Methods and Tools for Developing Ontology-Based Data Management Solutions

Designing OWL Ontologies: from UML to Graphol

Domenico Lembo, Valerio Santarelli, **Domenico Fabio Savo**

OBDA SYSTEMS

SAPIENZA UNIVERSITÀ DI ROMA

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**.

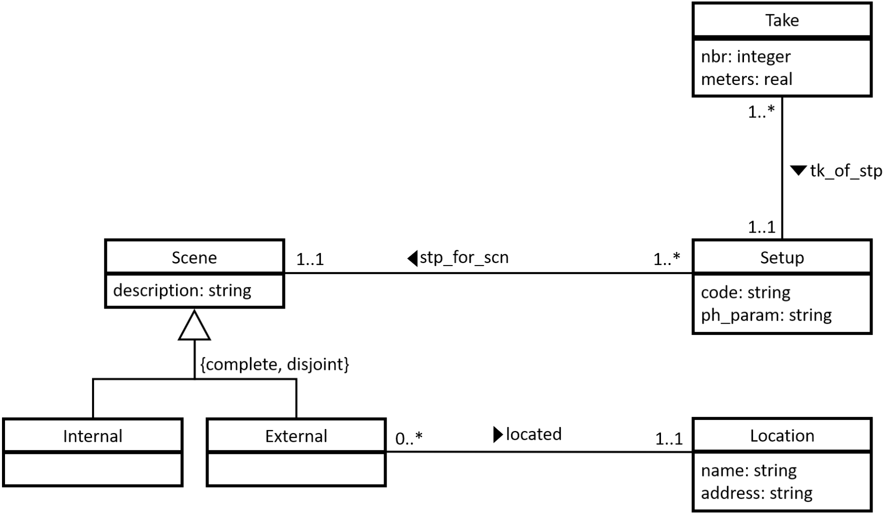
Every setup is characterized by a code (a string) and a text in natural language where the photographic parameters are noted (e.g., aperture, exposure, focal length, filters, etc.). Note that a setup is related to a single scene.

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)!!!



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:

$Scene(x), Setup(x), Take(x), Internal(x), External(x), Location(x), stp\_for\_scn(x, y), located(x, y), \ldots$

Axioms:

$\forall x, y.\, description(x, y) \rightarrow Scene(x) \wedge Text(y)$

$\forall x, y.\, code(x, y) \rightarrow Setup(x) \wedge String(y)$
$\forall x, y.\, ph\_param(x, y) \rightarrow Setup(x) \wedge Text(y)$

$\forall x, y.\, nbr(x, y) \rightarrow Take(x) \wedge Integer(y)$
$\forall x, y.\, meters(x, y) \rightarrow Take(x) \wedge Real(y)$

$\forall x, y.\, name(x, y) \rightarrow Location(x) \wedge String(y)$
$\forall x, y.\, address(x, y) \rightarrow Location(x) \wedge String(y)$

$\forall x.\, Setup(x) \rightarrow (1 \leq \sharp\{y \mid code(x, y)\} \leq 1)$
$\forall x.\, Take(x) \rightarrow (1 \leq \sharp\{y \mid nbr(x, y)\} \leq 1)$
$\ldots$

$\forall x, y.\, stp\_for\_scn(x, y) \rightarrow Setup(x) \wedge Scene(y)$
$\forall x, y.\, tk\_of\_stp(x, y) \rightarrow Take(x) \wedge Setup(y)$
$\forall x, y.\, located(x, y) \rightarrow External(x) \wedge Location(y)$

$\forall x.\, Setup(x) \rightarrow 1 \leq \sharp\{y \mid stp\_for\_scn(x, y)\} \leq 1$
$\forall y.\, Scene(y) \rightarrow 1 \leq \sharp\{x \mid stp\_for\_scn(x, y)\}$
$\forall x.\, Take(x) \rightarrow 1 \leq \sharp\{y \mid tk\_of\_stp(x, y)\} \leq 1$
$\forall x.\, Setup(y) \rightarrow 1 \leq \sharp\{x \mid tk\_of\_stp(x, y)\}$
$\forall x.\, External(x) \rightarrow 1 \leq \sharp\{y \mid located(x, y)\} \leq 1$

$\forall x.\, Internal(x) \rightarrow Scene(x)$
$\forall x.\, External(x) \rightarrow Scene(x)$
$\forall x.\, Internal(x) \rightarrow \neg External(x)$
$\forall x.\, Scene(x) \rightarrow Internal(x) \vee External(x)$

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.

Solution 3: mix them!!! (cont.)

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

Solution 3: mix them!!! (cont.)

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

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 $\textit{DL-Lite}_{\mathcal{A}}$ and OWL 2 QL.

Graphol

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 $DL\text{-}Lite_{\mathcal{A}}$ and OWL 2 QL.

Graphol

Graphol is a graphical language developed at Sapienza to facilitate use of ontologies in industrial settings.

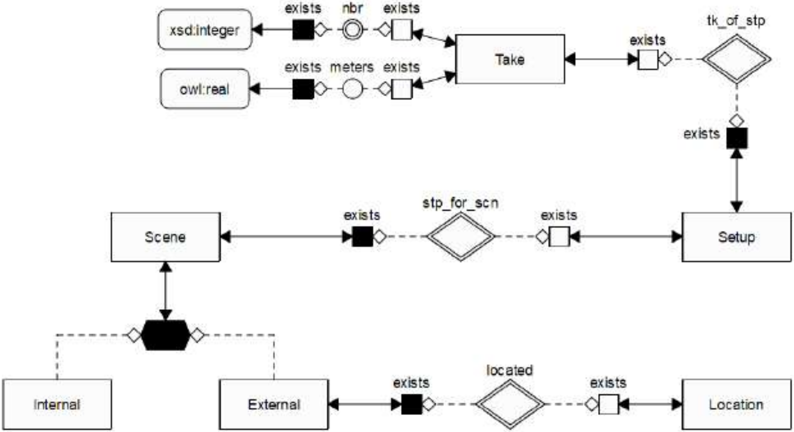
- Looks similar to UML class diagrams and entity-relationship diagrams
- However is a graphical counterpart of full OWL 2!
- Nice fragments capture $DL\text{-}Lite_{\mathcal{A}}$ and OWL 2 QL.

Graphol

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 $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ 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:

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)
ObjectPropertyRange(movies:stp_for_scn movies:Scene) FunctionalObjectProperty(movies:tk_of_stp)
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))

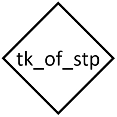
(We left out some attributes for simplicity.)

Graphol: symbols for atomic elements

Every element in the **alphabet of the ontology** has a graphical representation

– Class (rectangle)

– Object Property (diamond)

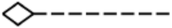


– Data Property (circle)

– Datatype (rounded-corner rectangle)

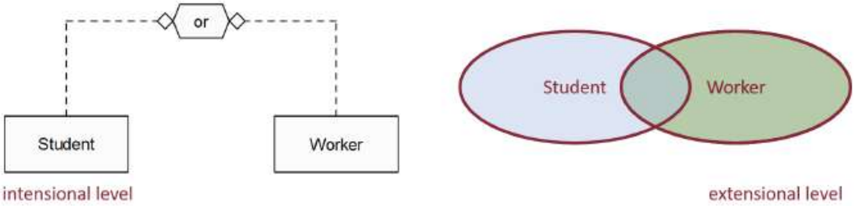
Graphol: operators for complex expressions

- 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:



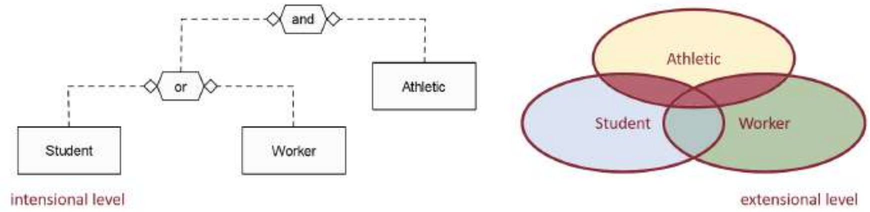
Obviously, complex expressions (in turn constructed through other operators) can be arguments of operators.

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:

((Student OR Worker) AND Athletic)

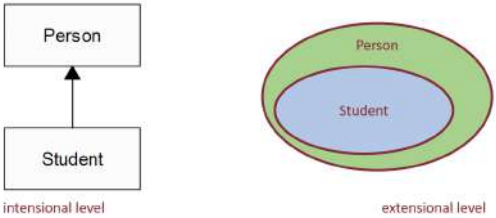


In Graphol, the inclusion axiom is represented through a **solid arrow** from the subsumee to the subsumer (pairs of concepts, roles, or attributes).

Example:

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

Student ISA Person

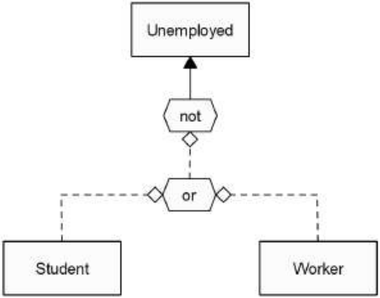


Example:

The following Graphol diagram represents the following inclusion axiom:

NOT(Student OR Worker) ISA Unemployed

That is, all individuals that are not students or workers are unemployed



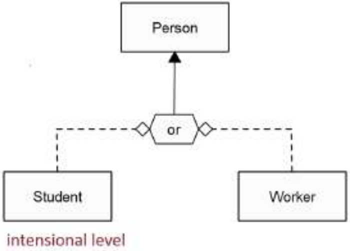
Graphol: generalization

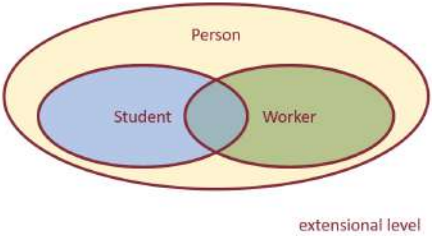
Example:

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

The Graphol diagram corresponds to the following inclusion axiom:

(Student OR Worker) ISA Person





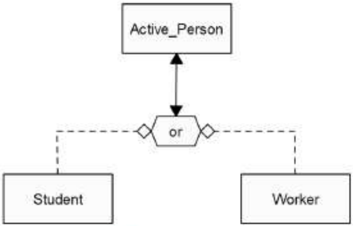
Example:

Complete generalization are represented by specifying the equivalence.

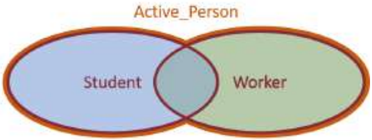
The Graphol diagram corresponds to the following axiom:

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

That is, Active_Person is **equivalent** to the **union** of Student and Worker,



intensional level



extensional level

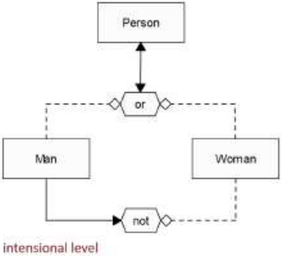
Graphol: complete and disjoint generalization

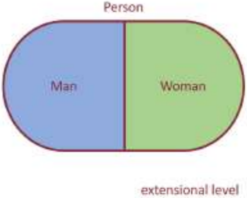
Example:

Complete and disjoint generalizations are represented as follows.

The Graphol diagram corresponds to the following axioms:

(Man OR Woman) ≡ Person
Man ISA NOT(Woman)



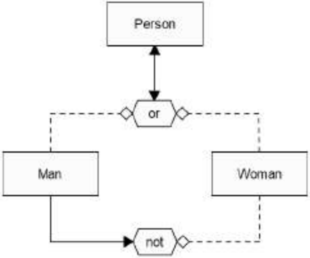


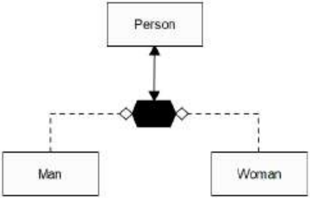
Graphol: a useful shortcut for denoting disjoint OR

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

denotes **disjoin OR**

The following two diagrams are equivalent.

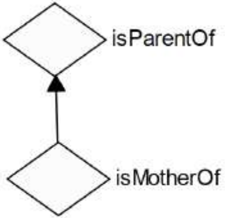




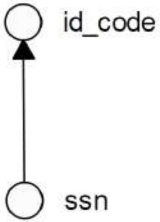
Graphol: more on inclusion axioms

Inclusion axioms can also be specified on **Object Properties** and **Data Properties** as follows.

– Object Property inclusion axiom

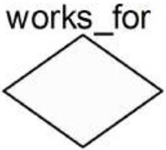


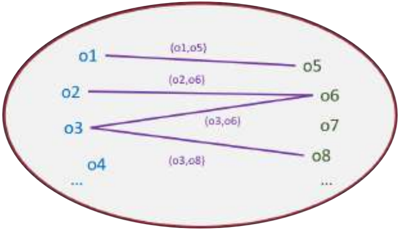
– Data Property inclusion axiom



Note: semantics of object properties

At the **extensional** level an object property is a **set of pairs of objects**.





o1, o2, o3,. . . denote objects of an **interpretation**.

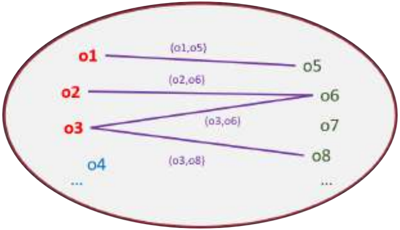
The object property is "oriented" from the so-called object property domain to the object property range, , i.e., each instance is in fact a labeled pair: e.g., (dom:o1, ran:o6); (dom:o2, ran:o7), etc.

In the following, we assume that each pair **(o,o')** is labeled **(dom:o,ran:o')**.

Graphol: operator for object property domain

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

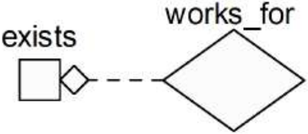
Example: the instances of the domain of **works_for** are {o1, o2, o3 }



To represent it in Graphol we use the operator 

that takes in input a property and returns its domain.

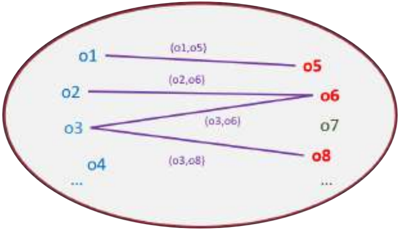
Example:



Graphol: operator for object property range

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

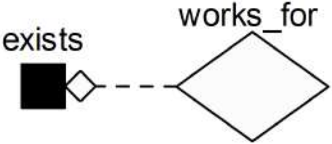
Example: the instances of the range of **works_for** are {o5, o6, o8 }



To represent it in Graphol we use the operator 

that takes in input a property and returns its range.

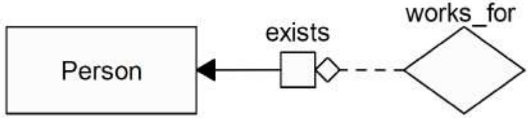
Example:



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**:

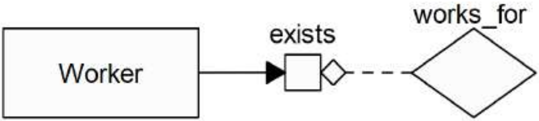
DOMAIN(works_for) ISA Person

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.

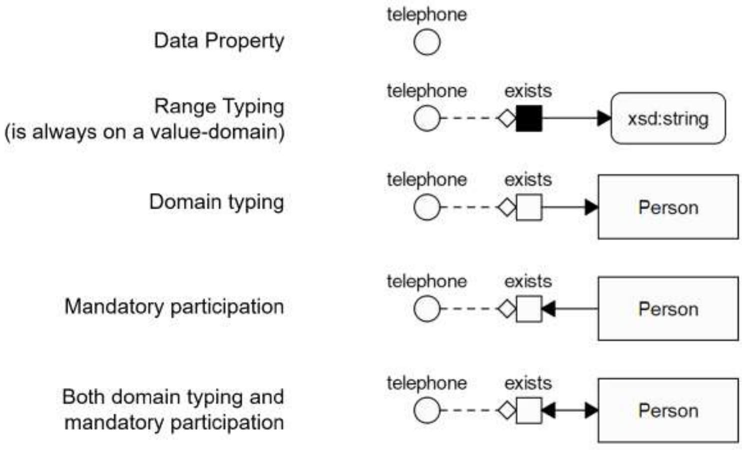
Example:



The above Graphol diagram corresponds to the following **inclusion axiom**:

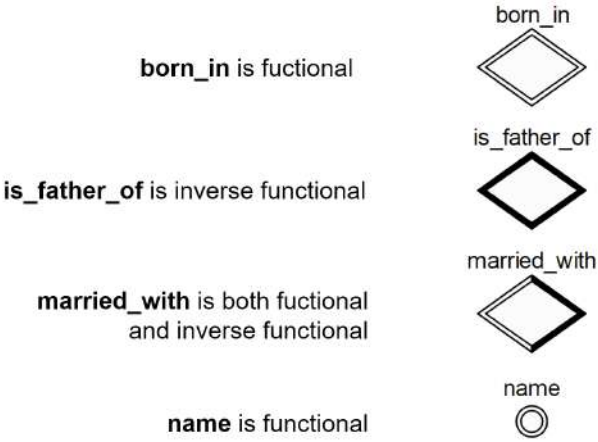
Worker ISA DOMAIN(works_for)

Data properties denote binary relations between (instances of) concepts and value-domains. **They behave similarly to object properties.**

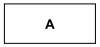
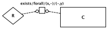
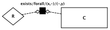
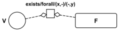
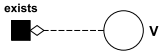
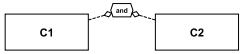
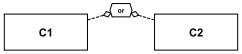
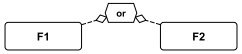
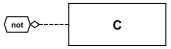
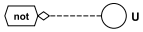
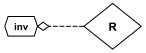
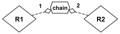
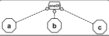


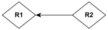
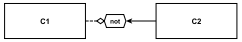
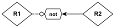
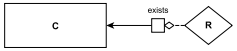
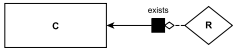
Graphol: global functionality

Global functionality and **inverse global functionality** over object properties and data properties can be specified in Graphol as follows:



Other Graphol Expressions

Expression	DL syntax	Graphol syntax	Expression	DL syntax	Graphol syntax
Atomic Concept	A		Atomic Role	P	
Atomic Attribute	U		Atomic Value-domain	T	
Domain restrictionon role	$\exists R.C \quad \forall R.C$ $\geq xR.C \quad \leq yR.C$		Range restrictionon role	$\exists R^{-}.C \quad \forall R^{-}.C$ $\geq xR^{-}.C \quad \leq yR^{-}.C$	
Domain restrictionon attribute	$\exists V.F \quad \forall V.F$ $\geq xV.F \quad \leq yV.F$		Range existentialrestriction on attribute	$\exists V^{-}$	
Concept Intersection	$C_1 \sqcap C_2$		Concept Union	$C_1 \sqcup C_2$	
Value-domain Intersection	$F_1 \sqcap F_2$		Value-domain Union	$F_1 \sqcup F_2$	
Concept Complement	$\neg C$		Role Complement	$\neg R$	
Attribute Complement	$\neg U$		Value-domain Complement	$\neg F$	
Role Inverse	R^{-}		Role Chain	$R_1 \circ R_2$	
Concept One-of	$\{a, b, c\}$		Value-domain One-of	{"1", "2", "3"}	

Axiom	DL syntax	Graphol syntax	Axiom	DL syntax	Graphol syntax
Concept inclusion	$C2 \sqsubseteq C1$		Role inclusion	$R2 \sqsubseteq R1$	
Concept disjunction	$C2 \sqsubseteq \neg C1$		Role disjunction	$R2 \sqsubseteq \neg R1$	
Role domain specification	$\exists R \sqsubseteq C$		Role range specification	$\exists R^{-} \sqsubseteq C$	

End Part II