Having defined the Semantic Web infrastructure, which enables the creation of a web of data, two aspects remain to be seen. The first one concerns the rich semantics that were announced as a part of the Semantic Web vision: how can the conceptual knowledge useful for a range of applications be successfully ported to and exploited on the Semantic Web? The second aspect concerns the access to Semantic Web data: how can one query and successfully find the information that is represented on these large RDF graphs that constitute the Semantic Web information sphere?

The goal of this chapter is to make the reader familiar with relevant languages that address these two crucial matters: representing conceptual knowledge and querying RDF data. With respect to conceptual knowledge, there exist very expressive languages like OWL, which allow to formally specify the ontologies that guide the creation of RDF data and make it amenable to automated reasoning processes. This will be the object of Section 7.2. However, we will see in section 7.3 that OWL does not meet all conceptual representation requirements.
In particular, its high level of formal precision is not appropriate for modeling a wide range of more lightweight vocabularies, where relations between concepts are not completely sharp, and which are not expected to guide the structure of RDF descriptions. Hence we will introduce the Simple Knowledge Organization System SKOS in section 7.3, which is suitable for modeling such lightweight ontologies.

In Section 7.4 we will introduce SPARQL, the recently standardized Semantic Web Query language, with an emphasis on aspects relevant to querying multimedia metadata in the running examples of COMM annotations. As a practical guide, we will reference relevant implementations as well as list limitations of SPARQL and give an outlook of possible future extensions to address these limitations.

7.1 The need for ontologies on the Semantic Web

As seen in the previous chapter, the Semantic Web requires defining of vocabularies for the creation of useful RDF data. Consider the XML realm: there, DTDs or XML Schemas are crucial, as they allow for specifying which XML elements and attributes may appear in XML documents, and how these entities should be organized (Bray et al. 2006; Fallside and Walmsley 2004). RDF provides a more flexible way to represent knowledge. However, it is still by essence a mere description framework, that requires some form of controlled vocabularies to adequately express and share information. For example, to represent information about (audiovisual) multimedia documents, one would expect building blocks like “program”, “sequence” and “scene” to identify documents and their parts, and attributes such as “genre”, “creator” or “subject” to characterize them.

In the Semantic Web context, such vocabularies are called ontologies. An ontology should provide the conceptual elements required to describe entities such as encountered in a specific domain, or manipulated by a given application. An essential point, related to the intended semantic nature of the Semantic Web, is that ontologies are not restricted to the gathering of (named) building blocks for RDF graphs. They should also provide appropriate semantics for such building blocks. This implies of course that these vocabularies should emerge from careful processes, capturing and reflecting consensus in specific fields of human knowledge. But, beyond this, ontologies should be also exploitable by automated reasoning processes. A typical means for this, inherited from Artificial Intelligence research, is to rely on formalisms which allow for definitions suitable for inference. For example, a segment of an image again is an image and each image has at least one creator.

To achieve such goals, an appropriate ontology language is necessary. In this section we will focus on explaining the details of one specific ontology language, namely OWL, the ontology language developed by W3C.

7.2 Representing ontological knowledge using OWL

RDF Schema, which was introduced in the previous chapter, is already a simple ontology language. Using it, we can introduce classes such as Document and Person, and define specialisation relations (e.g. Author is a sub-class of Person), state that authorOf is a property, which has domain Document and range Author. We can then assert in RDF such a language can actually be called a meta-language, since it enables in practice to design application-specific description languages.
facts such as that Aristotle is an instance of Author who is the author of the document physics.

However, such specification may be insufficient for a whole range of semantic-intensive use cases (Heflin 2004). The W3C has therefore created the OWL ontology language, which extends the RDFS language by adding more operators, following the requirements and design principles that could be elicited from these observed use cases (Bechhofer et al. 2004).

7.2.1 OWL constructs and OWL syntax

OWL is defined as an extension of RDFS, aimed at providing vocabularies to create RDF data. It is thus heavily influenced by the RDF data model, which is based on graphs composed of nodes and directed arcs. As RDFS, OWL distinguishes two main types of ontological resources:

- **Classes**, that are used to define the types of a domain’s entities (entities which are then called the instances of these classes);
- **Properties**, that are used to relate entities together.

As a matter of fact, the designers of OWL have opted for re-using the constructs introduced by RDFS to basically characterize these entities, like rdfs:subClassOf, rdfs:subPropertyOf, rdfs:domain and rdfs:range. OWL however introduces new constructs to specify classes and properties with richer axioms.

For instance, one can specify more advanced semantic characteristics of properties, e.g. that hasAuthor and authorOf are the inverse of each other, that hasISBN is an inverse-functional property (i.e. that an object has only one possible ISBN value which identifies it) or that hasPart is transitive (every part of a part of a given object is a part of this object).

OWL also enables much richer specifications for classes. It allows to define classes in a complete way, or to give at least necessary conditions for individuals to belong to a class. In the first case we can, for example, define a Publication as the class equivalent to the class of things having an ISBN. The second case is weaker, we can, for example, define as a necessary condition for something to be a Journal that it must be a publication, formally SubClassOf(Journal Publication). This, however, does not imply that every publication must also be a journal.

Such definitions may involve simple class individuals or more complex expressions that OWL allows:

- **combinations** of classes using boolean constructs, such as union, intersection or complement;
- **local restrictions** on properties. This is an extension of the range and domain restrictions of RDFS. With the construct owl:allValuesFrom one can, for example, enforce that Books have to be written by Persons only. Formally, this would be written as SubClassOf(Book, restriction(hasAuthor allValuesFrom(Person))).

In fact, OWL proposes a whole range of expressions that enable handling semantic similarity, as well as basic identity and distinction statements. As seen, one can define two classes to be equivalent. It is also possible to assert that two given classes are disjoint, i.e. they
do not share individuals. For properties, the owl:EquivalentProperty construct can be used to declare that two properties apply to the same subjects and objects. This feature is completed, at the level of individuals in general, by constructs that allow to state that individuals are different from or same as other individuals.

An important OWL feature is that such identity and distinction axioms may perfectly hold between elements coming from different ontologies. One can therefore state that a property used in one context is semantically equivalent to a property used in a different context, or one can simply reuse it in a different ontology. This is crucial to meet crucial interoperability requirements, such as the “metadata interoperability” use case in Chapter 2.

Finally, it is worth mentioning that in the OWL world ontologies are themselves considered as full-fledged entities (of type owl:Ontology) that can themselves be described. It is therefore possible to provide ontologies with metadata – creator, date of creation, etc. More interestingly from a Semantic Web perspective, one can state that an ontology imports another. This allows for sharing and re-using complete vocabularies – and their axiomatizations – in an explicit manner throughout the Semantic Web.

Concerning syntax, a first observation is that, as RDFS, OWL allows to define ontologies using RDF itself. It can therefore be written using the RDF XML syntax, or any of the other syntaxes mentioned in the previous chapter, like Turtle. In OWL, books can, for example, be restricted to a subclass of all objects with at most one ISBN – we then say that the link between books and ISBN number has a maximal cardinality of 1. In the XML syntax of OWL, this reads as follows:

```xml
<owl:Class rdf:about="#Books">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasISBN" />
      <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

However, OWL also has additional syntaxes, including an XML syntax (Motik et al. 2009b), a functional abstract syntax (Motik et al. 2009d) and the popular Manchester Syntax (Horridge and Patel-Schneider 2009) to make the language more accessible for human readers (Patel-Schneider et al. 2004). The same axiom would now read as SubClassOf(Books, restriction(hasISBN minCardinality(1))) in Functional Syntax. For the sake of better readability, we will use Functional Syntax or DL expressions in the rest of this book.

### 7.2.2 The formal semantics of OWL and its different layers

One of the major interests of an ontology language like OWL is its being grounded on formal semantics. As seen in the previous chapter, RDFS constructs come with entailment rules that can be implemented by inference engines to derive new facts from asserted ones. OWL also provides semantics to its constructs, which enable to infer new information from existing one, but also to check the consistency of a given RDF graph with respect to the specifications.
found in its underlying ontology(ies). Or, for ontology engineering purposes, to verify if the axioms found in an ontology do not raise inconsistency by themselves.

A very simple example would be to define Shakespeare to be both a Person and a Group. If we also add an axiom that persons and groups are disjoint, i.e., these classes that cannot share any instances (formally $\text{DisjointClasses}(\text{Person, Group})$), the ontology is inconsistent.

These mechanisms rely on a formal semantic interpretation of the ontology language constructs. In the case of OWL, one proposed interpretation is a model-theoretic one, inspired from description logic approaches (Baader et al. 2003). Considering a domain of interpretation (a set of resources), an OWL class can be mapped to a set of elements from this domain, its instances. Similarly, a property can be associated a set of pairs of resources, corresponding to the subjects and objects of each statement for which the property is the predicate.

The axioms and expressions defining classes and properties are in turn interpreted as conditions that apply to the extensional interpretations of these vocabulary elements. Consider the expression $\text{DisjointClasses}(\text{Person, Group})$ mentioned above. It can be formally interpreted as a semantic condition $\text{EC}(\text{Person}) \cap \text{EC}(\text{Group}) = \emptyset$, where $\text{EC}(c)$ denotes the interpretation of a class $c$. This condition can then be checked by an inference engine once the extension of the classes are (partially) known thanks to $\text{rdf:type}$ statements, or based on other ontology axioms.

The Web Ontology Language OWL in its second version is based on the description logic $\text{SROIQ}(D)$, i.e. $\text{SROIQ}$ extended with datatypes. In the following we briefly introduce a fragment of OWL-2 and the fundamental reasoning problems related to DLs. For details we refer the reader to (Hitzler et al. 2009; Horrocks et al. 2006). Later in this book, we propose extend reasoning with OWL towards fuzzy logics. Table 7.1 gives an overview of the operators used in this fragment, both in description logic syntax, which is very compact, and in OWL functional syntax. We omit operators, which can be reduced to more basic ones, e.g. $\text{EquivalentClasses}(C, D)$ can be reduced to $\text{SubClassOf}(C, D)$, $\text{SubClassOf}(D, C)$. We also omit data properties, i.e. properties ranging over literal values. The functional syntax for data properties is analogous to that for object properties, but with the prefix $\text{Data}$ instead of $\text{Object}$.

Classes, roles and individuals form the vocabulary of an ontology. In OWL, they are named using URIs, or anonymous and denoted by blank nodes.

**Definition 7.2.1 (Vocabulary)** A vocabulary $V = (N_C, N_P, N_I)$ is a triple where

- $N_C$ is a set URIs used to denote classes,
- $N_P$ is a set URIs used to denote roles and
- $N_I$ is a set URIs used to denote individuals.

$N_C, N_P, N_I$ need not be disjoint.

An interpretation grounds the vocabulary in objects from an object domain.

**Definition 7.2.2 (Interpretation)** Given a vocabulary $V$, an interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I}_C, \cdot^\mathcal{I}_R, \cdot^\mathcal{I}_I)$ is a quadruple where

- $\Delta^\mathcal{I}$ is a nonempty set called the object domain;
<table>
<thead>
<tr>
<th>#</th>
<th>Description</th>
<th>OWL abstract syntax</th>
<th>DL syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The universal concept</td>
<td>owl:Thing</td>
<td>⊤</td>
</tr>
<tr>
<td>2</td>
<td>The empty concept</td>
<td>owl:Nothing</td>
<td>⊥</td>
</tr>
<tr>
<td>3</td>
<td>$C$ is a subclass of $D$</td>
<td>SubClassOf(C D)</td>
<td>$C \sqsubseteq D$</td>
</tr>
<tr>
<td>4</td>
<td>Class intersection of $C$ and $D$</td>
<td>ObjectIntersectionOf(C D)</td>
<td>$C \cap D$</td>
</tr>
<tr>
<td>5</td>
<td>Class union of $C$ and $D$</td>
<td>ObjectUnionOf(C D)</td>
<td>$C \sqcup D$</td>
</tr>
<tr>
<td>6</td>
<td>$C$ is the inverse class of $D$</td>
<td>ObjectComplementOf(C D)</td>
<td>$D \equiv \neg C$</td>
</tr>
<tr>
<td>7</td>
<td>The class of things, which stand in relation $R$ only with with $Cs$</td>
<td>ObjectAllValuesFrom(R C)</td>
<td>$\exists R.C$</td>
</tr>
<tr>
<td>8</td>
<td>The class of things, which stand in relation $R$ with at least one $C$</td>
<td>ObjectSomeValuesFrom(R C)</td>
<td>$\exists R.C$</td>
</tr>
<tr>
<td>9</td>
<td>The class of things, which are $R$-related to themselves</td>
<td>ObjectHasSelf(R)</td>
<td>$\exists R.\text{Self}$</td>
</tr>
<tr>
<td>10</td>
<td>The class of things, which are $R$-related to at least $n$ $Cs$</td>
<td>ObjectMinCardinality(n R C)</td>
<td>$\geq nR$</td>
</tr>
<tr>
<td>11</td>
<td>The class of things, which are $R$-related to at most $n$ $Cs$</td>
<td>ObjectMaxCardinality(n R C)</td>
<td>$\leq nR$</td>
</tr>
<tr>
<td>12</td>
<td>The class consisting exactly of $a_1,\ldots,a_n$</td>
<td>ObjectOneOf(a_1,\ldots,a_n)</td>
<td>${a_1,\ldots,a_n}$</td>
</tr>
<tr>
<td>13</td>
<td>$R$ is a subproperty of $S$</td>
<td>SubObjectPropertyOf(R S)</td>
<td>$R \sqsubseteq S$</td>
</tr>
<tr>
<td>14</td>
<td>The property obtained from chaining $R_1,\ldots,R_n$</td>
<td>ObjectPropertyChain(R_1,\ldots,R_n)</td>
<td>$R_1 \circ \ldots \circ R_n \sqsubseteq S$</td>
</tr>
<tr>
<td>15</td>
<td>The inverse property of $R$</td>
<td>ObjectInverseOf(R)</td>
<td>$R^-$</td>
</tr>
<tr>
<td>16</td>
<td>$R$ is asymmetric</td>
<td>AsymmetricObject(Property(R))</td>
<td>$\text{Asy}(R)$</td>
</tr>
<tr>
<td>17</td>
<td>$R$ is reflexive</td>
<td>ReflexiveObject(Property(R))</td>
<td>$\text{Ref}(R)$</td>
</tr>
<tr>
<td>18</td>
<td>$R$ is reflexive</td>
<td>IrreflexiveObject(Property(R))</td>
<td>$\text{Irr}(R)$</td>
</tr>
<tr>
<td>19</td>
<td>$R$ and $S$ are disjoint</td>
<td>DisjointObjectProperties(R, S)</td>
<td>$\text{Dis}(R, S)$</td>
</tr>
<tr>
<td>20</td>
<td>$a$ is a $C$</td>
<td>ClassAssertion(a C)</td>
<td>$a : C$</td>
</tr>
<tr>
<td>21</td>
<td>$a$ and $b$ stand in relation $R$</td>
<td>ObjectPropertyAssertion(R a b)</td>
<td>$(a, b) : R$</td>
</tr>
<tr>
<td>22</td>
<td>$a$ and $b$ do not stand in relation $R$</td>
<td>NegativeObjectPropertyAssertion(R a b)</td>
<td>$(a, b) : \neg R$</td>
</tr>
<tr>
<td>23</td>
<td>$a$ and $b$ are the same</td>
<td>SameIndividual(a b)</td>
<td>$a = b$</td>
</tr>
<tr>
<td>24</td>
<td>$a$ and $b$ are different</td>
<td>DifferentIndividuals(a b)</td>
<td>$a \neq b$</td>
</tr>
</tbody>
</table>

Table 7.1 A core fragment of OWL2
• \( \mathcal{I}_C \) is the class interpretation function, which assigns to each class a subset of the object domain \( \mathcal{I}_C : N_C \rightarrow 2^{\Delta^I} \)
• \( \mathcal{I}_R \) is the role interpretation function, which assigns to each role a set of tuples over the object domain \( \mathcal{I}_R : N_R \rightarrow \Delta^I \times \Delta^I \)
• \( \mathcal{I}_I \) is the individual interpretation function, which assigns to each individual \( a \in N_I \) an element \( a^I \) from \( \Delta^I \).

Let \( C, D \in N_C \), let \( R, R_i, S \in N_R \) and \( a, a_i, b \in N_I \). We extend the role interpretation function \( \mathcal{I}_R \) to role expressions:

\[
(R^-)^I = \{(x, y) \mid (y, x) \in R^I\}
\]

We extend the class interpretation function \( \mathcal{I}_C \) to class descriptions:

\[
\top^I = \Delta^I \\
\bot^I = \emptyset \\
(C \cap D)^I = C^I \cap D^I \\
(C \cup D)^I = C^I \cup D^I \\
(-C)^I = \Delta^I \setminus C^I \\
(\forall R.C)^I = \{x \in \Delta^I \mid (x, y) \in R^I \rightarrow y \in C^I\} \\
(\exists R.C)^I = \{x \in \Delta^I \mid \exists y \in \Delta^I : (x, y) \in R^I\} \\
(\exists R.self)^I = \{x \in \Delta^I \mid (a, a) \in R^I\} \\
(\geq nR)^I = \{x \in \Delta^I \mid \exists y_1, \ldots, y_m \in \Delta^I : \\
\langle x, y_1 \rangle, \ldots, \langle x, y_m \rangle \in R \land m \geq n\} \\
(\leq nR)^I = \{x \in \Delta^I \mid \exists y_1, \ldots, y_m \in \Delta^I : \\
\langle x, y_1 \rangle, \ldots, \langle x, y_m \rangle \in R \land m > n\} \\
\{a_1, ..., a_n\}^I = \{a_1^I, ..., a_n^I\}
\]

Class expressions are used in axioms.

**Definition 7.2.3 (Axiom)** An axiom is one of

- A general concept inclusion of the form \( A \sqsubseteq B \) for concepts \( A \) and \( B \);
- An individual assertion of one of the forms \( a : C, (a, b) : R, (a, b) : \neg R, a = b \) or \( a \neq b \) for individuals \( a, b \) and a role \( R \);
- A role assertion of one of the forms \( R \sqsubseteq S, R_1 \circ \ldots \circ R_n \sqsubseteq S, \text{Asy}(R), \text{Ref}(R), \text{Irref}(R), \text{Dis}(R, S) \) for roles \( R, R_i, S \).

Satisfaction of axioms in an interpretation \( I \) is defined as follows.
Definition 7.2.4 (Satisfaction of Axioms) Satisfaction of axioms in an interpretation $\mathcal{I}$ is defined as follows. With $\circ$ we denote the composition of binary relations.

\[
(R \subseteq S)^I \equiv (x, y) \in R^I \rightarrow (x, y) \in S^I
\]

\[
(R_1 \circ \ldots \circ R_n \subseteq S)^I \equiv \forall (x, y_1) \in R_1^I, (y_1, y_2) \in R_2^I, \ldots,
\]

\[
(y_{n-1}, z) \in R_n^I : (x, z) \in S^I
\]

\[
(\text{Asy}(R))^I \equiv \exists (x, y) \in R^I : (y, x) \notin R^I
\]

\[
(\text{Ref}(R))^I \equiv \forall x \in \Delta^I : (x, x) \in R^I
\]

\[
(\text{Irr}(R))^I \equiv \forall x \in \Delta^I : (x, x) \notin R^I
\]

\[
(\text{Dis}(R, S))^I \equiv R^I \cap S^I = \emptyset
\]

\[
(C \subseteq D)^I \equiv x \in C^I \rightarrow x \in D^I
\]

\[
(a : C)^I \equiv a^I \in C^I
\]

\[
((a, b) : R)^I \equiv (a^I, b^I) \in R^I
\]

\[
((a, b) : \neg R)^I \equiv (a^I, b^I) \notin R^I
\]

\[
a = b \equiv a^I = b^I
\]

\[
a \neq b \equiv a^I \neq b^I.
\]

An ontology is comprised of a set of axioms. We can further differentiate between the T-Box describing the terms or classes in the ontology, the R-Box describing the properties, and the A-Box containing assertions about individuals.

Definition 7.2.5 (Ontology) A $\text{SROIQ}$ ontology $O$ is a set of axioms as defined in definition 7.2.3. A T-Box (Terminology) of $O$ defines the concepts used in $O$. It consists only of axioms of the forms 1-12 in table 7.1. A R-Box (Properties) of $O$ defines the properties used in $O$. It consists only of axioms of the forms 13-19 in table 7.1. An A-Box (Assertions) of $O$ defines the individuals used in $O$ and their relations. It consists only of axioms of the forms 20-24 in table 7.1.

7.2.3 Reasoning Tasks

The benefit of using a description logics based language of cause is the ability to automatically draw conclusions from the available facts. In this section we list the most common reasoning tasks for OWL2 (Motik et al. 2009c).

Definition 7.2.6 (Basic Reasoning Tasks) Let $O, O'$ be ontologies, and $\mathcal{V}$ the vocabulary of $O$.

Consistency $O$ is consistent (satisfiable), if a model of $O$ w.r.t. $\mathcal{V}$ exists;

Entailment Let $\mathcal{I}$ be a model of $O$ w.r.t. $\mathcal{V}$. $O$ entails $O'$ ($O \models O'$), if $\mathcal{I} \models O \Rightarrow \mathcal{I} \models O'$. We say $O$ and $O'$ are equivalent, if $O \models O'$ and $O' \models O$;
Satisfiability Let $C$ be a class expression. $C$ is satisfiable w.r.t. $O$ if a model $I$ of $O$ w.r.t. $V$ exists such that $C_I \neq \emptyset$. We say $O$ is satisfiable if all class expressions in $O$ are satisfiable. We say $O$ and $O'$ are equisatisfiable, if $O$ is satisfiable if and only if $O'$ is satisfiable.

Subsumption Let $C$ and $D$ be class expressions. $D$ subsumed by $C$ ($D \subseteq C$) w.r.t. $O$, if for each model $I$ of $O$ w.r.t. $V$: $D^I \subseteq C^I$. $C$ and $D$ are equivalent, if $D \subseteq C$ and $C \subseteq D$;

Instance Checking Let $C$ be a class expression. $a$ is an instance of $C$ w.r.t. $O$, if for each model $I$ of $O$ w.r.t. $V$: $a^I \in C^I$ and

Boolean Conjunctive Query Answering Let $Q$ be a boolean conjunctive query of the form $$\exists x_1, \ldots, x_n : A_1, \ldots, A_m$$ where $A_i$ is an atom of the form $C(s)$ or $P(s, t)$ with $C$ a class, $P$ a property and $s$ and $t$ individuals or some variable $x_j$. $Q$ is an answer w.r.t. $O$ if $Q$ is true in each model of $O$ w.r.t. $V$.

As these basic reasoning tasks can be reduced to each other, it is sufficient for a reasoner to implement one of them (Baader et al. 2003).

A common task is to compute the full class hierarchy of an ontology, for example for visualization purposes. The full hierarchy can be computed for example by mutually checking subsumption for all classes in an ontology.

SPARQLDL (E. Sirin and B. Parsia 2007) is a SPARQL entailment regime, which uses SPARQL to express conjunctive queries in order to query OWL knowledge bases.

7.2.4 OWL flavors

At the time of writing this book, there exist a number of available OWL reasoning engines which are able to carry such reasoning processes. Examples include Hermit (Motik et al. 2009e), RacerPro (Haarslev and Möller 2003) or Pellet (Sirin et al. 2007).

It is important to mention that several flavors of OWL have been introduced, depending on the formal expressivity and complexity required by the application at hand. Indeed, the reasoning mechanisms on which OWL inference tools rely can exhibit different levels of formal complexity, depending on the selection of OWL constructs that is considered. The more constructs are used, the less tractable inference will be.

The designers of the original OWL (S. Bechhofer et al. 2004) have thus proposed three sub-languages which embody different levels of compromise between expressivity and tractability: OWL-Full, OWL-DL and OWL-Lite.

The first, OWL-Full, actually includes all the constructs of RDFS and the description logic $SHOIN(D)$, and imposes no constraints on their use. It is for example allowed to consider classes themselves as instances of other classes. The second, OWL-DL, allows to use all $SHOIN(D)$ constructs, but with some constraints that prevent too complex knowledge bases to be built. Finally, the simplest flavor of OWL, OWL-Lite, imposes further constraints and limit the constructs that can be used to describe classes and properties.

In OWL2, the Full sub-language corresponds to the RDF based semantics. The former OWL-DL corresponds to the direct semantics. Moreover, there are three dialects with reduced expressivity and desirable computational behaviour (Motik et al. 2009a): OWL2 EL targets at very large ontologies with a large number of classes and allows for reasoning
in polynomial time. OWL2 QL allows for an implementation in LogSpace using relational database technology and hence is especially useful for implementations on top of existing databases. Finally, OWL2 RL is the fragment of OWL2, which can be implemented using forward chaining rules. It is particularly useful for relatively lightweight ontologies with a large number of individuals.

7.2.5 Beyond OWL

Readers should be aware that even the most complete flavor of OWL may yet not fit all ontology engineering requirements. Among the most noticeable shortcomings, OWL does not allow useful features like operations on (typed) literals, such as concatenating string values or summing integers. Neither does it not enable specification of chains of properties – as a famous example, it is impossible to state that the brother of the father of one individual is her uncle.

This explains why a number of complementary knowledge representation languages co-exist alongside OWL. In particular, languages that enable to represent knowledge in the form of generic rules are often mentioned, such as Description Logic Programs (DLP) (Grosof et al. 2003) or the Semantic Web Rule Language (SWRL) (Horrocks et al. 2004).

With OWL it is possible to create various kinds of ontologies that can be useful to the multimedia domain, such as the Music Ontology mentioned in Chapter 2, or the COMM ontology that will be presented in Chapter 9. A first application of OWL which we will however discuss right now is the creation of an ontological framework to represent conceptual vocabularies that are not amenable to full-fledged semantic web formal ontologies.

7.3 A language to represent simple conceptual vocabularies: SKOS

Ontology languages, like RDFS and OWL, are a necessary part of the Semantic Web technology. With them, it is possible to create structured bodies of knowledge that enable to describe and exchange information in a flexible, yet shared and controlled way. However, ontologies are not the single source of conceptual information to be found and used on the Semantic Web.

7.3.1 Ontologies vs. Knowledge organization systems

A number of concept schemes exist, that are not meant as schemas to represent structured data about entities—defining for instance a “subject” property that applies to “books”. Instead, they provide mere values that will appear in such structured descriptions for these entities—e.g., a “geography” topic. Typical examples are thesauri (International Standards Organisation 1986) and classification schemes that are used to describe documents in all kind of institutions, say, libraries and audiovisual archives. Often, these schemes result from knowledge organization efforts that largely predate the rise of Semantic Web technology, but are nevertheless of crucial interest for the establishment of a rich information network.

These description vocabularies, often referred to as knowledge organization systems, or KOSs, clearly belong to the knowledge level: they convey some controlled meaning for

2Indeed this is expressed by the “support controlled vocabularies” requirement of Section 2.3.
identified descriptive entities, be they “terms”, “notations” or “concepts”. Further, they do it in a structured and controlled manner, as testified by the existence of a number of dedicated shared guidelines (International Standards Organisation 1986) and formats, like ZThes\(^3\) or Marc 21.\(^4\)

Machine-readability and exploitation have high priority in the design of ontologies. Knowledge organization systems like thesauri are more targeted at human users; and the range of approaches to exploit them tend to privilege domain-intuitive associations rather than complex reasoning (Isaac et al. 2009). As a result, KOSs are not easily amenable to formal semantic characterization. They focus rather on the description of lexical knowledge and simple semantic relation between concepts, like specialization or mere association, as exemplified in the following:

```
animal
  NarrowerTerm cat
  NarrowerTerm wildcat
  cat
    UsedFor domestic cat
    RelatedTerm wildcat
    BroaderTerm animal
  ScopeNote used only for domestic cats
```

To port these conceptual information on the Semantic Web, one first solution consists in formalizing the knowledge they contain, making them full-fledged ontologies. This has already been done, and can be fruitful for projects that have a clear use for such added semantics (Hyvonen et al. 2004). Yet, it proves to be a very difficult task, comparable to the one of creating a new ontology, but at a scale larger than most ontology engineering work—KOSs routinely contain thousands of concepts.

A second, simpler solution is to represent KOS knowledge as close as possible to what can be found in the original information sources, using an appropriate Semantic Web ontology. This of course restricts the resulting amount of formal information available, but corresponds to observed requirements and eases conversion work. The latter can even be automatized, once a mapping has been established between the original KOS schema and the one chosen to represent KOS information on the Semantic Web (van Assem et al. 2005). One possible means for achieving such a representation is to use SKOS.

### 7.3.2 Representing concept schemes using SKOS

**The SKOS context**

SKOS\(^5\), which stands for Simple Knowledge Organization System, is a standard model proposed by the W3C—in the context of the Semantic Web Deployment Working Group\(^6\)—in order to represent common KOS information in a semantic-web compatible form, so as to:

\(^{3}\)http://zthes.z3950.org/
\(^{4}\)http://www.loc.gov/marc/authority/
\(^{5}\)http://www.w3.org/2004/02/skos/
\(^{6}\)http://www.w3.org/2006/07/SWD/
Semantic Web Languages

- share information that corresponds to the domain practices (that is, close to norms like ISO 2788);
- develop and re-use standard tools (concept-based search engines, browsers) for the exploitation of the KOSs published on the Semantic Web.

The development of SKOS was driven by requirements elicited from a number of practical use cases gathered from the community of KOS users (Isaac et al. 2009). As such, it benefited from the input of semantic web researchers, but also from information science specialists. The resulting model is presented in the SKOS Reference document (Miles and Bechhofer 2009), while a more informal overview is given in the SKOS Primer (Isaac and Summers 2009).

Describing concepts in SKOS

SKOS proposes a concept-based model for representing controlled vocabularies. As opposed to a term-based approach, where terms from natural language are the first-order elements of a KOS, SKOS uses more abstract concepts which can have different lexicalizations. SKOS introduces a special class (skos:Concept) to be used in RDF data to properly type the concerned entities. To define further the meaning of these conceptual resources, SKOS features three kinds of descriptive properties.

Concept labels

Labelling properties, e.g. skos:prefLabel and skos:altLabel, link a concept to the terms that represent it in language. The prefLabel value should be a non-ambiguous term that uniquely identifies the concept, and can be used as a descriptor in an indexing system. altLabel is used to introduce alternative terms (synonyms, abbreviations, variants) which can be used to retrieve the concept via a proper search engine, or to extend a document’s index by additional keywords. It is important to note that SKOS allows the concepts to be linked to preferred and alternative labels in different languages. KOSs can thus be used seamlessly in multilingual environments, as demonstrated by the following example (using the RDF Turtle syntax introduced in the previous chapter).

```
@prefix ex: <http://example.org/animals/> .
ex:cat a skos:Concept ;
skos:prefLabel "cat"@en ;
skos:altLabel "domestic cat"@en ;
skos:prefLabel "chat"@fr .
```

Semantic relationships between concepts

Semantic properties are used to represent the structural relationships between concepts, which are usually at the core of controlled vocabularies like thesauri. SKOS uses skos:broader, which denotes the generalization link, skos:narrower, its reciprocal link, and skos:related, which denote non-hierarchical associative relationship. For instance, using skos:broader to relate two concepts (say, cat and animal) allows to represent that the first admits the second as a generalization.

```
ex:cat skos:broader ex:animal ;
skos:related ex:wildcat .
ex:animal skos:narrower ex:cat .
```
**Documentation**

Even though less crucial to representing the meaning of a concept, informal documentation plays an important role in KOSs. To represent it, SKOS features documentation notes that capture the intended meaning of a concept using glosses—skos:scopeNote, skos:definition, skos:example—and management notes—skos:editorialNote, skos:changeNote and skos:historyNote.

```
ex:cat skos:scopeNote "used only for domestic cats"@en .
```

**Representing conceptual vocabularies**

Analogous to OWL, SKOS allows vocabularies themselves to be represented and described as full-fledged individuals. It coins a skos:ConceptScheme class for this. It also introduces specific properties to represent the links between KOSs and the concepts they contain. skos:inScheme is used to assert that a concept is part of a given concept scheme, while skos:hasTopConcept states that a KOS contains a given concept as the root of its hierarchical network.

```
ex:plantVocabulary a skos:ConceptScheme ;
    skos:hasTopConcept ex:tree .
ex:tree skos:inScheme ex:plantVocabulary .
```

**7.3.3 Characterizing concepts beyond SKOS.**

There are more features to SKOS than the ones discussed in the above paragraph. Readers are referred to the SKOS documentation, which presents more advanced functionalities, such as grouping concepts or linking labels together (Miles et al. 2005). They should be aware that the SKOS model however does not provide means to capture all the information that can be virtually attached to KOSs and their elements. Indeed, for a number of features that are not specific to the KOS domain, such as the mention of a concept’s creator, SKOS recommends the use of constructs that already exist in other vocabularies—in that specific case, the Dublin Core dct:creator property. The SKOS documentation also encourages users to extend the SKOS constructs so as to fit their specific needs. A typical example will be to create a sub-property of skos:related to represent a specific flavor of association, like between a cause and its consequence. Application thus benefit from a finer representation grain while retaining compatibility with standard SKOS tools.

```
@prefix dct: <http://purl.org/dc/terms/> .
@prefix myVoc: <http://example.org/mySKOSextension/> .
myVoc:hasBodyPart rdfs:subPropertyOf skos:related .
ex:cat dct:creator <http://www.few.vu.nl/~aisaac/foaf.rdf#me> ;
    myVoc:hasBodyPart ex:whiskers .
```
Linking concepts together using SKOS

SKOS allows to explicitly relate concepts within one concept scheme using semantic relations like skos:broader. But one of the aims of SKOS is to enable the creation of richer networks. On the Semantic Web, the value of descriptions come from the links they maintain to each other, and especially to descriptions coming from a different context. Using SKOS to represent controlled vocabularies naturally allows to have their elements involved in statements that explicitly relate them to entities outside of their original context.

While a broad variety of relations are of course possible, SKOS focuses on two aspects. The first is the re-use of entities across concept schemes. In SKOS, it is perfectly legal to state that a concept belongs to several schemes at a same time, or to use a semantic relation like skos:broader to link two concepts from different schemes.

Further, SKOS proposes a range of mapping properties to link concepts, typically for applications where semantic interoperability has to be enabled by using semantic alignment technologies (Euzenat and Shvaiko 2007). Using the properties skos:exactMatch or skos:closeMatch, it is possible to assert that two concepts have equivalent meanings. broadMatch, narrowMatch and relatedMatch are used to mirror the semantic relations presented in the above paragraph. These properties can be employed in situations where it is important to distinguish application-specific alignment links from carefully designed semantic relationships that specify the intrinsic meaning of considered concepts.

```
@prefix ex1: <http://example.org/animals/>
@prefix ex2: <http://example.org/moreGeneralClassification/>

ex1:felis_silvestris_catus skos:exactMatch ex2:cat .
```

The ease of creating such explicit relations across contexts with SKOS naturally stems from its relying on Semantic Web architecture: from a knowledge representation perspective, there is no fundamental difference between links within one vocabulary and links across different vocabularies. But even though this may appear obvious to the reader of this book, we recall that this is a crucial departing from the usual KOS formats, where such links can only be represented in an implicit way—typically as mere textual notes.

The formal semantics of SKOS

To be fully usable on the Semantic Web, SKOS comes as an OWL ontology. This allows to specify some semantic axioms which reasoners can use to derive information from asserted SKOS statements. A typical example is the property skos:broaderTransitive which is used to represent “ancestor” links between concepts. To automatically produce the desired information from existing skos:broader statements using an OWL inference engine, this property is defined both as transitive and a super-property skos:broader. Consider the following example:

```
ex:cat skos:broader ex:animal .
ex:siamese_cat skos:broader ex:cat .
```

From these two statements, an OWL reasoning engine can infer the triple:

```
7http://www.w3.org/TR/skos-reference/skos.rdf
```
ex:siamese_cat skos:broaderTransitive ex:cat .

It is worth mentioning that it is impossible to represent all SKOS axioms using OWL alone. Some constraints would therefore require implementation by other means.

7.3.4 Using SKOS concept schemes on the Semantic Web

As said, the aim of SKOS is to port existing concept schemes to the Semantic Web and to seamlessly use them in RDF descriptions. Of course this requires to combine SKOS data with other kind of data—that is, information described using other ontologies. One typical example is when SKOS concepts are used to describe the subject of a document (its subject indexing). Such a thing can be done by creating an RDF description of the concerned document using a dedicated ontology, and relate this description to the one of the appropriate SKOS concept(s). Examples of ontologies that may be used, depending on the documentary context, are the Dublin Core metadata element set, as in the following example.


7.4 Querying on the Semantic Web

In the previous sections and chapter 6 we have explained, how knowledge can be expressed on the Semantic Web using a lasercake of increasingly expressive languages, and how to publish it using the linked data principles. To make actual use of this knowledge, an additional language to query knowledge on the Semantic Web is needed. This language is called SPARQL - the SPARQL protocol And RDF Query Language. All knowledge representation languages described in the previous sections are based on RDF graphs. Hence, SPARQL is a language for querying RDF graphs, based on graph pattern matching. In addition to the actual query language, SPARQL specifies a protocol and query/result formats for interacting with SPARQL endpoint implementations on the web (Clark et al. 2008; Dave Beckett and Jeen Broekstra (eds.) 2008). SPARQL offers the same expressiveness as Datalog and SQL (Angles and Gutierrez 2008). Syntactically, SPARQL is very close to Trig, as we will see in the following.

7.4.1 Syntax

A SPARQL query consists of three main parts:

1. A query result form. Possible result forms are
   - SELECT queries, which return variable bindings in a table (in the above example we have a SELECT query, see line 1),
   - CONSTRUCT queries, which return RDF by inserting variable bindings into a graph template,
   - ASK queries, which simply return true, if the graph pattern matches, and false otherwise.

8http://dublincore.org/documents/dcmi-terms/
2. A dataset. A dataset is a set of RDF graphs the graph pattern should be matched against, here `<http://example.org/annotationGraph>` in line 2, and finally

3. A graph pattern.

The dataset defines the scope of query evaluation, e.g., a particular ontology or a set of image annotations: A list of named graphs to be used can be defined. Additionally, a default graph is defined, which is used if and only if no scoping to named graphs is done. The smallest graph pattern is a single statement pattern. By default, statement patterns are matched against the default graph. However, they can be scoped to named graphs from the dataset using the keyword `GRAPH`. The scoping named graph can also be a variable, in which case all named graphs from the dataset are matched and the variable is bound accordingly.

Example 7.4.1 For example if the dataset and graph patterns shown below are used, pattern 1 would be matched against the union of `http://dbpedia.org` and `gn:dbpediaMapping`. Pattern 2 would be matched against `http://dbpedia.org` and `<http://swoogle.com>`, binding `?g` to the name of the named graph containing the matched statement. Pattern 3 would be matched against `http://dbpedia.org` only and pattern 4 would match nothing, because `gn:dbpediaMapping` is not one of the named graphs declared in the dataset using `FROM NAMED`. Note that variable names are written with a leading question mark in SPARQL.

```
# prefixes as in Turtle
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
PREFIX owl: <http://www.w3.org/2002/07/owl#> .
PREFIX foaf: <http://xmlns.com/foaf/0.1/> .

# Dataset
FROM <http://dbpedia.org>
FROM gn:dbpediaMapping
FROM NAMED <http://dbpedia.org>
FROM NAMED <http://swoogle.com>

# graph pattern 1:
GRAPH {?s rdf:type dbpedia:Visitor,Attraction}

# graph pattern 2:
GRAPH {?s rdfs:type dbpedia:Visitor,Attraction}

# graph pattern 3:
GRAPH {<http://dbpedia.org/>, ?s rdf:type dbpedia:Visitor,Attraction}

# graph pattern 4:
GRAPH gn:dbpediaMapping {?s owl:sameAs ?o}
```

Complex graph patterns are composed of simpler ones using joins, unions, left outer joins and filter expressions, which further constrain variable bindings. We will describe particularities of these connectives later in this chapter.

In the case of a SELECT query, the variable bindings produced by matching the graph pattern are projected to a tabular representation. For example the expression `SELECT ?g ?p ex:constant` would return a table containing three columns with the bindings of `?g`, `?p`, and a constant value. In the result of a SELECT query, null bindings for a subset of the variables used are allowed.

In the case of CONSTRUCT queries, the variable bindings are used to construct new RDF statements. The CONSTRUCT pattern in this case again is a graph pattern. However, it
is not matched against existing data but instantiated with the variable bindings computed during graph pattern matching. All valid RDF statements created during this instantiation are returned as result of the query. In this context, ‘valid’ means subjects and objects of all statements are in \( U \cup B \cup L \) (as defined in the previous chapter), predicates in \( U \) and all variables are bound. All blank nodes in an instantiated CONSTRUCT pattern are standardized apart. The scope of a blank node is exactly the RDF graph resulting from a single instantiation of the variables in the CONSTRUCT pattern. This goes in line with Definition 6.2.12, which scopes blank nodes to a single RDF graph.

In the following formal introduction we will use a syntactical subset of SPARQL, which however semantically can express all valid SPARQL SELECT and CONSTRUCT queries. For the full specification, we refer the reader to (E. Prud’hommeaux, A. Seaborne eds.).

We start by defining graph patterns. Graph patterns can contain filter expressions. For didactic reasons, we defer the definition of filter expressions.

**Definition 7.4.2 (Graph Patterns)** Let \( V \) be the set of SPARQL variables. A statement pattern is an element of \( V \cup U \cup B \cup L \times V \cup U \times V \cup U \times V \cup U \times V \cup B \times V \cup L \) as defined in definitions 6.2.1, 6.2.3 and 6.2.5. Statement patterns are written as statements in Turtle. A graph pattern is defined inductively as follows:

- A statement pattern is a graph pattern. A statement pattern is basic graph pattern (BGP).
- If \( P_1, \ldots, P_n \) are graph patterns, \( P = P_1 \ldots P_n \) is a graph pattern. If \( P_1, \ldots, P_n \) are BGPs, \( P \) is a BGP.
- The empty BGP \( \{ \} \) is a graph pattern.
- If \( P \) is a graph pattern, \( \{ P \} \) is a graph pattern called a group graph pattern.
- If \( P \) is a graph pattern and \( F \) is a filter expression, \( P \text{ FILTER } (F) \) is a graph pattern called a filter pattern. The scope of \( F \) is the surrounding group graph pattern.
- If \( P, Q \) are group graph patterns, \( P \text{ OPTIONAL } Q \) is a graph pattern called an optional pattern.
- If \( P, Q \) are group graph patterns, \( P \text{ UNION } Q \) is a graph pattern called an alternative pattern.
- Let \( g \in U \cup V \). If \( P \) is a group graph pattern, \( \text{GRAPH } g \ P \) is a graph pattern called a graph graph pattern.

Let \( P \) be a graph pattern. By \( \text{vars}(P) \) we denote the set of variables used in \( P \).

Definition 7.4.2 is a bit more restrictive than (E. Prud’hommeaux, A. Seaborne eds.), which allows to use the same abbreviations for BGPs as Turtle allows for RDF graphs. Moreover, we require the use of group graph patterns in some places, where SPARQL also allows BGPs. The additional brackets in our case make scoping more obvious. Both differences are purely syntactical. In fact, where convenient, we will use Turtle abbreviations in examples and assume that they are resolved before any further processing takes place. Moreover, we require that FILTER always occurs after a BGP. In contrast, (E. Prud’hommeaux, A. Seaborne eds.) allows for filter expressions to occur anywhere within a BGP. Hence, in this work we do not need to ensure FILTERs are evaluated in any special
order, which makes the algebraic specification of the semantics and later the specification of extensions a bit simpler.

Example 7.4.3 The following graph pattern selects all things, which are of type foaf:Person and their foaf:names. If available, the dbpprop:birthplace or the dbpprop:placeOfBirth are selected afterwards.

```sparql
1 { ?person a foaf:Person. ?person foaf:name ?name } 
2 OPTIONAL { ?person dbpprop:birthplace ?city } UNION 
3 { ?person dbpprop:placeOfBirth ?city } 
4 }
```

Graph patterns are matched against a dataset. The result is a set of variable bindings, which can further be restricted using filter expressions.

Definition 7.4.4 (Filter Expressions) Let \( f \) be a URI or one of the built-in operators listed in (E. Prud’hommeaux, A. Seaborne eds.), section 119.

- Let \( v_1, \ldots, v_n \in \mathcal{V} \cup \mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \). Then \( f(v_1, \ldots, v_n) \) is a filter expression.
- Let \( v_1, \ldots, v_n \in \mathcal{V} \cup \mathcal{U} \cup \mathcal{B} \cup \mathcal{L} \cup \{ \text{filter expressions} \} \). Then \( f(v_1, \ldots, v_n) \) is a filter expression.
- If \( F_1, F_2 \) are filter expressions, \((F_1 \land F_2), (\lor F_1)\) and \((\neg F_1)\) are filter expressions.

As we can see, filter expressions again can be parameters of filter expressions.

Example 7.4.5 We extend our example, such that only those persons are returned, who have a name starting with "A".

```sparql
1 { { ?person a foaf:Person. ?person foaf:name ?name } 
2 FILTER (REGEX(?name, "^A") ) } 
3 OPTIONAL { 
4 { ?person dbpprop:birthplace ?city } UNION 
5 { ?person dbpprop:placeOfBirth ?city } 
6 }
```

Dataset declarations, which we have intuitively introduced at the beginning of this chapter, are defined next. Then, we have everything in place to define SPARQL queries.

Definition 7.4.6 (Dataset Declaration) Let \( n \) be a URI, which is a name of a named graph. Then

- FROM \( n \) is a dataset declaration. We say graph \( n \) is part of the default graph of \( D \).
- FROM NAMED \( n \) is a dataset declaration. We say \( n \) is a named graph in \( D \).
- If \( D_1 \) and \( D_2 \) are dataset declarations, then \( D_1 D_2 \) is a dataset declaration.

---

9 STR casts any literal or URI to a string; LANG extracts only the language tag of a plain literal; LANGMATCHES checks whether a plain literal has a particular language tag; DATATYPE returns the datatype of a typed literal; BOUND checks whether a variable has been bound; sameTerm checks whether two RDF terms are the same; isIRI, isURI, isBLANK and isLITERAL check for the type of a variable; REGEX does regular expression string pattern matching; and the usual comparators >, <, =, <=, >=, !=.
If $D$ is a dataset, by $D_D$ we denote the graphs comprising the default graph of $D$, by $D_G$ we denote the union of the graphs in $D_D$ and by $D_N$ we denote the named graphs in $D$.

There are four types of SPARQL queries: SELECT, CONSTRUCT, ASK and DESCRIBE queries. SELECT queries return a table of variable bindings, as known from SQL. CONSTRUCT queries return RDF graphs. ASK queries return true, if the graph pattern matches the dataset. DESCRIBE queries are used to obtain a description of a single resource in RDF. We will not discuss DESCRIBE queries in this work. In the following we will omit BASE and PREFIX declarations, as they are purely syntactical.

Definition 7.4.7 (SELECT query) Let $D$ be a dataset declaration and $P$ be a group graph pattern. Let $v_1, \ldots, v_n \in \mathbb{V} \cup \mathbb{L} \cup \mathbb{U}$. Then $Q = \text{SELECT } v_1, \ldots, v_n \text{ WHERE } P$ is a SELECT query. We call $P$ the WHERE pattern of $Q$ and $v_1, \ldots, v_n$ the select variables of $Q$.

Example 7.4.8 The following query extracts a list of visitor attractions in Berlin from DBpedia (everything, which is in a subcategory of visitor attraction). The location is restricted using a bounding box on the geographic coordinates of the visitor attraction.

```sparql
SELECT ?attraction
FROM <http://dbpedia.org>
WHERE {
  ?attraction geo:lat ?lat.
  FILTER (?lat > "52.3"^^xsd:float & &
  ?lat < "52.7"^^xsd:float & &
  ?long > "13.1"^^xsd:float & &
  ?long < "13.5"^^xsd:float )
}
```

Definition 7.4.9 (CONSTRUCT query) Let $D$ be a dataset declaration and $P$ be a group graph pattern. Let $C$ be a BGP. Then $Q = \text{CONSTRUCT } \{ C \} \text{ WHERE } P$ is a CONSTRUCT query. We call $P$ the WHERE pattern of $Q$ and $C$ the CONSTRUCT pattern of $Q$. We call a statement pattern in $C$ a CONSTRUCT statement pattern of $Q$.

Example 7.4.10 The following CONSTRUCT query matches multiple properties used in DBpedia for places of birth and returns a new graph, which uses only one such property.

```sparql
CONSTRUCT { ?person a foaf:Person. ?person placeOfBirth ?city }
WHERE { 
  ?person a foaf:Person 
  OPTIONAL { (?person dbpprop:birthPlace ?city) UNION
              (?person dbpprop:placeOfBirth ?city) }
}
```

Definition 7.4.11 (ASK query) Let $D$ be a dataset declaration and $P$ be a group graph pattern. Then $Q = \text{ASK } D \text{ P}$ is an ASK query.

Example 7.4.12 The following query checks, whether we have a German label for dbpedia:Berlin.
7.4.2 Semantics

It remains to define the semantics of a valid SPARQL query. (E. Prud’hommeaux, A. Seaborne eds.) define an algebraic semantics, which we summarize here. Sparql employs a multi set based semantics, i.e. a query result may contain duplicate solutions, in order to ease implementation. For ease of specification we use a set based semantics. We also slightly restrict the syntax, such that the FILTER operator can be formalized simpler compared to (E. Prud’hommeaux, A. Seaborne eds.), which allows FILTER expressions to occur anywhere inside a BGP, which requires a more complex definition of the semantics of FILTER.

BGP matching forms the basis for the SPARQL semantics.

Definition 7.4.13 (BGP matching) Let \( P \) be a BGP. Let \( \mathcal{V} \) be the set of variables used in \( P \) and \( \mathcal{B} \) be the set of blank nodes used in \( P \). Let \( A \) be an RDF instance mapping as defined in Definition 6.2.12 and \( T \) be a term mapping of the form \( T: \mathcal{V} \rightarrow \mathcal{U} \cup \mathcal{L} \cup \mathcal{B} \). We say \( P \) matches an RDF graph \( G \), if there exist \( A, T \), such that \( P/(A \cup T) \) is a subgraph of \( G \). We call \( A \cup T \) a pattern instance mapping. A solution \( \mu \) of \( P \) wrt. \( G \) is the restriction of \( A \cup T \) to the variables in \( P \). The only solution of the empty BGP is the empty solution \( \mu_0 \).

Obviously, \( \mu \) is a mapping of the form \( \mathcal{V} \rightarrow \mathcal{U} \cup \mathcal{L} \cup \mathcal{B} \). Note that SPARQL uses the subgraph relationship instead of entailment for matching. This avoids infinite redundant mappings from blank nodes to other blank nodes. Instead, there is one solution per pattern instance mapping, and the number of pattern instance mappings is bound by the number of terms used in \( P \) and \( G \).

SPARQL graph pattern matching takes place on the extension of an RDF graph, i.e. the closure of all statements, which can be inferred from the knowledge base. Of cause, an implementation, does not need to materialize this extension. The exact semantics, for which this extension is computed, is defined in an entailment regime. The standard entailment regime is basic RDF graph pattern matching. In this case, the above required match would not be retrieved. However, if the RDF repository supports lightweight OWL reasoning (in particular transitivity of properties), the graph pattern would be matched against the RDFS closure of the repository. In this case, the statement \texttt{ex:Felix rdf:type ex:Cat} could be inferred and hence the pattern would match. Analogously, entailment regimes can be defined for RDFS and more expressive fragments of OWL. One of the most expressive proposed entailment regimes is SPARQL-DL (E. Sirin and B. Parsia 2007), which supports querying OWL ontologies and instances. In contrast, many OWL reasoners only support conjunctive queries. An OWL fragment supported by many repositories, which allow for lightweight OWL reasoning, is OWL Horst (ter Horst, H. J. 2005) - the fragment of OWL expressible using forward chaining rules.

Operators such as OPTIONAL and UNION combine solutions.
Definition 7.4.14 (Compatibility and Merge of Solutions) Let \( \mu_1, \mu_2 \) be solutions. We say \( \mu_1 \) and \( \mu_2 \) are compatible, if \( (x, y) \in \mu_1 \land (x, z) \in \mu_2 \Rightarrow y = z \), i.e. \( \mu_1 \) and \( \mu_2 \) map shared variables to the same terms. If \( \mu_1 \) and \( \mu_2 \) are compatible, then merge\((\mu_1, \mu_2) = \mu_1 \cup \mu_2.\)

Obviously, \( \mu_0 \) is trivially compatible with every solution and merge\((\mu, \mu_0) = \mu.\)

Filter expressions in SPARQL are functions returning a literal or the constant error. The definitions of the logical connectives \&\& \text{, } || \text{ and } ! \text{ are defined on error and the effective boolean values of their parameters. That means, if some filter function produces an error, the entire solution is (silently) discarded. Here we only give a very brief definition of filter functions and refer the reader to (E. Prud’hommeaux, A. Seaborne eds.) for the complete definition.}

A filter function is a mapping of the form \( f : n^{V \cup U \cup B \cup B \cup \{error\}} \rightarrow U \cup B \cup B \cup \{error\}.\)

If in a filter expression a parameter again is a filter expression, it is evaluated first and replaced by its result, before evaluation continues. Parameters are casted to suitable types, if a suitable casting function is defined by (Anders Berglund and Scott Boag and Don Chamberlin and Mary F. Fernández and Michael Kay and Jonathan Robie and Jérôme Siméon 2007). If a filter expression is not defined (e.g. because its parameters are outside its domain, or variables are not bound), it evaluates to error. Every filter function returns error, if one of its parameters is error. The special filter function \( \text{BOUND} \text{ returns true, if its parameter is a variable, which is not bound, false, if its parameter is a variable, which is bound, and error otherwise. It can be used to model negation by failure. The logical connectives of filter expressions work on a three-valued logic, including the error. The definitions of \&\& (a,b), || (a,b) \text{ and } !a \text{ are as follows (ebv is a function returning the effective boolean value of a variable):}

\[
\begin{array}{cccc}
\text{ebv(a)} & \text{ebv(b)} & || (a,b) & \&\& (a,b) \\
\text{true} & \text{true} & \text{true} & \text{true} \\
\text{true} & \text{false} & \text{true} & \text{false} \\
\text{true} & \text{error} & \text{true} & \text{error} \\
\text{false} & \text{true} & \text{true} & \text{false} \\
\text{false} & \text{false} & \text{false} & \text{false} \\
\text{false} & \text{error} & \text{error} & \text{error} \\
\text{error} & \text{true} & \text{true} & \text{error} \\
\text{error} & \text{false} & \text{false} & \text{true} \\
\text{error} & \text{error} & \text{error} & \text{error}
\end{array}
\]

\[
\begin{array}{cc}
\text{ebv(a)} & !a \\
\text{false} & \text{true} \\
\text{true} & \text{false} \\
\text{error} & \text{error} \\
\text{error} & \text{error}
\end{array}
\]

Next the semantics of graph pattern matching is defined, before we finish with the three types of queries. Our semantics is based on sets, instead of multisets as in (E. Prud’hommeaux, A. Seaborne eds.). (E. Prud’hommeaux, A. Seaborne eds.) uses multisets to ease implementation, even though this introduces duplicates in query results. Additional operators (DISTINCT and REDUSED) are then introduced to remove duplicates. For this work, set semantics makes no significant difference, as our logic based semantics can ensure unique results anyway. In order to keep the specification as short as possible, we therefore use sets and hence do not need to track cardinalities. We leave out all additional solution modifiers, such as ORDER BY, LIMIT and OFFSET, which do not add to the expressiveness, but can be added easily.
Definition 7.4.15 (Semantics of SPARQL Operators) Let $B$ be a BGP and $P, Q$ be graph patterns. Let $G$ be an RDF graph called the active graph. Let $\mu_0$ be the empty solution and $\Omega_0$ be the set containing exactly $\mu_0$. $\text{eval}$ is the SPARQL evaluation function, which defines the semantics of SPARQL graph patterns with respect to a dataset $D$. $\text{eval}$ is recursively defined as follows:

<table>
<thead>
<tr>
<th>expression</th>
<th>semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{eval}(B, G, D)$</td>
<td>${ \mu</td>
</tr>
<tr>
<td>$\text{eval}({ P }, D)$</td>
<td>$\text{eval}(P, G)$</td>
</tr>
<tr>
<td>$\text{eval}(P \cdot Q)$</td>
<td>${ \mu</td>
</tr>
<tr>
<td>$\text{eval}(P \text{ FILTER}(F), G, D)$</td>
<td>${ \mu</td>
</tr>
<tr>
<td>$\text{eval}(P \text{ OPTIONAL } Q, G, D)$</td>
<td>${ \mu</td>
</tr>
<tr>
<td>$\text{eval}(P \text{ UNION } Q, G, D)$</td>
<td>$\text{eval}(P, G, D) \cup \text{eval}(Q, G, D)$</td>
</tr>
<tr>
<td>$\text{eval}(\text{GRAPH } g \ (P), G, D)$</td>
<td>$\begin{cases} \text{eval}(P, g, D) \land g \in D_N \land g \in D_N; \ \text{eval}({ \text{GRAPH } n_1 P }) \cup \cdots \cup \text{eval}({ \text{GRAPH } n_n P }); G, D)</td>
</tr>
</tbody>
</table>

Both SELECT and CONSTRUCT queries project out some variables used in the graph pattern to the result.

Definition 7.4.16 (Projection) Let $V$ be a set of variables and $\mu$ be a solution. Then the projection $\pi$ of $\mu$ to $V$ is $\pi(\{ v_1, \ldots, v_n \}) = \begin{cases} (x, y) | x \in V \land \exists z, y \in \mu \land y = \text{null} \\ x \in V \land (x, y) \in \mu \end{cases}$

Definition 7.4.17 (Semantics of a SELECT query) Let $Q = \text{SELECT } v_1, \ldots, v_n \ D \text{ WHERE } P$ be a SELECT query. A result of $Q$ is $\pi(\{ v_1, \ldots, v_n \}, \text{eval}(P, D_G, D))$.

Example 7.4.18 Consider the query $Q$ from example 7.4.8.

The dataset is $D$ with $D_N = \{ \}, D_D = \{ <$ http://dbpedia.org $>$\}

Let $BGP = \{ \text{?attraction skos:subject ?category.} \\
\text{?category skos:broader} \\
\text{dbpedia:Category:Visitor_attractions.} \\
\text{?attraction geo:lat ?lat.} \\
\text{?attraction geo:long ?long } \}$
Then \( \mu = \{ (\text{attraction}, \text{dbpedia:Brandenburg\_Gate}),
(\text{lat}, "52.516272"ˆˆxsd:double), (\text{long}, "13.377722"ˆˆxsd:double) \} \) is a solution for eval(BGP, \( D_G, D \)):

\[
\begin{align*}
\text{?lat} & > "52.3"ˆˆxsd:float \&
\text{?lat} & < "52.7"ˆˆxsd:float \&
\end{align*}
\]

Let \( F = \{ \text{?long} > "13.1"ˆˆxsd:float \&
\text{?long} < "13.5"ˆˆxsd:float \} \).

\( \mu \) is also a solution for eval(\{\{\text{BGP}\} FILTER \( F, D_G, D \))

Hence, \( \nu = \{ (\text{attraction}, \text{dbpedia:Brandenburg\_Gate}) \} \) is a solution for \( Q \), because \( \pi(\text{attraction}, \text{eval(\{\{\text{BGP}\} FILTER \( F, D_G, D \}))} = \nu \).

**Definition 7.4.19 (Semantics of a CONSTRUCT query)** Let \( Q = \text{CONSTRUCT} \{ C \} \) \( D \)
WHERE \( P \) be a CONSTRUCT query. The result of \( Q \) is an RDF graph \( G = \{ (s,p,o) \mid \exists \mu \in \text{eval}(P, D_G, D) : (s,p,o) \in C/\mu \text{ and } (s,p,o) \text{ is a valid RDF statement} \} \).

Intuitively speaking, we evaluate \( P \) and instantiate \( C \) with every solution. From the result we only take the valid RDF statements, i.e. only partially instantiated statements are discarded, as are those with blank nodes or literals in predicate positions.

**Example 7.4.20** DBpedia contains the following statements:

\[
\begin{align*}
\text{dbpedia:Richard\_Feynman} & \text{ a foaf:Person} \\
\text{dbpprop:placeOfBirth dbpedia:Queens,\_New\_York} & \\
\text{dbpedia:Theodor\_Mommsen} & \text{ a foaf:Person} \\
\text{dbpprop:birthplace dbpedia:Schleswig} & \\
\text{dbpedia:Garding} &
\end{align*}
\]

Hence, the following are solutions for the graph pattern of the query in Example 7.4.10:

\[
\{ (?\text{person}, \text{dbpedia:Richard\_Feynman}),
(?\text{city}, \text{dbpedia:Queens,\_New\_York}),
(?\text{person}, \text{dbpedia:Theodor\_Mommsen}),
(?\text{city}, \text{dbpedia:Schleswig}),
(?\text{person}, \text{dbpedia:Theodor\_Mommsen}),
(?\text{city}, \text{dbpedia:Garding}) \}
\]

The result of the query therefore contains the following statements:

\[
\begin{align*}
\text{dbpedia:Richard\_Feynman} & \text{ a foaf:Person} \\
\text{dbpprop:placeOfBirth dbpedia:Queens,\_New\_York} & \\
\text{dbpedia:Theodor\_Mommsen} & \text{ a foaf:Person} \\
\text{dbpprop:placeOfBirth dbpedia:Schleswig} & \\
\text{dbpedia:Garding} &
\end{align*}
\]

**Definition 7.4.21 (Semantics of an ASK query)** Let \( Q = \text{ASK} \) \( D \) \( P \) be an ASK query. The solution of \( Q \) is true, if \( \exists \mu : \mu \in \text{eval}(P, D_G, S) \). It is false otherwise.

**Example 7.4.22** The query from example 7.4.12 evaluates to true. As the reader can verify by visiting [http://dbpedia.org/sparql?query=SELECT+*+FROM+%3Chttp://dbpedia.org\%3E+WHERE+%7B+%3Chttp://dbpedia.org\%3E+rdfs:label+?label%7D+FILTER+(LANGMATCHES(LANG(?label),\%22DE\%22)+\%22EN\%22)] there is at least one solution for the graph pattern of the query [as of 2010-07-21].
7.4.3 Default Negation in SPARQL

As SPARQL queries are evaluated on the extension of the knowledge base, it locally closes the world for the specified dataset. The underlying knowledge representation languages RDF and OWL, in contrast, assume an open world. Based on this locally closed world, SPARQL can be used to model negation by failure using a combination of OPTIONAL graph patterns and the BOUND filter, which checks whether a variable has been bound during graph pattern matching. We will call such a construct in a query bound negation. The semantics of such a construct intuitively is: "Try to find all solutions which I want to exclude from the result. Introduce a new (and possibly redundant) variable, which is bound only if such a solution is found. Then throw away all solutions, where this variable is bound."

In general, bound negation looks as follows. Let \( ?x \) be a variable occurring in \( GP2 \), \( ?y \) a new variable, which does not occur anywhere else in the query and \( s \ p \ ?x \) a statement pattern in \( GP2 \).

\[
\begin{align*}
\text{FILTER} \ (\text{BOUND}(?y)) \\
\end{align*}
\]

Obviously, using negation in SPARQL puts a heavy burden on the user. As a consequence, the next version of SPARQL (Harris and (eds.) 2010), which is undergoing standardization at the time of writing of this document, will provide an explicit operator for negation.

Example 7.4.23 The following query selects all geographic entities from DBpedia, which have a mapping to an entry from GeoNames, but from which DBpedia does not contain geocoordinates. These could then be imported from GeoNames. The query selects all places from dbpedia, which have some mapping (lines 7,8). For each mapping, we check, whether it is a mapping to a place in GeoNames (line 11). Then we try to find geocoordinates for this place in DBpedia (lines 13-18). If we find them, the place is discarded from the result (line 19), leaving only those without geocoordinates.

```
PREFIX gno <http://www.geonames.org/ontology#>
PREFIX dbpedia <http://dbpedia.org>
WHERE {
  GRAPH <http://dbpedia.org> {
    ?geoent a <http://dbpedia.org/ontology/Place>;
    owl:sameAs ?gnentry
  }
  GRAPH <http://sws.geonames.org> {
    ?gnentry gno:featureClass gno:P
  }
  OPTIONAL {
    GRAPH <http://dbpedia.org> {
      ?geoent geo:lat ?lat ;
      geo:lon ?lon
    }
  }
  FILTER (!BOUND(?lat))
}
```

7.4.4 Well Formed Queries

OPTIONAL patterns can lead to strange behaviour, if queries are not well formed. Queries are not well formed, if OPTIONALs are nested, such that the inner OPTIONAL uses some
variable from the surrounding graph pattern, which is not used in the outer OPTIONAL (Pérez et al. 2006a).

**Definition 7.4.24 (Well Formed Query)** We say a query is well formed, if for all nested OPTIONAL patterns \( GP_1 \{ GP_2 \text{ OPTIONAL } GP_3 \} \), \( \text{vars}(P_3) \cap \text{vars}(P_1) \subseteq \text{vars}(P_2) \cap \text{vars}(P_1) \).

To illustrate the problem, consider the following simple query, which is not well formed:

```sql
SELECT ?x
WHERE {
  {?x ex: b ?y} . {{?x ex: c ?z} OPTIONAL {?z ex: b ?y}}
}
```

When evaluated against the following graph

```plaintext
ex: a
ex: b ex: c
ex: a ex: c ex: d
```

there is exactly one solution \( \{(?x, \text{ex}:z)\} \).

Now if we add a single statement

```plaintext
ex: d ex: b ex: a
```

the solution is empty.

### 7.4.5 Querying for Multimedia Metadata

Queries on semantically-enriched media assets vary from navigating the decomposition of a video into shots and keyframes to retrieving all documents annotated with a given concept. Sophisticated queries may even take background knowledge into account or ask for complete scene descriptions represented in RDF. Hence, RDF and SPARQL allow for much richer, but still semantically well defined annotations than most of the state of the art (cf. section 7.4.7).

In the scenario presented in Section 2.1, we might be interested, e.g., in all images showing the heads of the United States and the Soviet Union together. To answer this query, we need to take decompositions of images, semantic annotations, and domain-specific knowledge into account in order to determine whether the persons depicted are heads of the USA or USSR.

In order to process and answer such queries, we are faced with various challenges with respect to the potential size of the dataset, complexity of queries, recursiveness of queries, and interactive access to media asset annotations. These challenges are elaborated below. As SPARQL alone is not sufficient to address all of them, we discuss proposed extensions where appropriate.

### Large Datasets.

The queried datasets may become extremely large. We estimate annotations in COMM of one million triples for one hour of video, which is decomposed into keyframes and annotated region based. If basic inferencing is done to compute subclass and instance relations, this may easily result in an increase by a large constant factor. On the other hand, most state of the art RDF repositories scale to tens or hundreds of million of statements\(^\text{10}\). Since about 2008,\(^\text{10}\)

\(^{10}\text{See http://esw.w3.org/topic/RdfStoreBenchmarking for an overview of RDF benchmarks.}\)
when the billion triples border has been broken (Harth et al. 2007; Kiryakov et al. 2005b; Schenk et al. 2009), performance of RDF stores has increased dramatically. Today, multi-billion statement repositories are possible (see also section 6.6.1). However, such repositories usually require powerful hardware or even clusters of repositories. Compared to this scale, typical datasets of background world knowledge, like DBPedia11, can almost be considered small.

Complex queries.

Queries can become extremely complex. A typical instantiation of a pattern in the Core Ontology for MultiMedia (COMM), which will be described in chapter 9, results in up to 20 statements. This complexity is not COMM specific, but typical for multimedia annotation, in order to capture the necessary expressivity (Troncy et al. 2007a). In turn, this results in a query with 20 statement patterns and 19 joins. Given the size of the datasets, this is a challenge that also most existing relational databases fail to meet. In order to avoid errors, it is desirable to hide these complex queries from application developers. In the case of COMM, COMM-API provides an abstraction layer for developers, which allows to access COMM items as Java objects without writing SPARQL queries.

As an example for a COMM based SPARQL query, assume we want to retrieve all images showing a cat, which is at least 3 years old. Additionally we would like to know, what type of cat we are faced with. We would like to select results 5 to 10. Then we could use the following query:

```
SELECT DISTINCT ?URI
FROM <http://example.org/annotationGraph>
WHERE {
  ?image rdf:type core:image-data;
  core:realized-by ?URI;
  ?annotation rdf:type core:semantic-annotation;
  core:setting-for ?image;
  core:setting-for ?label;
  core:satisfies [ rdf:type core:method;
  core:defines ?annotated-data-role;
  core:defines ?semantic-label-role].
  ?semantic-label-role rdf:type core:semantic-label-role.
  ex:age ?age.
  FILTER (?age > "3"^^xsd:integer).
} LIMIT 5 OFFSET 10
```

![Figure 7.1 Querying COMM](http://wiki.dbpedia.org/Datasets)
Complex recursive queries.

Annotations to multimedia items can be done on a variety of levels of decomposition. For example, a whole image can be annotated with a concept but also only a segment showing the concept or a compound ODF document containing the image. Hence, retrieval queries need to recursively follow the decompositions. Extensions to SPARQL that add support for regular path expressions have been proposed (Alkhateeb et al. 2009; Kochut and Janik 2007; Polleres et al. 2007; Prez et al. 2010), featuring regular graph expressions. However, such regular expressions are not expressive enough to capture all patterns used in COMM to annotate media assets. For this reason, a metadata repository must additionally support a specialized set of rules that allows to (recursively) follow decompositions (or other patterns) during retrieval. Such rules can be expressed for example in SWRL (Horrocks et al. 2004), N3 (Berners-Lee 2001) or using SPARQL syntax - Networked Graphs (Schenk 2008). For less complex (in particular non cyclic) rules, also DL-save rules (Motik et al. 2005) or regular path expressions (Alkhateeb et al. 2009; Kochut and Janik 2007; Prez et al. 2010) for SPARQL could be used.

7.4.6 Partitioning Datasets.

In contrast to many sources of world knowledge, multimedia metadata can easily be split horizontally. This means that annotations of two media assets are to a very large degree independent of each other. The links between them are usually indirect, specified through world knowledge. For example, two images could show the same scenery from different angles. However, the scenery is not part of the actual multimedia annotation but world knowledge. As a result, one possible approach to scaling querying of multimedia metadata is to distinguish between multimedia annotation and world knowledge and to accordingly split the datasets and queries. This allows us to come up with easier problems due to shorter queries and a smaller dataset. On the other hand, new challenges arise when splitting queries and datasets, such as determining relevant fragments for answering (a part of) a query or joining query results like efficiently handling distributed joins. Even though many of these challenges are well known from distributed and federated relational databases, they are more problematic for RDF as schema information is not reflected in the structure of data and an extremely high number of joins has to be handled compared to relational databases. For illustration, please remember that in relational databases the table structure implicitly reflects the schema of the data. In contrast, in RDF we have triples as the only structure and schema information is expressed explicitly using special predicates.

Systems supporting federated SPARQL evaluation include (Quilitz and Leser 2008; Schenk et al. 2009). (Eric Prud’hommeaux 2007) specifies a SPARQL extension to natively support federation in the language. Federation is a proposed optional feature of SPARQL1.1 (Prud’hommeaux and Seaborne 2010). This extension does only support semi joins and the marking up of subqueries, which should be evaluated remotely. The problem of query optimization for distributed queries is not addressed. Several state of the art systems support clustering, e.g. 4store (Harris et al. 2009) and Virtuoso12.

12http://virtuoso.openlinksw.com/
7.4.7 Related Work

Related work for querying semantic multimedia includes languages, which are similarly
generic as SPARQL, but also specialized multimedia query languages:

Other RDF query languages

Several RDF query languages have been designed, which implement parts of the SPARQL
extensions proposed above, including RQL (Karvounarakis et al. 2002), SeRQL (Broekstra
and Kampman 2003), TRIPLE (Sintek and Decker 2002), RDQL (Seaborne 2004) and N3
(Berners-Lee 2001). A comparison of supported query operators and capabilities in these
languages is given by (Haase et al. 2004), analysing support for graph operators such as
simple and optional path expressions, relational operators such as union and set difference,
quantification, aggregation operators such as count, sum and grouping, recursive operators
in addition to schema-level recursion in transitive predicates, abstract operators to query
reification and collections, operators to handle languages and datatypes in literals, and
support for RDF(S) entailment in query answering.

The semantics and computational behaviour of SPARQL has been studied (de Bruijn
et al. 2005; Muñoz et al. 2007), revealing for example some computationally and
semantically problematic corner cases (Pérez et al. 2006b). Also, approaches have been
developed to translate SPARQL queries into relational algebra (Cyganiak 2005; Harris and
Shadbolt 2005), enabling SPARQL query answering using relational databases, and into
Datalog (Polleres 2007; Schenk 2007), allowing engines to combine SPARQL queries with
ontological rules.

At the time of writing this book, the SPARQL Working Group\(^\text{13}\) is specifying SPARQL
1.1, which will bring the following additions to SPARQL (Harris and (eds.) 2010): Select
expressions will allow to return results, which are computed from bindings of variables. For
example, a price could be selected in Euro together with an exchange rate for the Dollar
and the Dollar price could be returned. Subselects allow for the specification of subqueries,
which can use the full range of result modifiers, including for example select expressions
and LIMIT. Property paths allow to specify regular expressions for predicates. For example,
one could write a query, which follows a transitive property. With the current version of
SPARQL, only paths of a fixed length can be matched. Optional features include entailment
regimes for the popular languages RDFS, all OWL profiles and the Rule Interchange Format
RIF. Moreover, an update language and basic support for query federation are being added.

Multimedia Query Languages

Besides the approach described above for querying media assets by the use of a semantic
database, there are also other approaches and solutions to query semantic multimedia. While
these languages are tailored to particular needs of multimedia querying, they lack the ability
of querying Semantic Web background knowledge.

For example, the commercial database Oracle with its Oracle Multimedia\(^\text{14}\) feature
provides for retrieving images, audio, and video. The multimedia package is an extension

---

\(^\text{13}\)http://www.w3.org/2009/sparql/wiki/Main\_Page

of the relational Oracle database. It supports the extraction of metadata from media assets and allows querying for media assets by specific indices.

The Digital Memory Engineering group at the Research Studios in Austria developed with the multimedia database METIS a sophisticated storage and management solution for structured multimedia content and its semantics (King et al. 2004; Ross et al. 2004). The METIS database provides a flexible concept for the definition and management of arbitrary media elements and their semantics. It is adaptable and extensible to the requirements of a concrete application domain by integrating application-specific plugins and defining domain-specific (complex) media types. In METIS, the semantic relationship of specific media elements and their semantics can be described to form new, independent multimedia data types. Those domain-specific media types can be bundled up and distributed in form of so-called semantic packs.

The multimedia presentation algebra (MPA) by Adali et al. (Adali et al. 2000) extends the relational model of data and allows for dynamically creating new presentations from (parts of) existing presentations. With the MPA, a page-oriented view on multimedia content is given. A multimedia presentation is considered as an interactive presentation that consists of a tree, which is stored in a database. Each node of this tree represents a non-interactive presentation, e.g., a sequence of slides, a video element, or a HTML page. The branches of the tree reflect different possible playback variants of a set of presentations. A transition from a parent node to a child node in this tree corresponds to an interaction. The proposed MPA allows for specifying a query on the database based on the contents of individual nodes as well as querying based on the presentation’s tree structure. It provides extensions and generalizations of the select and project operations in the relational algebra. However, it also allows to author new presentations based on the nodes and tree structure stored in the database. The MPA defines operations such as merge, join, path-union, path-intersection, and path-difference. These extend the algebraic join operation to tree structures and allow to author new presentations by combining existing presentations and parts of presentations.

The main advantage of multimedia specific approaches based on algebras is that the requested multimedia content is specified as a query in a formal language. However, typically high effort is necessary to learn the algebra and their operators and it is very difficult to apply such a formal approach. Consequently, the presented algebras remain purely academic so far.
References

Alikhateeb F, Baget JF and Evuzenat J 2009 Extending SPARQL with regular expression patterns (for querying RDF).
B. S and A. S 2002 Learning with Kernels, MIT Press.
Benmokhtar R and Huet B 2006 Classifier fusion : combination methods for semantic indexing in video content


Clavel C, Ehrette T and Richard G 2005 Events detection for an audio-based surveillance system. *International Conf. on Multimedia and Expo*.


Copeland G and Maier D 1984 Making smalltalk a database system, pp. 316–325.


DMS-1 2004 Material Exchange Format (MXF) – Descriptive Metadata Scheme-1 SMPTE 380M.


Fitzgibbon AW and Zisserman A 2002 On affine invariant clustering and automatic cast listing in movies
Fluit C, Sabou M and van Harmelen F 2003 Supporting User Tasks through Visualisation of Light-weight
Fogel LJ, Owens AJ and Walsh MJ 1966 Artificial Intelligence through Simulated Evolution. John Wiley & Sons,
New York.
13th International Conference.
Furui S 2001 Digital Speech Processing, Synthesis and Recognition Signal Processing and Communications Series
2nd edition edn. Marcel Dekker, Inc.
G. Stoilos, G. Stamou, V. Tzouvaras, J.Z. Pan, and J. Horrocks 2005 A fuzzy description logic for multimedia
Metokus Deliverable 7.
Gangemi A, Guarino N, Masolo C, Oltramari A and Schneider L 2002 Sweetening Ontologies with DOLCE
13th International Conference on Knowledge Engineering and Knowledge Management (EKAWE02), pp. 166–181,
Siguenza, Spain.
Garcia R and Celma O 2005 Semantic Integration and Retrieval of Multimedia Metadata 9th International Workshop
on Knowledge Markup and Semantic Annotation.
van Harmelen, F.
Geroimenko V and Chen C 2003b Visualizing the Semantic Web — XML-based Internet and Information
Visualization. Springer-Verlag.
Geuvs J, Osenbruggen Jv and Hardman  I. 2005 Requirements for practical multimedia annotation Workshop on
Multimedia and the Semantic Web.
Recognition and Artificial Intelligence 15(1), 9–42.
Gillet O and Richard G 2004 Automatic transcription of drum loops IEEE International Conference on Acoustics,
Speech and Signal Processing (ICASSP), Montréal, Canada.
Gillet O and Richard G 2005 Automatic transcription of drum sequences using audiovisual features. Acoustics,
iis/206 Vol. 3.
Gillet, O, Essid S and Richard G 2007 On the correlation of automatic audio and visual segmentations of music
Giorgos Stoilos, Giorgos Stamou, Vassilis Tzouvaras, Jeff Z. Pan, and Ian Horrocks 2007 Reasoning with very
Goldberg D and Deb K 1991 A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. Urbana
51, 61801–2996.
Grossol BN, Horrocks I, Volz R and Decker S 2003 Description logic programs: Combining logic programs with
N4039.
Guttman A 1984 R-trees: a dynamic index structure for spatial searching Proceedings of ACM SIGMOD
International Conference on Management of Data, pp. 47–57. ACM Press, Boston, USA.
Research 3, 1157–1182.
Haarslev V and Moller R 2003 Racer: A core inference engine for the semantic web 2nd International Workshop on
Halasz F and Schwartz M 1999b The Dexter Hypertext Reference Model. Communications of the ACM 37(2),
30–39.


Harth A, Umbrich J, Hoganas A and Decker S 2007 YARS2: A Federated Repository for Querying Graph Structured Data from the Web ISWC. Springer.


Hausenblas M 2008 Building Scalable and Smart Multimedia Applications on the Semantic Web PhD thesis Graz University of Technology.


Hefflin J 2004 Owl web ontology language use cases and requirements. Technical report, W3C.


Hildebrand M, van Ossenbruggen J and Hardman L 2006 facet!: A Browser for Heterogeneous Semantic Web Repositories ISWC.


Hos SC, Wong LL and Lyu A 2006 Chinese university of hongkong at trecvid 2006: Shot boundary detection and video search TRECVis 2006 Workshop, pp. 76–86 NIST. National Institute of Standards and Technology, Green Auditorium, Gaithersburg, Maryland, USA.


Isaac A and Troncy R 2004 Designing and Using an Audio-Visual Description Core Ontology Workshop on Core Ontologies in Ontology Engineering.


Jaimes A and Chang SF 2000 A conceptual framework for indexing visual information at multiple levels IS&T/SPIE Internet Imaging, vol. 3964.


Kim HG, Moreau N and Sikora T 2005 MPEG-7 Audio and Beyond: Audio Content Indexing and Retrieval; John Wiley & Sons.


Kim Y.S. SW and F. M 2002 Evolutionary model selection in unsupervised learning. 6, 531–556.


Krumhansl CL 1989 Structure and Perception of Electroacoust Sound and Music Nielzen and Oisson (Elsevier, Amsterdam) chapter Why is musical timbre so hard to understand?


Lehane B and O’Connor NE 2004 Action sequence detection in motion pictures The international Workshop on Multidisciplinary Image, Video, and Audio Retrieval and Mining.

Lehane B and O’Connor NE 2006 Movie indexing via event detection 7th International Workshop on Image Analysis for Multimedia Interactive Services, Incheon, Korea, 19-21 April.


Lu C, Drew MS and Au J 2002 An automatic video classification system based on a combination of hmm and video
Luo J, Boustell M and Brown C 2006 Pictures are not taken in a vacuum. Signal Processing Magazine. IEEE 23(2),
101–114.
libraries Proceedings of CASSE'03 Forum at 15th Conference on Advanced Information Systems Engineering.
M. C 2007 Performance evaluators for relevance feedback and classifiers. Technical Report KSpace report k-
WP4.5-20070403, Tlcom ParisTech.
and Systems for Video Technology 11(6), 703–715.
John Wiley and Sons ltd.
Manjunath BS and Ma WY 1996 Texture features for browsing and retrieval of image data. IEEE Trans. Pattern
on Circuits and Systems for Video Technology 11, 703–715.
Manjunath BS, Salembier P and Sikora T 2002b Introduction to MPEG-7: Multimedia content description interface.
Wiley-Interscience.
m.c. schraefel and karger D 2006 The Pathetic Fallacy of RDF 3rd International Semantic Web User Interaction
Workshop (SWUI), Athens, Georgia, USA.
m.c. schraefel, smith D, owens A, russell A, Harris C and Wilson M 2005 The evolving mSpace platform:
leveraging the Semantic Web on the Trail of the Memex Hypertext, pp. 174–183, Szalburg, Austria.
McAdams S 1993 Thinking in Sound: The Cognitive Psychology of Human Audition Oxford University Press chapter
Recognition of Auditory Sound Sources and Events.
McKinney MF and Breebart J 2003 Features for audio and music classification International Symposium on Music
Information Retrieval, pp. 151–158.
Meng A 2006 Temporal Feature Integration for Music Organisation PhD thesis Informatics and Mathematical
Modelling, Technical University of Denmark, DTU.
Creating rich metadata in the tv broadcast archives environment: the PrestoSpace project 2nd International
Conference on Automated Production of Cross Media Content for Multi-channel Distribution, Leeds, UK.
Messing D, van Beek P and Errico J 2001 The mpeg-7 colour structure descriptor: image description using colour
pp. 670–673.
MET 2006 METS Profile Components http://www.loc.gov/standards/mets/profile_docs/
components.html.
Mezaris V, Kompatsiaris I and Strintzis M 2004 Still image segmentation tools for object-based multimedia
Furth (Editor). Springer.
Standards 3(2-3), 211–223.
Miliau S, Pitz M and Ney H 2001 Histogram based normalization in the acoustic feature space, Proc. of ASRU.
Moore B, Glasberg BR and Baer T 1997 A model for the prediction of thresholds, loudness and partial loudness,.
Morris O, Lee M and Constantinides A 1986 Graph theory for image analysis: An approach based on the shortest

MPEG-7 2001a Multimedia Content Description Interface ISO/IEC 15938.
MPEG-7 2001b Multimedia Content Description Interface ISO/IEC 15938.
MPEG-7 2001c Multimedia Content Description Interface Standard No. ISO/IEC 15938.
Nack F and Lindsay AT 1999 Everything you wanted to know about MPEG-7 (Parts I & II). IEEE Multimedia.


Quan D and Karger D 2004 How to Make a Semantic Web Browser 13th International World Wide Web Conference (WWW), New York, USA.

Quilitz B and Leser U 2008 Querying Distributed RDF Data Sources with SPARQL ESWC.


Rodet X and Jaillet F 2001 Detection and modeling of fast attack transients International Computer Music Conference.


Tanghe K, Degroote S and Baets BD 2005 An algorithm for detecting and labeling drum events in polyphonic music, the 2005 MIREX evaluation campaign.
Frequently Occurring First Names and Surnames From the 1990 Census

MediaMill: Semantic Video Search Engine


the krrs group n.d. Description-logic knowledge representation system specification. Technical report, the arpa knowledge sharing effort.
Tomadaki E 2006 Cross-document coreference between different types of collateral texts for films PhD dissertation, Dept. of Computing, University of Surrey.


TVA 2005a Broadcast and online services: Search, select, and rightful use of content on personal storage systems (“tv anytime”); part 2: System description ETSI TS 102 822-2 V1.3.1.

TVA 2005b Broadcast and online services: Search, select, and rightful use of content on personal storage systems (“tv anytime”); part 3: Metadata; subpart 1: Phase 1 metadata schemas ETSI TS 102 822-3-1 V1.3.1.


Ushold M 2003 Where are the semantics in the semantic web?. AI Magazine 24(3), 25–36.


Tomadaki E 2006 Cross-document coreference between different types of collateral texts for films PhD dissertation, Dept. of Computing, University of Surrey.


TVA 2005a Broadcast and online services: Search, select, and rightful use of content on personal storage systems (“tv anytime”); part 2: System description ETSI TS 102 822-2 V1.3.1.

TVA 2005b Broadcast and online services: Search, select, and rightful use of content on personal storage systems (“tv anytime”); part 3: Metadata; subpart 1: Phase 1 metadata schemas ETSI TS 102 822-3-1 V1.3.1.


Ushold M 2003 Where are the semantics in the semantic web?. AI Magazine 24(3), 25–36.


Vassiliou A 2006 Film content analysis: a text-based approach PhD dissertation, Dept. of Computing, University of Surrey.


VRA 2007 VRA Core http://www.vraweb.org/projects/vracore4-


Morgan Kaufmann, New York City, USA.


ACM, New York, NY, USA.


Fl. Lauderdale, Florida, USA.


Italy.


Zhang R, Ooi BC and Tan KL 2004 Making the pyramid technique robust to query types and workloads 20th IEEE ICDE, pp. 313–324, Boston, USA.