SIMANTICS DATA MODEL AND ONTOLOGY LANGUAGE

HANNU NIEMISTÖ AND ANTTI VILLBERG

Contents

1. Introduction

Simantics data model is based on RDF-like semantic graphs. The data model is semi-structural i.e. there is no separation of concepts and instance data: all concepts are modeled also using graph representation.

This document is organized as follows:

- Sections 2–7 define the semantics for the basic queries.
- Sections 8–11 define when the semantic graph is valid and mechanisms to add new integrity constraints.
- Sections 12–17 define concepts whose exact meaning depends on the Simantics platform implementation.
- Section 18 defines a graphical language for specifying ontologies.
- Section 19 describes common ontology development conventions.

2. Semantic graphs

2.1. **Informative definition.** *Semantic graph* is a graph whose nodes are called *resources* and edges *statements*. In addition, some resources are associated with *values*. A statement is defined by three resources: *subject*, *predicate* and *object*. Each statement of the semantic graph can be thought as a statement of a fact in natural language, for example, (PI123, **PartOf**, HE456) can be read as "PI123 is a part of HE456." In graph theoretic terms, a statement is an edge that goes from its subject to its object and its predicate is the color of the edge.

2.2. **Formal definition.** Let $\mathcal{R}$ be the set of all resources. Resources do not have any internal structure in this specification, only identity. Different implementations may choose to represent resources in different ways, for example as integers. Let $\mathcal{B}^{<\omega}$ be the set of all finite byte sequences. We define a semantic graph $\mathcal{G}$ as a pair $(S^{\mathcal{G}}, V^{\mathcal{G}})$, where $S^{\mathcal{G}}$ is a set of statements $S^{\mathcal{G}} \subseteq \mathcal{R}^3$ and $V^{\mathcal{G}}$ is a partial function $V^{\mathcal{G}} \colon \mathcal{R} \to \mathcal{B}^{<\omega}$ that associates resources with byte sequence encoding the value attached to the resource.

We abbreviate $(a, b, c) \in S^{\mathcal{G}}$ as $\langle a, b, c\rangle^{\mathcal{G}}$. For each resource b, we define a binary relation

$$b^{\mathrm{PRel},\mathcal{G}} = \{(a, c) \mid \langle a, b, c\rangle^{\mathcal{G}}\}$$

and partial unary function

$$b^{\mathrm{PFun},\mathcal{G}}(a) = c, \text{ if there exists a unique } c \text{ such that } \langle a, b, c\rangle^{\mathcal{G}}.$$

For the rest of the specification, we fix a semantic graph $\mathcal{G}$ and drop it usually from superscripts.

3. Values

3.1. **Informative definition.** Databoard specification [2] defines *data values*, *data types* and their binary and ascii encodings. Data types are also data values of data type `DataType`. Other data types used in this specification are 32-bit signed integers (`Integer`) and Unicode strings (`String`).

The values are attached to resources encoded as byte sequences. We specify in section 7 how the data type of the value is defined.

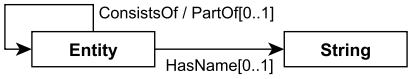
3.2. **Formal definition.** Let $\mathcal{V}$ be the set of all values and $\mathcal{T}$ the set of all data types as defined in [2]. Let $\mathrm{enc} \colon \mathcal{T} \times \mathcal{V} \to \mathcal{B}^{<\omega}$ be the binary encoding function for data values such that if T is the data type of v then $\mathrm{enc}(T, v)$ is the byte sequence that encodes v.

We can now define a partial function $\mathrm{Val}_T^{\mathcal{G}} \colon \mathcal{R} \to \mathcal{V}$ that gives a value attached to a resource, if its type is T. It is defined as

$$\mathrm{Val}_T^{\mathcal{G}}(r) = v \iff V^{\mathcal{G}}(r) = s \text{ and } \mathrm{enc}(T, v) = s.$$

Note that $\mathrm{Val}_T^{\mathcal{G}}(r)$ may be undefined for two reasons: there is no value attached to r or the byte sequence encoding the value is not compatible with data type T.

4. Unique resource identifiers



4.1. **Informative definition.** *Unique resource identifier* (URI) is a standard way to refer to a resource. In Simantics, URIs are encoded with three resources: **RootLibrary**, **PartOf** and **HasName**. The URI of **RootLibrary** is `http:/`. If resource c is a part of p (i.e. $\langle c, \mathbf{PartOf}, p\rangle$) and the URI of p is U, then the URI of c is formed by concatenating U, the character '`/`' and URI-encoded name of c. The *name* of a resource is the string value of the literal attached to the resource with **HasName**-relation.

All resource names in this document, excluding **RootLibrary**, refer to resources whose URIs are formed by appending the name to

`http://www.simantics.org/Layer0-1.0/`.

For example, the URI of the resource **PartOf** is

`http://www.simantics.org/Layer0-1.0/PartOf`.

4.2. **Formal definition.** The URI of a resource is defined formally by the following recursive function

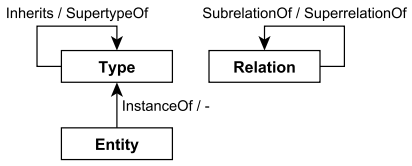
$$\mathrm{URI}(r) = \begin{cases} \text{"}\texttt{http}:/\text{"} & \text{if } c = \mathbf{RootLibrary}, \\ \mathrm{URI}(\mathbf{PartOf}^{\mathrm{PFun}}(r)) + \text{"}/\text{"} + & \text{otherwise} \\ \qquad \mathrm{encode}(\mathrm{Val}_{\texttt{String}}(\mathbf{HasName}^{\mathrm{PFun}}(r))) & \end{cases}$$

Function 'encode' percent-encodes characters that are illegal or reserved in URIs as specified in [1].

A resource doesn't have an URI if any of the partial functions used in the definition is undefined. Additionally a resource that is on a cycle of **PartOf** relation don't have an URI.

Two resources may have the same URI, if they have the same name and the same parent. This kind of semantic graph is invalid.

5. Instantiation and inheritance



5.1. **Informative definition.** Both *types* and *relations* have their own *subtype* and *subrelation* hierarchies. Immediate supertypes are related with **Inherits** and immediate supertypes with **SubrelationOf**. A type t inherits another type t' if there is a chain of **Inherits** statements from t to t'. Subrelation hierarchy is defined similarly.

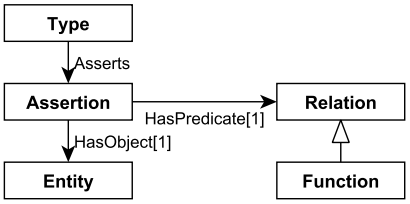
A resource is defined to be an *instance* of some type with **InstanceOf** relation. If a resource is instance of a type, it is also an instance of all of its supertypes. Additionally a resource inherits all types from its supertypes or superrelations.

We have not tried to form an exact philosophical view about what types and instantiation are. However, types form the basis for interpretating the content in the semantic graphs.

5.2. **Formal definition.** We write $t <_T t'$ if t inherits t' and $r <_R r'$ if r is a subrelation of r'. Relation $<_T$ is the transitive closure of $\mathbf{Inherits}^{\mathrm{PRel}}$ and $<_R$ is the transitive closure of $\mathbf{SubrelationOf}^{\mathrm{PRel}}$. We write $i : t$ when i is an instance of type t. The relation is formally defined as

$$i : t \iff \exists i', t' \in \mathcal{R}.(i = i' \vee i <_T i' \vee i <_R i') \wedge \langle i', \mathbf{InstanceOf}, t' \rangle \wedge t' \leq_T t$$

6. Assertions and derived statements



6.1. **Informative definition.** In addition to the statements that are directly asserted (i.e. belong to the set $S^{\mathcal{G}}$), a type can also assert statements to its instances. Together these statements form the set of *derived statements*. If a resource r is an instance of a type that **Asserts** an assertion and the assertion **HasPredicate** p and **HasObject** o, then we derive a statement (r, p, o). So all asserted statements are copied from types to their instances. This basic mechanism has one exception: if asserted relation is functional and it is defined in some subtype or in the instance, the statement is not copied to the instance.

6.2. **Formal definition.** We define first the set of all statements asserted in types:

$$A = \{(t, p, o) \in \mathcal{R}^3 \mid \langle t, \mathbf{Asserts}, a \rangle, \langle a, \mathbf{HasPredicate}, p \rangle, \langle a, \mathbf{HasObject}, o \rangle\}.$$

Let $\perp$ be a special unused resource. It is used to mark statements that are defined directly in resources. Let $<_T^*$ be the extension of $<_T$ such that $\perp <_T^* t$ for all resources $t \neq \perp$. Define $(t, p, o) \prec (t', p', o')$ if $t <_T^* t'$ and there exists a resource p_0 such that $p_0 : \mathbf{Function}$, $p \leq_R p_0$ and $p' \leq_R p_0$. If $s \prec s'$ we say that s inhibits s'.

The idea that assertions of functions in subtypes inhibit the assertions in supertypes is formalized by the following function:

$$\mathrm{filter}(X) = \{(t, p, o) \in X \mid \text{there is no } (t', p', o') \in X \text{ such that } (t', p', o') \prec (t, p, o)\}$$

We define the set of all derived statements for a resource s as

$$D_s = \mathrm{filter}(\{(\perp, p, o) \mid \langle s, p, o \rangle\} \cup \{(t, p, o) \in A \mid s : t\}).$$

The set of all derived statements is

$$D = \{(s, p, o) \in \mathcal{R}^3 \mid (t, p, o) \in D_s\}.$$

Finally, we can define a relation

$$\langle\langle s, p, o \rangle\rangle \iff \exists p' \in \mathcal{R}.p' \leq_R p \wedge (s, p', o) \in D$$

that contains all derived statements and their superstatements (i.e. a statement whose predicate is a superrelation of the predicate of the original statement and whose subjects and objects are same).

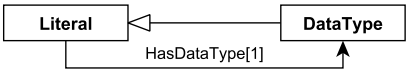
We have also new versions of relations and functions:

$$b^{\mathrm{Rel}} = \{(a, c) \mid \langle\langle a, b, c\rangle\rangle\}$$

and partial unary function

$$b^{\mathrm{Fun}}(a) = c, \text{ if there exists a unique } c \text{ such that } \langle\langle a, b, c\rangle\rangle.$$

7. Literals



7.1. **Informative definition.** We left it open in section 3 how the data types of the values are encoded. We define now that derived **HasDataType** statements are used to attach the data type to the literal.

7.2. **Formal definition.** We define partial functions Value: $\mathcal{R} \to \mathcal{V}$ and DataType: $\mathcal{R} \to \mathcal{T}$ that return the associated value and data type of the literal:

$$\begin{aligned}\mathrm{DataType}(r) =& \mathrm{Val}_{\texttt{DataType}}(\mathbf{HasDataType}^{\mathrm{Fun}}(r))\\ \mathrm{Value}(r) =& \mathrm{Val}_{\mathrm{DataType}(r)}(r).\end{aligned}$$

8. Validity

The semantic graph is valid if none of its integrity constrains are violated. There are three possible strategies for maintaining an integrity constraint:

(1) Allow only such primitive operations on semantic graph that don't break the constraint.
(2) Validate the graph when the write transaction is committed. Rollback if the graph is invalid.
(3) Allow invalid graphs but report the problems as issues to the user.

In the last two strategies, the implementation has to cope with invalid graphs: in (2) only during write transactions and in (3) also during read transactions.

In this specification, we do not define when and how the integrity checks are done or how they are reported. The following is the list of integrity constraints related to the mechanisms presented in the previous sections. It doesn't contain domain and range restrictions or cardinality restrictions of the relations. In subsequent sections, we add more constraints.

8.1. **Predicates of statements have to be relations.** The most straightforward way to formalize this constraint is

$$\langle\langle s, p, o\rangle\rangle \to p : \mathbf{Relation}$$

However the constraint is easier to implement with the combination of the following constraints:

$$\langle s, p, o\rangle \to (p = \mathbf{IsWeaklyRelatedTo} \vee \exists p'.\langle p, \mathbf{SubrelationOf}, p'\rangle)$$
$$(t, p, o) \in A \to p : \mathbf{Relation}$$

The second constraint will be checked as a range restriction of the relation **HasPredicate** and so only the first constraint requires special validation logic.

8.2. **Two resources may not have the same URI.** Again, the direct formalization is:

$$\mathrm{URI}(a) = \mathrm{URI}(b) \to a = b$$

but it is easier to check for all resources p that all c such that $\langle p, \mathbf{ConsistsOf}, c\rangle$ have different names. We should require this condition even if p doesn't have a URI.

8.3. **A resource has a value if and only if it is a literal.** This condition is written as

$$V(r) \text{ is defined} \leftrightarrow r : \mathbf{Literal}$$

8.4. **The value has to match the data type definition.** This condition can be written as

$$V(r) \text{ is defined} \to \mathrm{Value}(r) \text{ is defined}$$

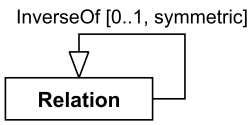
8.5. **Inheritance hierarchy does not have cycles and all types inherit Entity.** The first constraint says that the directed graph $\mathbf{Inherits}^{\mathrm{PRel}}$ is acyclic. Assuming it holds, the second condition can be written as

$$r : \mathbf{Type} \to \exists r'.\langle r, \mathbf{Inherits}, r'\rangle \vee r = \mathbf{Entity}.$$

Note that if r inherits something that is not a type, the graph is invalid because of the range restriction of **Inherits**.

8.6. **SubrelationOf hierarchy does not have cycles and all relations are subrelations of IsWeaklyRelatedTo.** This is completely analogous to the previous constraint.

9. Inverses



9.1. **Informative definition.** A relation may have an *inverse relation*. If it has, for all its statements there have to be also an *inverse statement* whose predicate is the inverse relation and whose subject and object have been swapped. Inverse relations are useful in queries where one tries to find all subjects with known predicate and object. A relation is called *symmetric* if it is its own inverse.

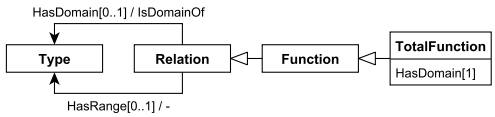
The subrelation hierarchy of a relation and its inverse have to match: If a relation is a subrelation of another relation with an inverse, it must have an inverse that is a subrelation of the inverse of the superrelation. Also domain and range restrictions that will be defined in the next section have to match.

9.2. **Formal definition.** Inverse relation mechanism is defined with certain constraints. Assuming that $\langle p, \mathbf{InverseOf}, p' \rangle$, we have:

$$\begin{aligned} \langle s, p, o \rangle &\rightarrow \langle o, p', s \rangle \\ \langle p, \mathbf{SuperrelationOf}, q \rangle &\rightarrow \exists q'.(\langle q, \mathbf{InverseOf}, q' \rangle \wedge \langle p', \mathbf{SuperrelationOf}, q' \rangle) \\ \langle p, \mathbf{HasDomain}, t \rangle &\rightarrow \langle p', \mathbf{HasRange}, t \rangle \\ \langle p, \mathbf{HasRange}, t \rangle &\rightarrow \langle p', \mathbf{HasDomain}, t \rangle \end{aligned}$$

We do not require that derived statements have inverses.

10. Relation constraints

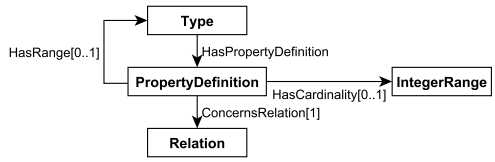


10.1. **Informative definition.** Relation can be defined to connect instances of certain types: the *domain* and *range* of the relation. Also its cardinality can be restricted: a **Function** may have only one derived statement per one subject and a **TotalFunction** must have exactly one such statement.

10.2. **Formal definition.** The constraints for the concepts are defined formally as

$$\begin{aligned} \langle\langle p, \mathbf{HasDomain}, t \rangle\rangle \wedge \langle\langle s, p, o \rangle\rangle &\rightarrow s : t \\ \langle\langle p, \mathbf{HasRange}, t \rangle\rangle \wedge \langle\langle s, p, o \rangle\rangle &\rightarrow o : t \\ p : \mathbf{Function} \wedge \langle\langle s, p, o \rangle\rangle \wedge \langle\langle s, p, o' \rangle\rangle &\rightarrow o = o' \\ p : \mathbf{TotalFunction} \wedge \langle\langle p, \mathbf{HasDomain}, t \rangle\rangle \wedge r : t &\rightarrow \exists o. \langle\langle s, p, o \rangle\rangle \end{aligned}$$

11. Type constraints



11.1. **Informative definition.** Types may contain additional integrity constraints that refine the constraints specified in relations.

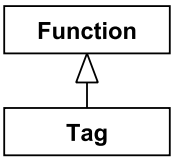
A **PropertyDefinition** constraints the use of one relation pointed by **ConcernsRelation** with the type. Restriction **HasCardinality** restricts the number of (derived) statements having the relation as predicate. Restriction **HasRange** restricts the type of the objects of such statements.

11.2. **Formal definition.** Assume t is a type, r is its instance ($r : t$), d is a property definition such that $\langle\langle t, \mathbf{HasPropertyDefinition}, d\rangle\rangle$ and p is relation such that $\langle\langle d, \mathbf{ConcernsRelation}, p\rangle\rangle$. The constraints for the concepts are defined formally as

$$\langle\langle d, \mathbf{HasRange}, t'\rangle\rangle \wedge \langle\langle r, p, o\rangle\rangle \rightarrow o : t'$$
$$\langle\langle d, \mathbf{HasCardinality}, c\rangle\rangle \rightarrow |\{o \mid \langle\langle r, p, o\rangle\rangle\}| \in \text{range}(c),$$

where range(c) is the set of integers in the **IntegerRange** c.

12. Tags



Tags are a way to encode unary relations to semantic graphs. If relation is a tag, it can be used only in statements whose subject and object are the same resource.

Layer0 ontology defines the following tags:

12.1. **Abstract.** A type or relation can be tagged **Abstract**. An abstract type may not be directly instantiated. An abstract relation may not be used directly in statements. Formally:

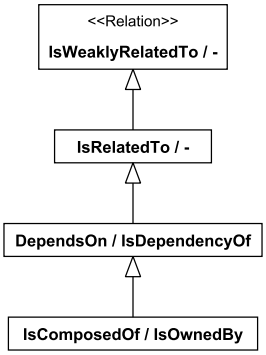
$$\neg(\langle\langle t, \mathbf{Abstract}, t\rangle\rangle \wedge \exists r.\langle r, \mathbf{InstanceOf}, t\rangle)$$
$$\neg(\langle\langle p, \mathbf{Abstract}, p\rangle\rangle \wedge \exists s, o.\langle s, p, o\rangle)$$

12.2. **Final.** A type or relation is tagged **Final** when it must not be inherited.

12.3. **Enumeration.** A type can be tagged **Enumeration** to indicate that it has only a fixed number of instanes all defined in the same ontology. All instances of the enumeration must be **PartOf** the type.

12.4. **Deprecated.** Any concept can be tagged **Deprecated** if it supports some deprecated mechanism or is replaced by other concepts. Deprecated concepts should not be used anymore and they will be removed in the future major revisions of the ontology.

13. Relation hierarchy



13.1. **IsWeaklyRelatedTo.** The base relation for all other relations. It has no other semantics.

13.2. **IsRelatedTo.** Used to specify which parts of the graph are reachable. Resources that are not reachable can be garbage collected. More formally, a resource r is reachable, if there exists a sequence of resources: $r_0, \ldots, r_n$ such that $r_0 =$ **RootLibrary** and $r_n = r$ and for all $i < n$, $\langle\langle r_i, \textbf{IsRelatedTo}, r_{i+1}\rangle\rangle$. An unreachable can be removed (i.e all its statements are removed) any time between transactions.

13.3. **DependsOn.** If a statement of the resource is added or removed or the value of the resource is changed, the transaction that modified the resource generates a change event on the resource. Relation **IsDependencyOf** tells how the change event is propagated. If change event happends on r and $\langle\langle r, \textbf{IsDependencyOf}, r'\rangle\rangle$, change event is also raised on r'.

13.4. **IsComposedOf.** If a resource is owned by another resource, its existence is tied to that resource and it should be removed when its owner is removed. This corresponds to composition in UML.

14. Predefined literal types

Layer0 defines the following types inheriting **Literal**. In the second column of the table is the data type that the type asserts.

Resource	Data type
Boolean	`Boolean`
Byte	`Byte`
Integer	`Integer`
Long	`Long`
Float	`Float`
Double	`Double`
String	`String`
BooleanArray	`Boolean[]`
ByteArray	`Byte[]`
IntegerArray	`Integer[]`
LongArray	`Long[]`
FloatArray	`Float[]`
DoubleArray	`Double[]`
StringArray	`String[]`
Variant	`Variant`
IntegerRange	`{ min : Optional(Integer), max : Optional(Integer) }`

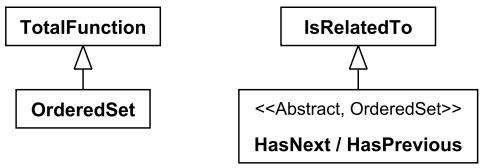
15. Organizing data

- Library
- Ontology

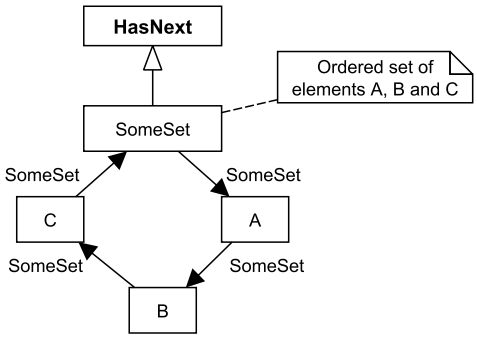
16. Annotating data

- HasLabel
- HasDescription
- HasComment

17. Ordered sets

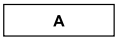


Ordered set is a way to write ordered data to the semantic graph. Ordered set itself is a relation inheriting **HasNext**. The ordered set is encoded as a circular list containing the set resource and the elements of the set. The subsequent elements of the list are connected with the ordered set relation. Here is an example:

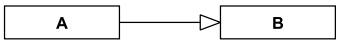


18. Graphical ontology definition language

The basic building blocks of the graphical ontology definition language are nodes and arcs between them. A node represents always a resource. If no type, supertype or superrelation is given for the resource, it inherits **Entity** by default. In this way nodes are by default types.

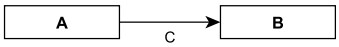


An edge with open arrowhead tells that a type inherits another type.



In this example, A inherits B.

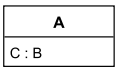
An edge with closed arrowhead defines a relation with certain domain and range. If domain or range is **Entity** or corresponds to the domain or range of the superrelation, it is not explicitly written to the graph.



In this example C is a relation with domain A and range B.

The label of the edge contains the name of the relation. If the relation is **Function**, it is indicated with `[0..1]` after the name of the relation. If the relation is **TotalFunction**, it is indicated with `[1]`. If the relation has specially named inverse, it is written to the label after /-character. Also the inverse may have its own cardinality restrictions, for example `ConsistsOf / PartOf[0..1]`. By notation ⟨relation name⟩`/-`, it is explicitly indicated that the relation does not have an inverse.

Another way to specify relations is to give them in the attribute section of the type:



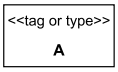
This indicates that C is a relation with domain A and range B.

By default, the relation in the attribute section is **TotalFunction**. It can be specified as normal relation by writing `C[*] : B` and as **Function** by `C[0..1] :`

`B`. Also the notation `C[1] : B` can be used to emphasize that the relation is a total function.

If the relation is already defined either by an edge or in the attribute section of a supertype of the type, the attribute section may contain further type restrictions. They are written in the same way as above.

Types and tags of the resources are indicated in the same way as stereotypes in UML:



Depending on whether the resource in guillemets is type or tag, the resource is instantiated or tagged. Multiple types and tags can be specified separating them with comma. If a type is given for the resource and it is not a type, the resource is not inherited from **Entity** by default.

[Should we use notation `A : B` in the node labels for typing?]

19. Conventions used in Simantics ontologies

19.1. **Names.** The names of the concepts begin with an uppercase letter and contain only letters and numbers. The names with multiple words are written in *CamelCase* convention.

Names of relations are usually verbs.

19.2. **Inverses.** If a relation has a specially named inverse, it has the same parent as the relation itself. By default, the inverse relation is named as ⟨URI of the relation⟩`/Inverse`.

References

[1] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), January 2005.

[2] Toni Kalajainen and Hannu Niemistö. Databoard specification, 2010.

E-mail address: `hannu.niemisto@vtt.fi`

E-mail address: `antti.villberg@semantum.fi`