Part 1: An Exploratory Introduction to Semantic Modeling

A. W. Crapo (mailto:crapo@ge.com), last revised 20 May, 2020

This seminar is broken down into two parts. Part 1 covers foundational concepts. While it is largely independent of any specific semantic modeling language, **it uses SADL and OWL to illustrate the concepts and to enable participants to do exercises** and think about the concepts in a hands-on fashion. Part2 covers additional information which is specific to OWL, SADL, and GE Research knowledge delivery mechanisms. To download and install SADL, follow the instructions at https://github.com/crapo/sadlos2/wiki/Installation-Instructions-for-SADL-IDE,-Version-3.3. Also download and install GraphViz per link at end of those instructions.

Introduction: For a Domain Familiar to You…

1) Identify the names of some common concepts in the domain
2) Categorize these names:
   a) Are they specific things, distinguishable from all other things?
   b) Are they types of things?
   c) Are they attributes of things?
   d) Are they relationships between things?
   e) Are there other possibilities?

Discussion: Semantics Is about Meaning—How can we make a model "mean something" to a computer?

Set Theory

1) A **set** is a collection of things which are similar in some important way
   a) can be defined "extensionally"--by naming all the members of the set, a trilogy of books, e.g., "The Fellowship of the Ring", Presidents of the United States, seasons of the year
   b) can be defined "intensionally"--by the characteristics of the members of the set, e.g., viruses
      i) intrinsic characteristics (Pierce's Firstness)
      ii) relationships to other things (Pierce's Secondness)
      iii) mediating concepts (Pierce's Thirdness)
2) Sets can be partially ordered into hierarchies: dog is a subset of mammal; mammal is a subset of animal. What does it mean to say that A is a subset of B? (Every instance of A is also an instance of B.) Note that every set is a subset of itself. However, when **A** and **B** are not the same set and **A** is a subset of **B**, then **A** is "proper" subset of **B**.
3) Sets can be created from other sets
   a) Union (Food is Edible or Potable)
   b) Intersection (Mother is Woman and Parent)

Propositional Logic

1) A proposition is something that is expected to be true of false, usually represented by a letter such as '*p*' or '*q*', e.g., *p* is the proposition *"it is raining (here, now)"*
2) Propositions can be combined using 5 logical operators
   a) Conjunction: '^', '&' (and)
      i) given *p* is true and *q* is false, what is "*p* ^ *q*"? (false)
      ii) given *p* is true and *q* is true, what is "*p* ^ *q*"? (true)
   b) Disjunction: 'v', '|' (or)
      i) given *p* is true and *q* is false, what is "*p v q*"? (true)
      ii) given *p* is true and *q* is true, what is "*p v q*"? (true)
   c) Implication: → (implies)
3) given *p* is true and *p* → *q*, what can you say about *q*? (*q* is true)
4) given *q* is true and *p* → *q*, what can you say about *p*? (nothing)
   a) Necessary and sufficient: ↔ (if and only if)
5) given *p* is true and *p* ↔ *q*, what can you say about *q*? (*q* is true)
6) given *q* is true and *p* ↔ *q*, what can you say about *p*? (*p* is true)
   a) Negation: ¬ (not)
      i) given *p* is true, what is ¬*p*? (false)
      ii) given *p* is true, what is ¬*(¬p)*? true

Verify that all participants have SADL installed properly and can run it. (This should be accomplished before the session begins. This is just to verify.)

1) Set preferences for our project:
   a) Window -> Preferences -> SADL
   b) Under "Translation Settings", check "Use indefinite and definite articles…"
   c) Under "Type Checking Settings"
      i) Uncheck "Property range specification required"
      ii) Check "Type checking issues as warnings only.
2) Create a new project for "SemanticsSeminar" (or whatever name one chooses)
   a) File -> New -> Project… -> General -> Project.
3) Create a SADL file, e.g., PredicateLogic.sadl and verify that the SADL editor works
   a) File -> New -> File.
4) Add the following to the model:
   a) Create a set (class) named *Person*
   b) Create two subsets named *Man* and *Woman*
   c) Create an instance of a *Man* named *Adam*
   d) Create an instance of a *Woman* named *Eve.* **(PL1)**

Predicate Logic (aka First Order Logic)

1) A **predicate** is a symbol which takes argument(s) and returns true or false
   a) *man(Adam)* is true (*Adam ϵ man*)
   b) *man(Eve)* is false
   c) *child(Adam, Cain*) is true
   d) *child(Adam, Eve)* is false
   e) *brother(Cain, Abel)* is true

2) The number of arguments is called the "arity", e.g., "arity of *child* above is 2"
3) a predicate with arity 1 can be used for set (class) membership
   a) *man(Adam)* true means *Adam* belongs to the set *man*
4) A predicate with arity 2 can be used to define relationships
   a) *brother(Cain, Abel)* true means *Abel* is *Cain's* brother
   b) Note: if a language has only predicates with arity <= 2, then a model in that language is a mathematical directed graph
      i) A directed graph is a set of binary relationships: node → edge → node
         (1) Cain → brother→ Abel
         (2) A statement in a graph-based language is sometimes called a **triple**.
         (3) The *predicate* of the triple, the "edge", is called *property*
         (4) Class membership formulated as a triple: *Cain rdf:type Man*
      ii) OWL is a graph language => SADL is a graph language (plus other things)
      iii) Graphs have certain properties that can be useful
         (1) Imagine a graph query language. What would a graph pattern look like?
            (a) <node> <property> <node> (would return true or false)
            (b) <node> < property > ? (could return an instance of a class, e.g., *Philp friend ?* might return *Albert*)
            (c) ? < property > <node>
            (d) ? < property > ?
            (e) <node> ? ?
            (f) ? ? <node>
            (g) ? ? ?
            (h) <node> < property > ?x, ?x < property > ?y (who are the friends of Joe's siblings?)
            (i) etc.
         (2) In SADL, a query is preceded by "Ask:" and variables are not preceded by "?". Write a query asking for all instances of Person. (PL2)
   c) It can be useful to have predicates with arity > 2
      i) How might you represent this:
         (1) "Dan and Eileen were married in Palmyra, NY on July 1, 2000"?
         (2) In an n-ary language, one could just say
         (3) marriage(Dan, Eileen, PalymraNY, July1-2000)
         (4) what is the arity of *marriage* in this language? (4)
5) A **function** is a symbol which takes argument(s) and returns another non-logical symbol (not true, not false, not one of the 5 operators)
   a) *friend(Philip)* might return *Albert*
6) **Quantification**: binds variables to non-variable symbols—it is how you make statements about multiple things declaratively
   a) Existential
      i) ∃x: Person(x) ^ worksFor(x, GE)
         (1) Read "there exists x such that x is a Person and x works for GE"
   b) Universal
      i) ∀x: Rich(x) → loves(x, Money)
         (1) Read "for all x, if x is rich then x loves money"
      ii) How would one say "Everyone loves money?"

Domain and Range of a Binary Predicate (aka Binary Property)

1) **Domain**: if *pred* is a binary predicate (*pred(x,y)*), then the set of all possible values of *x* is the *domain of pred*.
   a) In algebra, y = f(x) = x^2, 4 <= x <= 10, implies the domain of f(x) is [4-10] (4 to 10 inclusive) (4 to 10 exclusive is sometimes written as (4-10).
   b) In OWL-style semantics, a statement (or triple) is a binary predicate whose 1st argument is called the *subject* and whose 2nd argument is called the *object*, e.g., *child(Eileen, Philip*) has *Eileen* as subject, *child* as predicate, *Philip* as object.
   c) What might be the domain of *child*?
   d) Depending on the "domain of discourse", it might be the class *parent*. You might also think of it as *Person*.
2) **Range**: if *pred* is a binary predicate (*pred(x,y*)), then the set of all possible values of *y* is the *range of pred*.
   a) In algebra, y = f(x) = x^2, 4 <= x <= 10, implies the range of f(x) is 16 <= y <= 100
   b) In OWL-style semantics, predicates are divided into two kinds--those that have instances [of classes] as object values (aka relationships), and those that have numbers, strings, dates, etc., as object values (aka attributes)
      i) The first is *owl:ObjectProperty*, e.g., *brother(Cain, Abel)*
      ii) The second is *owl:DatatypeProperty*, e.g., *age(Philip, 13)*
         (1) What is the domain and range of *age*?
            (a) domain is probably *Person* (or *PhysicalObject*?)
            (b) range is probably non-negative integer [0,)
            (c) Remember that in OWL, unlike object-oriented languages, a property can be defined without any domain or range (but to be an OWL property you do have to say whether it is a owl:DatatypeProperty or owl:ObjectProperty)
            (d) However, rdf:Property is the superclass of owlDatatypeProperty and owl:ObjectProperty and so isn't so constrained.
3) The domain of any property and the range of an *owl:ObjectProperty* are usually expressed as classes (*owl:Class*).
   a) There are two kinds of classes (see above, Set Theory 1b). For intensionally defined classes:
      i) Class membership can be determined by examining the thing itself
         (1) Philip is Boy.
         (2) Eileen is a Woman.
         (3) MammyGarrettBoulder is a Rock.
      ii) Class membership is determined by the relationships of the thing to other things, aka a "role" class
         (1) Child is a role class--it is the class of everyone who is the object of a child relationship
         (2) Philip is a Son.
         (3) Eileen is a Mother.
         (4) Eileen is a Wife.
4) **Add the following to the model created previously:**
   a) **Create a property named *child* with domain and range *Person***
   b) **Create an instance of *Person* named *Cain*. State that *Adam* has a *child Cain*. State that *Eve* has *child Cain***
   c) **Create a property named *brother* with domain *Person* and range *Man***
   d) **State that *Cain* has a *brother* who is the *Man Abel***
   e) **Generate a graph of your model:**

i) Select the SADL file in the Package Explorer to give it the focus
ii) Click on the SADL toolbar icon (gears) and select "Graph Ontology" from the dropdown menu. (PL3)

5) How can n-ary relationships be represented with a binary language ("Dan and Eileen were married in Palmyra, NY on July 1, 2000")?
   a) (a *Marriage* with *husband Dan*,
      i) with *wife Eileen*,
      ii) with *location* "Palmyra, NY"
      iii) with *date* "July, 2000"
   b) Note: *Marriage* is the "mediating class" in Set Theory 1.b.iii.
   c) Create the equivalent of this arity-4 example in SADL
   d) Graph the model (PL4)
   e) Note that in OWL models it is not necessary to give everything a unique identifier. In this way it is like the real world where most things are identified by association, e.g., my car.

6) Create a property "favoriteThing" with domain Person. Do not restrict the range.
   a) Create yourself as an instance of Person and identify several of your favorite things.
   b) Draw a graph with you as the anchor node. Redraw the graph with radius 1. (PL5)

Types of Properties

1. Transitive Property
   a. *locatedIn* is transitive.
      1. *Atlanta locatedIn Georgia* and *Georgia locatedIn USA* => *Atlanta locatedIn USA*
2. Symmetric Property
   a. *spouse* is symmetrical.
      1. *Tarzan* has *spouse Jane* => *Jane* has *spouse Tarzan*.
3. Functional Property: a given subject can only have one triple with this property
   a. father describes Person with values of type Man.
   father has a single value.
      1. *father* can have a single object for a given subject: *Abel father Adam.*
4. Inverse Property
   a. *ownedBy* is the inverse of *owns.*
      1. *GeorgeWashington owns MountVernon* => *MountVernon ownedBy GeorgeWashington*
      2. Can an owl:DatatypeProperty have an inverse property?
5. Inverse Functional Property: only one subject can have a given object value for the property
   a. isBirthMotherOf describes Woman with values of type Person.
   isBirthMotherOf has a single subject.
      1. *isBirthMotherOf* can have a single subject for a given object: *Eve isBirthMotherOf Abel.*

Partial Ordering of Properties

1) Like classes, properties can be partially ordered with the rdfs:subPropertyOf relationship.
2) If p2 is a sub-property of p1, then (A p2 B) => (A p1 B)

   a) "son" rdfs:subPropertyOf "child"
   b) GeorgeHW son GeorgeW => GeorgeHW child GeorgeW

c) Create a property *son*, sub-property of *child*, in your SADL model.
d) Create two new instances of *Person* with one the *son* of the other.
e) Write a query that finds all parents and children. Are your two new instances in the results? (PL6)

Property Restrictions on Classes: Refining Class Definitions

1) Cardinality restrictions: How would we say
   a) "a hand has 5 fingers"?
      i) *digit* is an owl:ObjectProperty with domain *Hand*, range *Finger*
      ii) *digit* of *Hand* has exactly 5 values. (Note: this is cardinality.)
   b) "a hand has 4 fingers and a thumb"?
      i) *digit* is an owl:ObjectProperty with domain Hand, range Digit.
      ii) *digit* of *Hand* has 4 values of type *Finger*.
      iii) *digit* of *Hand* has 1 value of type *Thumb*.
         (1) Note: this is qualified cardinality.
         (2) Note: *digit describes Hand with a single value of type Thumb.* Also specifies a qualified cardinality on *digit* of *Hand*.
   c) *component* of *Bicycle* has exactly 2 values of type *Wheel*
   d) Choose one of these examples to implement in SADL <u>as a new model</u> (R1)
2) Minimum cardinality restrictions
   a) *child* of Parent has at least 1 value.
3) Maximum cardinality restrictions
   a) *spouse* of *Person* has at most 1 value.
4) Has value restrictions
   a) *gender* of *Man* always has value *Male*.
5) Some values from restrictions (existential quantification)
   a) *teaches* of *Professor* has at least one value of type *CollegeStudent*.
   b) $\forall x \exists y$: Professor(x) ^ teaches(x,y) → CollegeStudent(y)
6) All values from restrictions (universal quantification)
   a) *loves* of *RichMan* only has values of type Money.
   b) $\forall x \forall y$: Rich(x) ^ loves(x,y) → Money(y)

7) Implement a restriction that Hand must have some values of type Digit.
8) Implement a restriction that Hand can only have values of type Digit. (R2)

Necessary and Sufficient Conditions

1) Equivalent classes (↔ from propositional logic)
   a) A Person is a Parent only if child has at least 1 value.
      i) George is a Person with child James. → George is a Parent.
      ii) George is a Parent. → George is a Person with child ?x.
   b) Implement a necessary and sufficient condition that something is a Hand only if it has at least one digit. (R3)

Inference: What Statements are Implied by What is Known?

1) Examples
   a) "age" has domain Person and John has age 23 → John is a Person
   b) George is a Person with child James → George is a Parent.
2) OWL has different flavors with different entailments (inferences)
3) The default SADL reasoner, which is Jena-based, isn't really an OWL reasoner. It uses rules (to be covered below)

4) A Prolog reasoner is also available
5) OWL reasoning can have scaling issues—do only as much reasoning as you need

Rules

1) Some things are logical but complex to say and understand in axioms
   a) `Rule Uncle: given x is a Man and y is a Person and z is a Person if x is sibling of y and z is child of y then x is uncle of z.`
   b) Returning to your first model, define a symmetrical property *sibling* and write a rule to infer the *sibling* relationship.
   c) Add a query to find all siblings and run it using the SADL menu command "Test Model".
   d) Did you get the expected result? If not, how can you investigate?
      i) Demonstrate "Explain: Rule siblingRule.
      ii) What can you add to your model to make it do what you expect?
   e) How can you modify your query so that you see both siblings in pairs? (PL7)
2) Some implications are not logical
   a) `Rule AreaOfCircle: if x is a Circle then area of x = radius of x^2 * PI.`
   b) Create a new model and implement this rule. Test the model to make sure the rule works as you expect. (S1)
3) Many rule languages include snippets of procedural code, in the form of special functions called **built-ins**, that perform computations.
   a) pow(x, 2, y)
   b) product(y, 3.14159, z)
   c) These built-ins should not have side-effects in the knowledge base; they should simply return a value
   d) What happens if these built-ins are "out of sight" of the reasoner/rule engine?
      i) If an argument to a function is computed by another rule, how will the rule engine know to process the other rule first so the value will be known?
   e) Refresh the project (select the project name in the Project Explorer, then press F5). Open the .rules file in the OwlModels folder and look at your rule in Jena Rule syntax.
4) Rule processing must be integrated with logical inference as each will have an effect on the other.
   a) The rule language must have clear semantics that are compatible with the ontology language.

Miscellaneous

Identity

1. In OWL, identity is established by URI—if two things have the same URI they are the same thing
   a. An XML namespace is identified by a URI.
   b. Optionally, a URI can end with a "fragment identifier", e.g., http://sadl.org/concepts#Test ends in fragment "Test".
   c. Names of instances, classes, and properties are fragments in some namespace.

d. A name need only be unique within the namespace.
    i. http://sadl.org/concepts#Test is not the same as http://http://opencontent.org/evaluation#Test

Declarative versus Procedural

1) Languages such as C, C++, Python, and Java are procedural—they allow you to write a set of instructions which the computer will follow sequentially and precisely.
    a) Could you write a program in a procedural language that could drive a car?
2) Languages such as OWL, Common Logic (CL), and Prolog (mostly) are declarative—they declare what is and what is related but there is no specified sequence.
    a) *Person* is a class. *Man* is a type of *Person*. *Adam* is a *Man*.

Open World Assumption (versus Closed World)

1) Many IT systems, e.g., relational databases, assume that everything is known. Therefore, if something is not known to be true it can be assumed to be false. This is known as the "closed world assumption".
2) In the Web, such an assumption cannot be made. If we don't know something all we can conclude is that we don't know it. This is known as the "Open World Assumption".
3) Inference behaves quite differently under open versus closed world assumptions.

Negation as Failure

1) It is sometimes possible to infer that something is false because it is not known to be true (closely related to the closed world assumption). See Negation in http://sadl.sourceforge.net/tutorial/reasoningrulesqueries.html.