Methods and Tools for Developing Ontology-Based Data Management Solutions

Designing OWL Ontologies: from UML to Graphol

Domenico Lembo, Valerio Santarelli, Domenico Fabio Savo

SEMANTICS 2018
Vienna, Austria, September 11, 2018
Consider the following requirements ...

**Requirements:** We are interested in building a software application to manage filmed scenes for realizing a movie, by following the so-called “Hollywood Approach”.

Every **scene** is described by a text in natural language (a string) and is filmed from different positions (at least one), each of this is called a **setup**.

Every setup is characterized by a code (a string) and a text in natural language where the photographic parameters are noted (e.g., aperture, exposure, focal length, filters, etc.). Note that a setup is related to a single scene.

For every setup, several **takes** may be filmed (at least one). Every take is characterized by a (positive) natural number, a real number representing the number of meters of film that have been used for shooting the take. Note that a take is associated to a single setup.

Scenes are divided into **internals** and **externals**, which are filmed in a **location**. Locations are characterized by a name (a string) and the address of the location (a string).

**Write a precise specification of this domain using any formalism you like.**
Solution 1: use conceptual modeling diagrams (UML)!!!

```plaintext
Scene
  description: string

Internal

External

{complete, disjoint}

Setup
  code: string
  ph_param: string

Take
  nbr: integer
  meters: real

1..1

\text{stp}_\text{for scn}

1..*

\text{tk}_\text{of stp}

1..1

Location
  name: string
  address: string

0..*

\text{located}

1..1
```
Solution 1: use conceptual modeling diagrams (discussion)

**Good points:**
- Easy to generate (it’s the standard in software design)
- Easy to understand for humans
- Well disciplined, well-established methodologies available

**Bad points:**
- No precise semantics
- Verification (or better validation) done informally by humans
- Machine incomprehensible (because of lack of formal semantics)
- Automated reasoning out of question
- Limited expressiveness
Solution 2: use logic!!!

Alphabet:

Scene(x), Setup(x), Take(x), Internal(x), External(x), Location(x), stp_for_scn(x, y), located(x, y), ...

Axioms:

∀x, y. description(x, y) → Scene(x) ∧ Text(y)
∀x, y. code(x, y) → Setup(x) ∧ String(y)
∀x, y. ph_param(x, y) → Setup(x) ∧ Text(y)
∀x, y. nbr(x, y) → Take(x) ∧ Integer(y)
∀x, y. meters(x, y) → Take(x) ∧ Real(y)
∀x, y. name(x, y) → Location(x) ∧ String(y)
∀x, y. address(x, y) → Location(x) ∧ String(y)
∀x. Setup(x) → (1 ≤ #{y | code(x, y)}) ≤ 1
∀x. Take(x) → (1 ≤ #{y | nbr(x, y)}) ≤ 1
...
Solution 2: use logic (discussion)

**Good points:**
- Precise semantics
- Formal verification
- Machine comprehensible
- Virtually unlimited expressiveness

**Bad points:**
- Difficult to generate
- Difficult to understand for humans
- No well-established methodologies available
- Automated reasoning may be impossible
Solution 3: mix them!!!

These two approaches seem orthogonal, but they can in fact be integrated!

Basic idea:
- assign formal semantics to constructs of the conceptual design diagrams;
- use conceptual design diagrams as usual, taking advantage of methodologies developed for them in Software Engineering;
- read diagrams as logical theories when needed, i.e., for formal understanding, verification, automated reasoning, etc.
Important point: by using conceptual modeling diagrams one gets logical theories of a specific form.

- One gets limited (or better, well-disciplined) expressiveness
- One can exploit the particular form of the corresponding logical theory to simplify reasoning, hopefully getting:
  - decidability
  - reasoning procedures that match intrinsic computational complexity
We illustrate what we get from integrating logic with conceptual modeling diagrams.

- We can keep using a **graphical model**, such as UML Class Diagrams, for conceptual modeling diagrams
- But we formally assign semantics to the graphical construct in **Logic**
- Specifically we use a **Description Logic**, such as OWL 2 or DL-Lite (OWL2 QL), to understand the **computational properties of reasoning**.

Our key tool is **Graphol** which gives us a direct graphical representation of OWL 2 ontologies.
**Graphol** is a graphical language developed at Sapienza to facilitate use of ontologies in industrial settings.

- Looks similar to UML class diagrams and entity-relationship diagrams
- However is a graphical counterpart of full OWL 2!
- Nice fragments capture $DL-Lite_A$ and OWL 2 QL.
**Graphol** is a graphical language developed at Sapienza to facilitate use of ontologies in industrial settings.

- Looks similar to UML class diagrams and entity-relationship diagrams
- However is a graphical counterpart of full OWL 2!
- Nice fragments capture $DL$-$Lite_A$ and OWL 2 QL.
Graphol is a graphical language developed at Sapienza to facilitate use of ontologies in industrial settings.

- Looks similar to UML class diagrams and entity-relationship diagrams
- However is a graphical counterpart of full OWL 2!
- Nice fragments capture $DL$-$Lite_A$ and OWL 2 QL.
**Graphol** is a graphical language developed at Sapienza to facilitate use of ontologies in industrial settings.

- Looks similar to UML class diagrams and entity-relationship diagrams
- However is a graphical counterpart of full OWL 2!
- Nice fragments capture $DL-Lite_A$ and OWL 2 QL.
Solution 3: use Graphol!!!

(We left out some attributes for simplicity.)
Solution 3: use Graphol!!!

The previous Graphol diagram corresponds to the following OWL 2 ontology:

```owl
Declaration(Class(movies:External))
Declaration(Class(movies:Internal))
Declaration(Class(movies:Location))
Declaration(Class(movies:Scene))
Declaration(Class(movies:Setup))
Declaration(Class(movies:Take))
Declaration(ObjectProperty(movies:located))
Declaration(ObjectProperty(movies:stp_for_scn))
Declaration(ObjectProperty(movies:tk_of_stp))
Declaration(DataProperty(movies:meters))
Declaration(DataProperty(movies:nbr))
FunctionalObjectProperty(movies:located)
ObjectPropertyDomain(movies:located movies:Location)
ObjectPropertyRange(movies:located movies:External)
FunctionalObjectProperty(movies:stp_for_scn)
ObjectPropertyDomain(movies:stp_for_scn movies:Setup)
ObjectPropertyDomain(movies:tk_of_stp movies:Take)
ObjectPropertyRange(movies:tk_of_stp movies:Setup)
DataPropertyDomain(movies:meters movies:Take)
DataPropertyRange(movies:meters owl:real)
FunctionalDataProperty(movies:nbr)
DataPropertyDomain(movies:nbr movies:Take)
DataPropertyRange(movies:nbr xsd:integer)
EquivalentClasses(movies:External ObjectSomeValuesFrom(ObjectInverseOf(movies:located) owl:Thing))
DisjointClasses(movies:External movies:Internal)
EquivalentClasses(movies:Location ObjectSomeValuesFrom(movies:located owl:Thing))
EquivalentClasses(movies:Scene ObjectUnionOf(movies:External movies:Internal))
EquivalentClasses(movies:Scene ObjectSomeValuesFrom(ObjectInverseOf(movies:stp_for_scn) owl:Thing))
EquivalentClasses(movies:Setup ObjectSomeValuesFrom(movies:stp_for_scn owl:Thing))
EquivalentClasses(movies:Setup ObjectSomeValuesFrom(ObjectInverseOf(movies:tk_of_stp) owl:Thing))
EquivalentClasses(movies:Take ObjectSomeValuesFrom(movies:tk_of_stp owl:Thing))
EquivalentClasses(movies:Take DataSomeValuesFrom(movies:meters rdfs:Literal))
EquivalentClasses(movies:Take DataSomeValuesFrom(movies:nbr rdfs:Literal))
```

(We left out some attributes for simplicity.)
Every element in the **alphabet of the ontology** has a graphical representation:

- **Class**  (rectangle)
- **Object Property**  (diamond)
- **Data Property**  (circle)
- **Datatype**  (rounded-corner rectangle)
Complex expressions are defined by combining atomic elements through operators.

Every operator (and, or, not, etc.) has its symbol and takes some arguments which are given as inputs through dashed arrows of the following form:

Example:
The **OR** operator, which “constructs” the (complex) concept denoting the **union of its arguments**, is represented as follows:
Obviously, complex expressions (in turn constructed through other operators) can be arguments of operators.

**Example:**

In this example the complex expression corresponding to (Student OR Worker) is used as input of the **AND** operator.

The Graphol expression in figure corresponds to the expression:

```plaintext
((Student OR Worker) AND Athletic)
```
In Graphol, the inclusion axiom is represented through a **solid arrow** from the subsumee to the subsumer (pairs of concepts, roles, or attributes).

**Example:**

The Graphol diagram in figure corresponds to the following inclusion axiom:

Student **ISA** Person
Example:

The following Graphol diagram represents the following inclusion axiom:

$$\text{NOT}(\text{Student OR Worker}) \text{ ISA Unemployed}$$

That is, all individuals that are not students or workers are unemployed.
Graphol: generalization

Example:

A *generalization* in Graphol is represented as shown in figure.

The Graphol diagram corresponds to the following inclusion axiom:

$$(\text{Student OR Worker}) \ ISA \ \text{Person}$$
Graphol: complete generalization

**Example:**

Complete generalization are represented by specifying the equivalence.

The Graphol diagram corresponds to the following axiom:

\[(\text{Student OR Worker}) \equiv \text{Active\_Person}\]

That is, \text{Active\_Person} is equivalent to the union of \text{Student} and \text{Worker},
Graphol: complete and disjoint generalization

Example:

Complete and disjoint generalizations are represented as follows.

The Graphol diagram corresponds to the following axioms:

\[(\text{Man OR Woman}) \equiv \text{Person}\]
\[
\text{Man ISA NOT(Woman)}
\]
Graphol: a useful shortcut for denoting disjoint OR

To ease the drawing of complete and disjoint generalizations, Graphol allows the designer to use a shortcut for denoting the **disjoint OR**

```
[Diagram]
```
denotes **disjoint OR**

The following two diagrams are equivalent.
Inclusion axioms can also be specified on **Object Properties** and **Data Properties** as follows.

- **Object Property inclusion axiom**

- **Data Property inclusion axiom**
At the *extensional* level an object property is a **set of pairs of objects**.

\[ o_1, o_2, o_3, \ldots \text{ denote objects of an interpretation.} \]

The object property is “oriented” from the so-called object property *domain* to the object property *range*, i.e., each instance is in fact a labeled pair: e.g., \((\text{dom:}o_1, \text{ran:}o_6)\); \((\text{dom:}o_2, \text{ran:}o_7)\), etc.

In the following, we assume that each pair \((o,o')\) is labeled \((\text{dom:}o, \text{ran:}o')\).
The object property **domain** is in fact a **complex concept** denoting the set of objects occurring in domain position in the property instances.

**Example:** the instances of the domain of **works_for** are \{o1, o2, o3\}

To represent it in Graphol we use the operator that takes in input a property and returns its **domain**.

**Example:**
The object property range is in fact a complex concept denoting the set of objects occurring in range position in the property instances.

**Example:** the instances of the range of works_for are \{o5, o6, o8\}

To represent it in Graphol we use the operator that takes in input a property and returns its range.

**Example:**
Graphol: object property typing

Specifies the “type” of objects that can be instances of the domain (resp. range) of the object property.

**Example:**

This is actually an **inclusion axiom**:

\[
\text{DOMAIN}(\text{works_for}) \ ISA \ \text{Person}
\]

For specifying the “type” of objects that can be instances of the range of the object property we will use the operator
Graphol: mandatory participation

Imposes that every object which is instance of a concept (both atomic or complex) is also instance of the domain (resp. range) of an object property.

**Example:**

The above Graphol diagram corresponds to the following inclusion axiom:

Worker ISA DOMAIN(works_for)
Data properties denote binary relations between (instances of) concepts and value-domains. **They behave similarly to object properties.**
Global functionality and inverse global functionality over object properties and data properties can be specified in Graphol as follows:

- **born_in** is functional
- **is_father_of** is inverse functional
- **married_with** is both functional and inverse functional
- **name** is functional
### Other Graphol Expressions

<table>
<thead>
<tr>
<th>Expression</th>
<th>DL syntax</th>
<th>Graphol syntax</th>
<th>Expression</th>
<th>DL syntax</th>
<th>Graphol syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic Concept</td>
<td>$A$</td>
<td>$A$</td>
<td>Atomic Role</td>
<td>$P$</td>
<td>$P$</td>
</tr>
<tr>
<td>Atomic Attribute</td>
<td>$U$</td>
<td></td>
<td>Atomic Value-domain</td>
<td>$T$</td>
<td></td>
</tr>
<tr>
<td>Domain restriction on role</td>
<td>$\exists R.C \forall R.C \geq xR.C \leq yR.C$</td>
<td>$\exists R.C \forall R.C \geq xR.C \leq yR.C$</td>
<td>Range restriction on role</td>
<td>$\exists R^{-}.C \forall R^{-}.C \geq xR^{-}.C \leq yR^{-}.C$</td>
<td>$\exists R^{-}.C \forall R^{-}.C \geq xR^{-}.C \leq yR^{-}.C$</td>
</tr>
<tr>
<td>Domain restriction on attribute</td>
<td>$\exists V.F \forall V.F \geq xV.F \leq yV.F$</td>
<td>$\exists V.F \forall V.F \geq xV.F \leq yV.F$</td>
<td>Range existential restriction on attribute</td>
<td>$\exists V^{-}$</td>
<td>$\exists V^{-}$</td>
</tr>
<tr>
<td>Concept Intersection</td>
<td>$C_1 \cap C_2$</td>
<td>$C_1 \cap C_2$</td>
<td>Concept Union</td>
<td>$C_1 \sqcup C_2$</td>
<td>$C_1 \sqcup C_2$</td>
</tr>
<tr>
<td>Value-domain Intersection</td>
<td>$F_1 \cap F_2$</td>
<td>$F_1 \cap F_2$</td>
<td>Value-domain Union</td>
<td>$F_1 \sqcup F_2$</td>
<td>$F_1 \sqcup F_2$</td>
</tr>
<tr>
<td>Concept Complement</td>
<td>$\neg C$</td>
<td>$\neg C$</td>
<td>Role Complement</td>
<td>$\neg R$</td>
<td>$\neg R$</td>
</tr>
<tr>
<td>Attribute Complement</td>
<td>$\neg U$</td>
<td>$\neg U$</td>
<td>Value-domain Complement</td>
<td>$\neg F$</td>
<td>$\neg F$</td>
</tr>
<tr>
<td>Role Inverse</td>
<td>$R^{-}$</td>
<td>$R^{-}$</td>
<td>Role Chain</td>
<td>$R_1 \circ R_2$</td>
<td>$R_1 \circ R_2$</td>
</tr>
<tr>
<td>Concept One-of</td>
<td>${a, b, c}$</td>
<td>${a, b, c}$</td>
<td>Value-domain One-of</td>
<td>${1&quot;, &quot;2&quot;, &quot;3}$</td>
<td>${1&quot;, &quot;2&quot;, &quot;3}$</td>
</tr>
<tr>
<td>Axiom</td>
<td></td>
<td></td>
<td>Axiom</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concept inclusion</td>
<td>$C_2 \sqsubseteq C_1$</td>
<td>$C_1 \sqsubseteq C_2$</td>
<td>Role inclusion</td>
<td>$R_2 \sqsubseteq R_1$</td>
<td>$R_1 \sqsubseteq R_2$</td>
</tr>
<tr>
<td>Concept disjunction</td>
<td>$C_2 \sqsubseteq \neg C_1$</td>
<td>$C_1 \sqsubseteq C_2$</td>
<td>Role disjunction</td>
<td>$R_2 \sqsubseteq \neg R_1$</td>
<td>$R_1 \sqsubseteq R_2$</td>
</tr>
<tr>
<td>Role domain specification</td>
<td>$\exists R \sqsubseteq C$</td>
<td>$C \sqsubseteq R$</td>
<td>Role range specification</td>
<td>$\exists R^{-} \sqsubseteq C$</td>
<td>$C \sqsubseteq R$</td>
</tr>
</tbody>
</table>
End Part II