain-class)
relation has-value
    (?domain-instance ?binary-relation ?value)
function all-values (?domain-instance
    ?binary-relation) :-> ?set-of-values
relation value-type
    (?domain-instance ?binary-relation ?class)
function value-cardinality
    (?domain-instance ?binary-relation) :-> ?n
relation same-values
    (?domain-instance ?relation-1 ?relation-2)
relation inherited-slot-value
    (?domain-class ?binary-relation ?value)
function all-inherited-slot-values
    (?domain-class ?binary-relation)
    :-> ?set-of-values
relation slot-value-type (?domain-class
    ?binary-relation ?range-class)
function slot-cardinality
    (?domain-class ?binary-relation) :-> ?n
relation minimum-slot-cardinality
    (?domain-class ?binary-relation ?n)
relation maximum-slot-cardinality
    (?domain-class ?binary-relation ?n)
relation single-valued-slot
    (?domain-class ?binary-relation)
relation same-slot-values
    (?domain-class ?relation-1 ?relation-2)
class class-partition (?set-of-classes)
relation subclass-partition (?c ?class-partition)
relation exhaustive-subclass-partition
    (?c ?class-partition)
relation asymmetric-relation (?binary-relation)
relation antisymmetric-relation
    (?binary-relation)
relation antireflexive-relation (?binary-relation)
relation irreflexive-relation (?binary-relation)
relation reflexive-relation (?binary-relation)
relation symmetric-relation (?binary-relation)
relation transitive-relation (?binary-relation)
relation weak-transitive-relation
    (?binary-relation)
relation one-to-one-relation (?binary-relation)
relation many-to-one-relation (?binary-relation)
relation one-to-many-relation (?binary-relation)
relation many-to-many-relation
    (?binary-relation)
relation equivalence-relation (?binary-relation)
relation partial-order-relation (?binary-relation)
relation total-order-relation (?binary-relation)
relation documentation (?object ?string)
```

Fig. 1 The Frame Ontology of Ontolingua (see [Grub 1993]).

A *set-valued* signature expression *SetMethod @* $W_1,...,W_s$ =>> ($B_1,...,B_t$)
Signature expressions define types for applying methods (i.e., attributes) to object. A method *m* applied to the object *O* and the parameter $V_1,..,V_m$ must deliver a value that is element/subclass of $A_1,...,A_r$.

F-formulae are built of simpler F-formulae in the usual manner by means of logical connectives and quantifiers.

Ontolingua and Frame logic integrate frames (i.e., classes) into a logical framework. The main difference between Ontolingua and Frame logic is the manner in which they realize frame-based modeling primitives in a logical language. Ontolingua characterizes the frame-based modeling primitives via axioms in the language. Frame logic defines their semantics externally via an explicit definition of their semantics. Simplified: Ontolingua applies standard semantics of predicate logic and uses axioms in this logic to exclude models that would not fit to the semantics of its modeling primitives. Frame logic provides a more complex semantics compared to predicate logic. The modeling primitives are explicitly defined in the semantics of Frame logic. A second difference between Frame logic and Ontolingua arises from the fact that Ontolingua inherits the powerful reification mechanism from KIF which allows the use of formulas as terms of (meta-level) formulas. In Frame logic, predicate names can be bound to variables but not entire formulas.

2.3 Description Logics

The main effort of research in knowledge representation is directed at providing theories and systems for expressing structured knowledge and for accessing and reasoning with it in a principled way. *Description Logics (DL)* [Brachman & Schmolze, 1985], also known as *terminological logics*, form an important powerful class of logic-based knowledge representation languages.[6] They result from early work in semantic networks and defined a formal and operational semantics for them. Description Logics try to find a fragment of first-order logic with high expressive power which has still a decidable and efficient inference procedure (cf. [Nebe 1996]). Implemented systems are, for example, BACK, CLASSIC, CRACK, FLEX, K-REP, KL-ONE, KRIS, LOOM, and YAK.[7]

A distinguishing feature of Description logics is that classes (usually called concepts) can be defined intensionally in terms of descriptions that specify the properties that objects must satisfy to belong to the concept. These descriptions are expressed using a language that allows the construction of composite descriptions, including restrictions on the binary relationships (usually called roles) connecting objects.

Figure 2 provide the syntax definition of the core language of CLASSIC. Its main modeling primitives are concept expressions and individual expressions (cf. [BBMR 1989]). A CLASSIC database is mostly a repository of information about individual objects. Objects have an intrinsic identity and are related to each other through binary relationships; these are called *roles* (elsewhere known as attributes or properties). Individuals will be grouped into collections indirectly by means of descriptions that apply to all members of a collection |these descriptions we shall call concepts or classes. The data definition language allows the definition of concepts either by grouping individuals together extensionally or grouping individuals implicitly, through the use of intensional descriptions in regard to their structure. Complex CLASSIC concepts are formed by composing expressions using a small set of constructors.

The simplest kind of description you can form in CLASSIC is a **primitive concept**. Primitive concepts are simple, but not necessarily atomic; each primitive concept, except for the topmost concept (which we call THING), is expected to have at least one parent (more general) concept. The simplest kind of primitive is one whose only parent is essentially vacuous, namely THING. For example, the concept of a CAR might be defined in this way:

6. Links to most papers, project, and research events in this area can be found at http://dl.kr.org/.
7. http://www.research.att.com/sw/tools/classic/imp-systems.html

```
<concept-expr> ::=
    THING | CLASSIC-THING | HOST-THING |
    [these three are built-in primitives]
    <concept-name> |
    ( AND <concept-expr> +)¹ |
    ( ALL <role-expr> <concept-expr>) |
    ( AT-LEAST <positive-integer> <role-expr>) |
    ( AT-LEAST <positive-integer> <role-expr>) |
    ( AT-MOST <non-negative-integer><role-expr>) |
    ( SAME-AS (<role-expr> +) (<role-expr>+)) |
    ( TEST <fn> <realm>) |
    ( ONE-OF <individual-name>+) |
    ( PRIMITIVE <concept-expr> <index>) |
    ( DISJOINT-PRIMITIVE
        <concept-expr> <partition-index> <index>)
<individual-expr> ::=
    <concept-expr> |
    ( FILLS <role-expr> <individual-name>) |
    ( CLOSE <role-expr>) |
    ( AND <individual-expr>+)
<realm> ::= host | classic
<concept-name> ::= <symbol>
<individual-name> ::= <symbol> | <host-lang-expr>
<role-expr> ::= <symbol>
<index> ::= <number> | <symbol>
<partition-index> ::= <number> | <symbol>
<fn> ::= a unary function with boolean return type that can be evaluated in the host language.
```

1. + means one or more values separated by blanks.

Fig. 2 The grammar of the CLASSIC language (taken from [BBMR 1989]).

(PRIMITIVE THING car)[8]

This expression means that whatever it designates is simply a type of THING with some unspecified difference from THING in general. This is quite the opposite of the case with other, non-primitive concepts, as we shall see in a moment.

Primitives can also have non-trivial parents. Thus, SPORTS-CAR might be defined as a subconcept of both CAR and another concept, EXPENSIVE-THING:

(PRIMITIVE (AND CAR EXPENSIVE-THING)
sports-car).

In fact, the parent of a primitive concept can be any CLASSIC concept, including another primitive. Primitives thus specify *necessary* conditions: if *Corvette*$_1$ is an instance of SPORTS-CAR, then it is both a CAR and an EXPENSIVE-THING. But note that there is no sufficiency condition specified for primitive concepts.

The CLASSIC language of concepts allows us to go substantially beyond the simple IS-A hierarchies of more traditional semantic data models. It offers three special ways of describing objects in terms of their structure. As we shall see, these constructors let some class membership relations be determined by inference. CLASSIC's three complex constructors are role *value restrictions*, *cardinality bounds*, and *co-reference constraints*.

8. The example is taken from [BBMR 1989].

Role value restrictions are type constraints that hold for the fillers for some single role.

Value restriction. For example, the concept expression

(ALL thing-driven CAR)

describes any object that is related by the thing-driven role solely to individuals describable by the concept CAR.

Bounds restrict the number of fillers for roles. For example,

(AT-MOST 4 thing-driven)

describes any object that is related to at most 4 distinct individuals through the thing-driven role.

(AT-LEAST 3 wheel)

describes any object that is related to at least 3 distinct individuals through the wheel role.

Co-reference constraints specify simple equalities between single-valued roles or, more generally, chains of such roles. For example, the expression

(SAME-AS (driver) (insurance payer))

describes all those individuals whose filler for the driver role is the same as the payer of their insurance role.

Each of the constructors acts as part of both necessary and sufficient conditions for concepts in which they appear (as long as they are not used in a primitive concept, in which case there are no sufficient conditions).

It is important to note that the meaning of concepts in CLASSIC is determined by their structure. This implies that certain relationships exist between concepts by virtue of their definition. For example, it is quite possible for several different concept expressions to denote the same class:

(AND (ALL thing-driven CAR)
(ALL thing-driven EXPENSIVE-THING))

is the same concept as

(ALL thing-driven
(AND CAR EXPENSIVE-THING)),

Various studies have examined extensions of the expressive power of such a language and the trade-off in computational complexity for deriving is-a relationship between concepts and individuals in such a logic. Efficient implementations for core sets of primitives in these languages have been developed in the meantime (cf. [BoPa 1994], [MacG 1994], and [HoPa 1999]), see for example DLP[9] and the FaCT system[10].

3 XML, RDF, and Ontology Languages

In the following we will examine how XML[11] and RDF[12] can be used to express

9. http://www.bell-labs.com/user/pfps/
10. http://www.cs.man.ac.uk/~horrocks/software.html.
11. http://www.w3.org/XML/
12. http://www.w3c.org/Metadata/

ontologies. XML is a tag-based language for describing tree structures with a linear syntax. It is a successor of SGML, which was developed long ago for describing document structures. XML provides semantic information as a by-product of defining the structure of the document. It prescribes a tree structure for documents and the different leaves of the tree have well-defined tags and contexts in which the information can be understood. That is, structure and semantics of document are interwoven. The *Resource Description Framework (RDF)*[13] (cf. [Mill 1998], [LaSw 1999]) provides a means for adding semantics to a document without making any assumptions about the structure of the document. RDF is an infrastructure that enables the encoding, exchange and reuse of structured meta data. Search engines, intelligent agents, information broker, browsers and human user can make use of *semantic* information. RDF is an XML application (i.e., its syntax is defined in XML) customized for adding meta information to Web documents. It is currently under development as a W3C standard for content descriptions of Web sources and will be used by other standards such as PICS-2, P3P, and DigSig.

3.1 DTD and Ontologies

The closest thing that XML offers for ontological modeling is the Document Type Definition (DTD) which defines the legal nestings of tags and introduces attributes for them. Defining tags, their nesting, and attributes for tags may be seen as defining an ontology. However, there are significant differences between an ontology and an DTD.

- First, a DTD specifies the legal *lexical* nesting in a document, which may or may not coincide with an *ontological* hierarchy (subclass relationship). That is, there is nothing in a DTD that corresponds to the is-a relationship of classes that is usually central in an ontology.
- Second, and in consequence, DTDs lack any notion of inheritance. In an ontology, subclasses inherit attributes defined for their super classes and super classes inherit instances defined for their subclasses. Both inheritance mechanisms do not exist for DTDs.
- Third, DTDs provide a rather poor means for defining the semantics of elementary tags. Basically, a tag can be defined as being composed of other tags or being a string. Usually, ontologies provide a much richer typing concept for describing elementary types.
- Fourth, DTDs define the order in which tags appear in a document. For ontologies, in contrast, the ordering of attribute descriptions does not matter.

We will use an example to clarify these differences (see Figure 3).

- Concept c_1 has two attributes, a_1 and a_2.
  This implies that the domains of a_1 and a_2 are the elements of c_1. The range of a_1 is the intersection of c_2 and c_3 and the range of a_2 is the union of c_4 and c_5.
- c_2 is defined as a subclass of c_1.
  This implies that all attributes defined for c_1 are also applicable for c_2. In addition, each element of c_2 is also an element of c_1.
- Finally c_6 is a subclass of c_4 and c_1.
  Therefore it inherits the attributes a_1 and a_2 from c_1.
  In addition, it refines the range restriction of attribute a_1 to c_2 and c_3 and c_4. That is, the value of a_1 applied to an element of c_6 must also be an element of c_4. This is not

13. http://www.w3c.org/Metadata/

necessary for an element of c_1 that is not also an element of c_6.

When translating this ontology into a DTD we first define c_1 as an element having two subtags $c_1.a_1$ and $c_1.a_2$, i.e.,

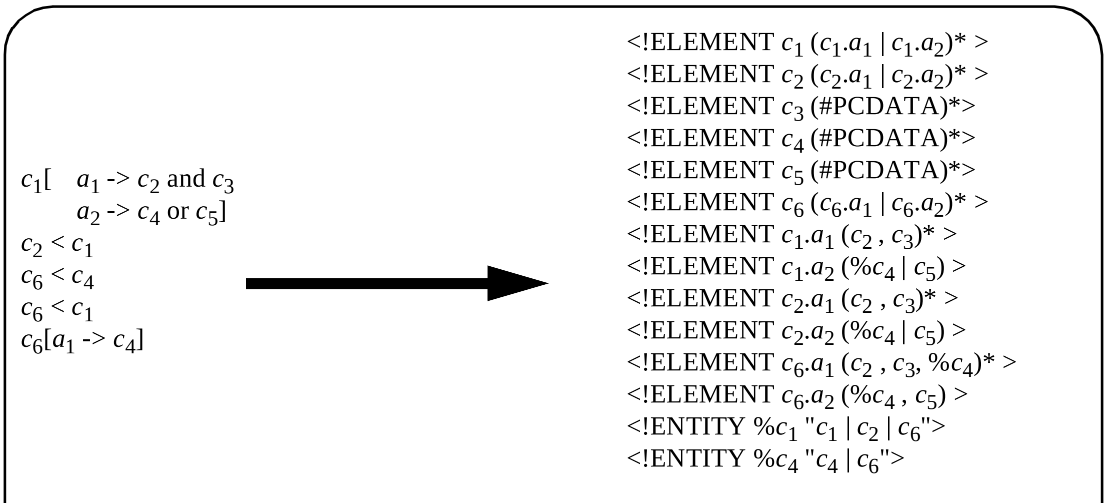
$<c_1>$
 $<c_1.a_1>...</c_1.a_1>$
 $<c_2.a_2>...</c_1.a_2>$
$</c_1>$

would be a valid document. Therefore, we reify the attribute names with the concept names to distinguish different appearances of attributes in various concepts. A number of problems arise in this translation process:

- The sequence of attribute values of an object does not matter in an ontology, i.e.
  $o[a_1=5, a_2=3]$ and $o[a_2=3, a_1=5]$ are equivalent.
  We express this by $(a_1 \mid a_2)$*. However, this implies that an object may have several values for the same attribute (which is allowed for set-valued but not for single-valued attributes).
- The attribute a_1 has for c_1 as range the intersection of c_2 and c_3. That is, a value of the attribute is an object for which the attributes of c_2 and c_3 can be applied. We express this via (c_2 , c_3)* which again imply that an object may have several values for the same attribute.
- The only primitive data type is PCDATA, i.e. arbitrary strings.

We can also see the two aspects of inheritance in the translation process

- First, we have to add all inherited attributes and their inherited range restrictions explicitly. For example:
  <!**ELEMENT** $c_6.a_1$ (c_2*, c_3*, %c_4*)* >
- Second, the value of an attribute may also be the element of a subclass of its value type (i.e., a super class inherits all elements of its subclasses). Therefore, whenever a class is used as a range restriction we have to add all its subclasses. For this we use the entity mechanism of DTDs. For example,
  <!**ENTITY** %c_1 “$c_1 \mid c_2 \mid c_6$“>



$c_1[\ \ a_1$ -> c_2 and c_3
$\quad a_2$ -> c_4 or c_5]
$c_2 < c_1$
$c_6 < c_4$
$c_6 < c_1$
$c_6[a_1$ -> c_4]

<!ELEMENT c_1 ($c_1.a_1 \mid c_1.a_2$)* >
<!ELEMENT c_2 ($c_2.a_1 \mid c_2.a_2$)* >
<!ELEMENT c_3 (#PCDATA)*>
<!ELEMENT c_4 (#PCDATA)*>
<!ELEMENT c_5 (#PCDATA)*>
<!ELEMENT c_6 ($c_6.a_1 \mid c_6.a_2$)* >
<!ELEMENT $c_1.a_1$ (c_2 , c_3)* >
<!ELEMENT $c_1.a_2$ (%$c_4 \mid c_5$) >
<!ELEMENT $c_2.a_1$ (c_2 , c_3)* >
<!ELEMENT $c_2.a_2$ (%$c_4 \mid c_5$) >
<!ELEMENT $c_6.a_1$ (c_2 , c_3, %c_4)* >
<!ELEMENT $c_6.a_2$ (%c_4 , c_5) >
<!ENTITY %c_1 "$c_1 \mid c_2 \mid c_6$">
<!ENTITY %c_4 "$c_4 \mid c_6$">

Fig. 3 Translation from an ontology into a DTD.

More details and further aspects of ontology to DTD translations can be found in [ErSt 1999], [RDC 1999], and [HaFe, subm.]. In a nutshell, DTDs are rather weak in regard to what can be expressed with them.

Work on *XML-schemes* may well contribute to bridging the gap between DTD's and ontologies. Schemes will be introducing: mechanisms for constraining document structure and content, mechanisms to enable inheritance for element, attribute, and data type definitions, mechanisms for application-specific constraints and descriptions, mechanisms to enable the integration of structural schemes with primitive data types, primitive data typing, including byte, date, integer, sequence, etc., and they shall allow creation of user-defined data types.

Up to now we have discussed the mapping from ontologies to DTDs. [WeId 1999] discuss a mapping from DTDs to an ontological representation. [WeId 1999] want to provide the reasoning service of description logic to query and manipulate XML documents. DTDs are therefore translated automatically into a representation of an ontology in description logic. This ontology simply consists of each element in the DTD. The taxonomy can be derived by the classifier of the description logic CLASSIC based on the use of entities and the type attributes.

3.2 RDF and Ontologies

RDFS can be used directly to describe an ontology. *Objects*, *Classes*, and *Properties* can be described. Predefined properties can be used to model *instance of* and *subclass of* relationships as well as *domain restrictions* and *range restrictions* of attributes. A speciality of RDFS is that properties are defined globally and are not encapsulated as attributes in class definitions. Therefore, a frame or object-oriented ontology can only be expressed in RDFS by reifying the property names with class name suffixes (as we already saw for XML). In regard to ontologies RDF provides two important contributions:

- a standardized syntax for writing ontologies,
- a standard set of modeling primitives like *instance of* and *subclass of* relationships.

On the one hand, RDFS provides rather limited expressive power. A serious weakness of RDF is that it lacks a standard for describing logical axioms. RDFS allows the definition of classes and properties through their types (by providing their names). No intensional definitions or complex relationships via axioms can be defined. On the other hand, RDFS provides a rather strong reification mechanism. RDF expressions can be used is terms in meta expressions. Here, RDFS provides reified second-order logic like it is used in CycL and KIF. Neither Frame logic nor most Description Logics provide such an expressivity[14].

3.3 Comparing RDF and XML

RDF is an application of XML for the purpose of representing meta data. For example, the RDF statements

date(http://www.xyz.de/Example/Smith/) = July 1999
subject(http://www.xyz.de/Example/Smith/) = Intelligent Agents

14. Exceptions are described in [CDL 1995] and [DeLe 1995], however, without an implemented reasoning service.

creator(http://www.xyz.de/Example/Smith/) = http://www.xyz.de/~smith/
name(http://www.xyz.de/~smith/) = John Smith
email(http://www.xyz.de/~smith/) = smith@organisation.de

can be represented in linear XML syntax (see Fig. 4). This may raise the question why there is a need for RDF at all because all meta data represented in RDF can also be presented in XML. However, RDF provides a *standard* way to represent meta data in XML. Directly using plain XML for representing meta data would result in being represented in various ways.

The difference becomes even more obvious when considering how to represent an ontology in RDF or XML. Above we showed how an ontology can be used to generate a DTD describing the structure of XML documents. However, we did not discuss how the ontology itself could be represented in XML. For defining a standardized manner in which ontologies can be represented in XML we have to decide about two questions:

- What are the epistemological primitives used to represent an ontology (i.e., things like classes, is-a relationship, element-of relationship, attributes, domain and range restrictions etc.). Basically these are decisions about the *meta*-ontology used to represent ontologies.
- How these concepts can be represented in the linear syntax of XML.

There are a number of different possibilities, and this makes clear how RDFS comes into the story. RDFS provides a fixed set of modeling primitives for defining an ontology (classes, resources, properties, is-a and element-of relationships, etc.) and a standard way to encode them in XML. Using XML directly for the purpose of representing ontologies would require us to duplicate this standardization effort.[15]

4 XOL

The BioOntology Core Group[16] recommends the use of a frame-based language with

```
<? xml version=1.0”?>
<RDF
    xmlns=”http://www.w3c.org/1999/02/22-rdf-syntax-ns#”
    xmlns:DC=”http://www.purl.org/DC#/”
    xmlns:y=”http://www.description.org/schema”>

    <Description about=”http://www.xyz.de/Example/Smith/”>
        <DC:date rdf:resource=”July 1999”/>
        <DC:subject rdf:resource=”Intelligent Agents”/>
        <DC:creator rdf:resource=”http://www.xyz.de/~smith/”/>
    </Description>

    <Description about=”http://www.xyz.de/~smith/”>
        <DC:name rdf:resource=”John Smith”/>
        <DC:email rdf:resource=”smith@organisation.de”/>
    </Description>

</RDF>
```

Fig. 4 XML representation of RDF statements.

15. However, it would also allow us to part with some of the unusual design decisions of RDF.

an XML syntax for the exchange of ontologies for molecular biology. The proposed language is called XOL[17] (cf. [KCT 1999], [MKA+ 1999]). The ontology definitions that XOL is designed to encode include both schema information (meta-data), such as class definitions from object databases – as well as non-schema information (ground facts) such as object definitions from object databases.

The syntax of XOL is based on XML. The modeling primitives and semantics of XOL are based on OKBC-Lite, which is a simplified form of the knowledge model for the *Open Knowledge Base Connectivity (OKBC)*[18] [CFF+ 1997]. OKBC is an application program interface for accessing frame knowledge representation systems. Its knowledge model supports features most commonly found in knowledge representation systems, object databases, and relational databases. OKBC-Lite extracts most of the essential features of OKBC, but omits some of its more complex aspects. XOL was inspired by Ontolingua. XOL differs from Ontolingua, however, as it has an XML-based syntax rather than a Lisp-based syntax. Still, the semantics of OKBC-Lite which underly XOL are extremely similar to the semantics of Ontolingua.

The design of XOL deliberately uses a generic approach to define ontologies, meaning that the single set of XML tags (defined by a single XML DTD) defined for XOL can describe any and every ontology. This approach contrasts with the approaches taken by other XML schema languages, in which a generic set of tags is typically used to define the schema portion of the ontology and the schema itself is used to generate a second set of application-specific tags (and an application-specific DTD), which in turn are used to encode a separate XML file that contains the data portion of the ontology. Compare the XOL definitions of Figure 5. All of the XML elements of this specification (meaning all the words inside brackets), such as *class*, *individual*, and *name* are generic, i.e., they pertain to all ontologies. All of the ontology-specific information is in the text portion of the XML file, that is between the pairs of elements. In contrast, approaches discussed

```
<class>
    <name>person</name>
</class>

<slot>
    <name>age</name>
    <domain>person</domain>
    <value-type>integer</value-type>
    <numeric-max>150</numeric-max>
</slot>

<individual>
    <name>fred</name>
    <type>person</type>
    <slot-values>
    <name>age</name>
    <value>35</value>
    </slot-values>
</individual>
```

Fig. 5 An example in XOL (taken from [KCT 1999]).

16. http://smi-web.stanford.edu/projects/bio-ontology/
17. http://www.ontologos.org/Ontology/XOL.htm.
18. http://www.ai.sri.com/~okbc/. OKBC has also been chosen by FIPA as exchange standard for ontologies, see http://www.fipa.org, FIPA 98 Specification, Part 12 Ontology Service (cf. [FIPA 98]).

```
<person>
    <name>fred</name>
    <age>35</age>
</person>
```

Fig. 6 Nonreusable Ontology Specification.

earlier in this paper might use this type of XML markup to define the individual Fred as shown in Figure 6.

The primary advantage of this approach is genericity and reusability. Only one XML DTD need be defined to describe any and every ontology. Using the non-generic approach, every ontology must define a second, ontology-specific, DTD for describing the data elements of the ontology. Otherwise, another application cannot process the ontology. XOL appears rather promising because it provides ontological modeling primitives expressed in one of the most important information exchange standards: XML.

5 Conclusions

XOL shows how many aspects of an ontology can be represented in XML. However, important aspects which are lacking are rules and constraints (i.e., class definitions). They are often a significant part of the knowledge provided by an ontology, and XOL does not provide support for their representation. Besides XOL, RDF provide a proposal for specifying ontologies. However, many design decisions in RDF are rather particular. Properties are defined globally, violating all modeling experiences from object-oriented and frame-based approaches that define attributes locally for classes. RDF provides powerful reification, i.e. expressions can become terms in meta-level expressions. This makes reasoning service for RDF rather tricky. Most description logics do not provide such a service. Finally, RDF also does not provide a means for defining constraints and rules which allow us to specify information intensionally. Currently, there are initiatives to define a proper *Ontology Interchange Language (OIL)* that overcomes these shortcomings.

6 References

[BBMR 1989] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick: CLASSIC: A Structural Data Model for Objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Mangement of Data*, pp. 59-67, 1989.

[BoPa 1994] A. Borgida and P. F. Patel-Schneider: A Semenatics and Complete Algorithm for Subsumption in the CLASSIC Description Logic, *Journal of Artificial Intelligence Reserach*, 1:277-308, 1994.

[Brachman & Schmolze, 1985] R. Brachman and J. Schmolze: An overview of the KL-ONE Knowledge Representation System, *Cognitive Science*, 9(2):171-216, 1985.

[CDL 1995] D. Calvanese, G. De Giacomo, and M. Lenzerini: Structured objects: Modeling and resoning. In *Proceedings of the Fourth International Conference on Deductive and Object-Oriented Databases (DOOD-95),* Lecture Notes in Computer Science (LNCS), Springer, 1995.

[CFF+ 1997] V. K. Chaudhri, A. Farquhar, R. Fikes, P. D. Karp, and J. P. Rice: *Open*

Knowledge Base Connectivity 2.0., Technical Report KSL-98-06, Knowledge Systems Laboratory, Stanford., July 1997.

[CMB 1998] A. M. de Carvalho Moura, M. L. Machado Campos, and C. M. Barreto: A survey on metadata for describing and retrieving Internet resources, *World Wide Web*, 1:221-240, 1998.

[DeLe 1995] G. De Giacomo and M. Lenzerini: What's in an Aggregate: Foundations for Description Logics with Tupels and Sets. In *Proceedings of the 14th International Conference on Artificial Intelligence (IJCAI-95)*, Montreal, Canada, 1995.

[ErSt 1999] M. Erdmann and R. Studer: *Ontologies as Conceptual Models for XML Documents*, research report, Institute AIFB, University of Karlsruhe, 1999.

[FFR 1997] A. Farquhar, R. Fikes, and J. Rice, The Ontolingua Server: A Tool for Collaborative Ontology Construction, *International Journal of Human-Computer Studies*, 46:707-728, 1997.

[FIPA 98] Foundation for Intelligent Physical Agents (FIPA): FIPA 98 Specification, Part 12, Ontology Service, October 1998. http://www.fipa.org.

[Gene 1991] M. R. Genesereth: Knowledge Interchange Format. In *Proceedings of the Second International Conference on the Principles of Knowledge Representation and Reasoning (KR-91)*, J. Allenet al., (eds), Morgan Kaufman Publishers, 1991, pp 238-249. See also http://logic.stanford.edu/kif/kif.html.

[GeFi 1992] M.R. Genesereth and R.E. Fikes: Knowledge Interchange Format, Version 3.0, Reference Manual. Technical Report, Logic-92-1, Computer Science Dept., Stanford University, 1992. http://www.cs.umbc.edu/kse/.

[Grub 1993] T. R. Gruber: A Translation Approach to Portable Ontology Specifications, *Knowledge Acquisition*, 5:199—220, 1993.

[Grub 1995] T.R. Gruber: Towards Principles for the Design of Ontologies used for Knowledge Sharing, *International Journal of Human-Computer Studies*, 43:907-928, 1995.

[Guha 1993] R. V. Guha: *Context Dependence of Representations in Cyc*, MCC Technical Report, CYC 066-93, 1993.

[GLM+ 1998] R. V. Guha, O. Lassila, E. Miller and D. Brickley: Enabling Inferencing. In [QL 1998].

[HaFe, subm.] F. van Harmelen & D. Fensel: Surveying notations for machine-processable semantics of Web sources, submitted.

[HoPa 1999] I. Horrocks and P. F. Patel-Schneider: Optimising Description Logic Subsumption, *Journal of Logic and Computation*, 9(3):267-293, 1999.

[KCT 1999] P. D. Karp, V. K. Chaudhri, and J. Thomere: XOL: An XML-Based Ontology Exchange Language, Version 0.3, July 3, 1999. ftp://smi.stanford.edu/pub/bio-ontology/OntologyExchange.doc.

[KLW 1995] M. Kifer, G. Lausen, and J. Wu: Logical Foundations of Object-Oriented and Frame-Based Languages, *Journal of the ACM*, 42, 1995.

[LaSw 1999] O. Lassila and R. Swick. Resource Description Framework (RDF). W3C proposed Recommendation, January 1999. http://www.w3c.org/TR/WD-rdf-syntax.

[Lena, subm.] D. B. Lenat: The Dimensions of Context Space, submitted. http://casbah.org/resources/cycContextSpace.shtml.

[LeGu 1990] D. B. Lenat and R. V. Guha: *Building large knowledge-based systems. Representation and inference in the Cyc project*, Addison-Wesley, Reading, Massachusetts, 1990.

[MacG 1994] R. MacGregor: A Description Classifier for the Predicate Calculus. In *Proceedings of the 12th National Conference on AI (AAAI-94)*, pp. 213-220, 1994.

[MaMa 1999] A. Malhotra and M. Maloney: XML-Schema Requirements. W3C Note, February 1999. http://www.w3.org/TR/NOTE-xml-schema-req.

[McCa 1993] J. McCarthy: Notes on Formalizing Context. In *Proceedings of the 13th International Conference on Artificial Intelligence (IJCAI-93)*, Chambery, France, 1993.

[MKA+ 1999] R. McEntire, P. Karp, N. Abernethy, F. Olken, R. E. Kent, M. DeJongh, P. Tarczy-Hornoch, D. Benton, D. Pathak, G. Helt, S. Lewis, A. Kosky, E. Neumann, D. Hodnett, L. Tolda, and T. Topaloglou: *An Evaluation of Ontology Exchange Languages for Bioinformatics*. August 1999, ftp://smi.stanford.edu/pub/bio-ontology/OntologyExchange.doc.

[Mill 1998] E. Miller: An Introduction to the Resource Description Framework, *D-Lib Magazine*, May 1998.

[Nebe 1996] B. Nebel: Artificial Intelligence: A Computational Perspective. In G. Brewka (ed.), *Principles of Knowledge Representation*, CSLI publications, Studies in Logic, Language and Information, Stanford, 1996.

[QL 1998] *Proceedings of W3C Query Language Workshop (QL'98) - The Query Languages Workshop*, Boston, Massachussets, December 3-4, 1998. http://www.w3.org/TandS/QL/QL98/.

[RDC 1999] A. Rabarijoana, R. Dieng, and O. Corby: Exploitation of XML for Corporative Knowledge Management. In D. Fensel and R. Studer (eds.), *Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Workshop (EKAW-99)*, Lecture Notes in Artificial Intelligence, LNAI 1621, Springer-Verlag, 1999.

[WeId 1999] C. Welty and N. Ide: Using the Right Tools: Enhancing retrieval From Marked-Up Documents, *Computers and the Humanities*, 33:1-2, Special Issue on the Tenth Anniversary of the Text Encoding Initiative, 1999. http://www.cs.vassar.edu/faculty/welty/papers/.