Part 2: OWL, SADL, and Knowledge Delivery in the Real World: Beyond the Foundational Concepts

A. W. Crapo, last revised 20 May, 2020

This seminar is broken down into two parts. Part 1 covers foundational concepts and is largely independent of any particular semantic modeling language, although it uses **SADL for exercises** to help participants better understand and remember the concepts. Part2 covers additional information which is specific to OWL, SADL, and GE Research knowledge delivery mechanisms.

The following are topics to be covered, but the exact order may depend upon how the exploration proceeds for the given participants and what they want to cover.

What is OWL?

1) OWL, the Web Ontology Language, is a **graph-based**, **Description Logics language** developed for the Semantic Web.
2) OWL supports modularity: "Information in OWL is gathered into ontologies, which can then be stored as documents in the World Wide Web. One aspect of OWL, the importing of ontologies, depends on this ability to store OWL ontologies in the Web." (http://www.w3.org/TR/owl-ref/#OWLDocument)
3) An OWL ontology can import another ontology, thereby making the concepts defined in the imported ontology available for use. For example:

```
uri "http://sadl.imp/shapes_top" alias top.
Shape is a class, described by area with values of type float.



uri "http://sadl.imp/shapes_specific" alias shapes-specific.
import "http://sadl.imp/shapes_top".
Circle is a type of Shape, described by radius with values of type float.
```

   If the second ontology did not import the first, "Shape" would be undefined in the second.

4) owl:imports is a transitive property; if A imports B and B imports C then A is "aware" of all of the concepts in both B and C.
5) Since the URI of an Ontology is not necessarily the actual location (URL) from which it may be retrieved, a mapping of URIs to URLs may be necessary. In SADL (using Apache Jena) this is provided by the "ont-policy.rdf" file in the project's OwlModels folder. **Note: this file should not be edited directly. It is maintained by the system.**
6) Not all OWL models are stored as documents in the Web. It is also possible to store one or more models in a repository called a "triple store". In the triple store, a "document" is called a "named graph".

7) **Refactor your "SemanticsSeminar" ontology so that all of the class and property declarations, along with rules are in a "meta-model" file and all of the instances are in a "instance data model" file. (You can create new .sadl files and leave the old models intact for reference if you wish.)** **(PMD1)**

8) SADL does not support circular imports, e.g., Model1 imports Model2 and Model2 imports Model1. (OWL does support circular imports.)

9) A SADL model can import any OWL model (not already in the project) using the SADL Model URL List file Wizard

    a) New -> Other -> SADL Model URL List Wizards -> SADL Model URL List file

    b) **Create a .url file using this Wizard**

    c) **Add this URL: http://sadl.sourceforge.net/owl/time-entry.owl , save the file, switch to the Download tab, and download the external model.**

        i) **(If the firewall prevents you from downloading, download in a browser, save, then use a "file:/…" URL (path to where you saved the file) as entry in the .url file.)**

    d) **Create a new SADL model with a name like "Temporal.sadl".**

        i) **Use content assist (cntrl-space) to fill in the uri statement.**

        ii) **Use content assist on a new line (new statement) and pick "import".**

        iii) **Use content assist again and scroll down to the URI ending with "time-entry.owl". (You might want to give this a local alias by adding " as teo" before the EOS period.)**

    e) **Create an instance of "IntervalThing" with a name like "MyLife"**

    f) **Create an instance of "InstantThing" with a name like "MyBirth".**

    g) **Create an "inside" relationship between "MyLife" and MyBirth".** **(T)**

10) **Eclipse keeps its own version of resources on disk, which can get out of synch with the file system. This is remedied by Refresh (right-click on the project in Package Explorer and select Refresh, or just press F5)**

11) **Sometimes the incremental building of ontologies needs to be done over. This is accomplished by**

    a) **Make sure that "Build Automatically" on the Project menu is checked (it should always be checked)**

    b) **Select "Clean…" on the Project menu**

Identity Revisited

1) URIs provide identity in the Web and in our semantic models

    a) All of the simply named concepts in a SADL model have a URI which is:

        i) <uriOfModel>#<simpleName>

2) Most things in the real world do not have unique identifiers

    a) My pen

    b) Your dog

3) Even when they do, we don't always use the unique identifier

    a) What's your car's unique identifier? Can you tell me your VIN #? We say, "Let's take your car."

    b) We say, "Meet me at my house."

4) Nodes in OWL do not have to have a URI—they can be a **blank node** (bnode)
   a) George has pet (a Dog with color Blonde, with weight 25).
   b) Inference may create bnodes
   c) **Add the property "sister" to your "people model". Add that Cain has an unnamed sister to your "people data". Look at the OWL model for "people data".   (PMD2)**
   d) **Add "age" as a property with domain Person and range int to your "people model". Add a value for "age" to Cain's sister. Look at the OWL model for your "people data".   (PMD3)**

What is SPARQL?

SPARQL (Recusive acronym: SPARQL Protocol and RDF Query Language)

1) Remember graph patterns from Part 1?
   a) E.g., <node> <predicate> ?x, ?x <predicate2> ?y
2) We just need to add some additional features…
   a) Select which variable binding to include in the results
   b) Allow matching of some graph patterns to be optional
   c) Allow joining of result sets (union)
   d) Provide filters to reduce result sets in ways not easily expressed in graph patterns, e.g. ?y < 23
   e) Use "." to separate triple patterns (meaning "and")
   f) Result: select ?x ?y ?z where {{?x <component> ?y . ?y <partNumber> ?z} union {?x <part> ?y . optional{?y <partNo> ?z}} . filter(isURI(?x)}
3) Note: a SPARQL *select* query outputs tabular data—***the semantics are lost!***

What is SADL?

SADL Is a Language: A Controlled-English Grammar Implemented with Xtext

1) References
   a) http://sadl.sourceforge.net
   b) https://github.com/crapo/sadlos2/wiki
   c) http://sadl.home.openge.ge.com
2) Expressivity: OWL 1 + qualified cardinality
   a) Classes definitions, including the set operations union and intersection
   b) Properties including domains, ranges
      i) rdf:Property
      ii) owl:ObjectProperty
      iii) owl:DatatypeProperty
      iv) owl:AnnotationProperty
         (1) *note*, *alias*, user-defined
   c) Instances

d) Property restrictions on classes: cardinality, qualified cardinality, has value, some values, all values

e) Equivalent classes (necessary and sufficient conditions)

f) User-defined data types

    i) Some clothing can be described by integer sizes or by character descriptions such as "S", "M", "L", etc.

        (1) One could specify a user-defined data type to allow either.

        (2) `clothingSize is a type of {decimal or string}`.

g) It supports modularity through owl:imports

3) In addition to OWL, SADL supports **Queries**

    a) Can be in SADL Query Language

        i) Controlled English subset of SPARQL

    b) Can be named, reusable

    c) Can be an opaque SPARQL query (in quotes)

        i) Graph patterns and filters, e.g., "?p <age> ?age . filter (?age > 18)"

        ii) To be efficient eliminate as much of search space as possible as fast as possible

        iii) Advanced property paths

            (1) One kind of property path is a **property chain**, the nesting together of triple patterns

            (2) `age of the wife of Adam` (Adam wife ?x, ?x age ?y)

            (3) See https://jena.apache.org/documentation/query/property_paths.html for many more useful SPARQL property path constructs

        iv) VALUES keyword in SPARQL

        v) OPTIONAL keyword in SPARQL

    d) Can be parameterized: use just a "?" for each parameter in definition

        i) Opaque SPARQL query string only (no SADL query syntax)

        ii) Values supplied after query name in a SADL list

        iii) `Ask: FindPeopleByAgeRange:[30,35]`.

        iv) Requires a space before and after each "?" in definition

    e) SADL keywords

        i) Ask

        ii) Graph (the query must return a table of data with three columns)

    f) SPARQL query types

        i) Ask

        ii) Select

        iii) Construct

        iv) Update

        v) Insert

        vi) Delete

    g) Add ages to other instances of Person in your "people data". Create a query to find all Persons over a specified age. Between two specified ages.   (PMD3b)

h) Create a Named Query in your "people model" to find all Persons at or over the age of 18. Reference and run the query in your "people data". Modify the query to order the results from youngest to oldest.

i) Create a Parameterized, Named Query to find people between two ages in your "people model". Reference and run the query in your "people data" to find Person's in the desired age range.   (PMD4)

4) SADL also supports **Tests**

    a) Tests allows the validity of your knowledge base to be validated in an easily repeatable manner

    b) In-model tests: "Test: " followed by any expression that returns true or false

    c) Test suites: files ending in ".test"; when run will execute all tests in all included files

    d) Write the following tests in your "people data"

        i) Adam is declared to be a Man. Test that he is inferred to be a Person.

        ii) Test that the age of someone is what you said it was.

        iii) Test that the age of someone who is over 18 is greater than 18.

        iv) Test that Eve has a child Cain.

        v) Test that Abel has sibling Cain.

        vi) Test that Adam's children are Cain and Abel.   (PMD4b)

5) SADL supports **Rules**

    a) Jena provides many built-ins including (see https://jena.apache.org/documentation/inference/#RULEbuiltins)

        i) Math operators: *difference, max, min, product, quotient, sum*

            (1) Many of these have symbols in the grammar: -, *, / +

        ii) Comparison operators: *equal, ge, greaterThan, le, lessThan, notEqual*

            (1) Many of these have symbols in the grammar: =, ==, is, >=, >, <=, <, !=, is not

        iii) Get the current dateTime (as a string): *now()*

    b) Custom built-ins can be added. SADL provides these (and many more, see documentation):

        i) *abs, average, modulus, pow*, trig functions,

            (1) More symbols in the grammar supported: %, ^

        ii) Some particularly useful and important customer built-ins

            (1) getInstance (tied to *there exists* in the grammar)

            (2) countMatches

            (3) countUniqueMatches

            (4) list

            (5) noValue

            (6) one of

            (7) print

            (8) subtractDates

        iii) countMatches, *countUniqueMatches*, *list*, and *noValue* take graph patterns as arguments

            (1) noValue(s,p)   (also has grammar: *<s> of <p> is unknown*)

            (2) noValue(s,p,o)

            (3) list(s,p)

            (4) list(s1, p1, o1, s2, p2, o2, s3, p3,….) where *s2 p2, o2, s3*, etc., refer to prior arguments

c) Create a date-of-birth property in your "people model" that has Person as domain and date as range. Give dates of birth to several people in your "people data" who were not given ages.

d) Write a rule that assigns a person's age by taking the difference between the date now and their date of birth.

e) Write and execute a query asking for people's age.

f) Create some grandchildren in your "people data". Use *countMatches* and *print* to write a rule that will compute and print the number of grandchildren each person has.

i) The rule can be in your "people data"—it is not a very good metamodel rule (Why not?) **(PMD 5)**

6) SADL supports **Debugging**

    a) Keyword "Explain: " can be followed by:

       i) "Rule <rulename>." – will provide information about pattern matching as seen in Part 1

       ii) A graph pattern, e.g., "*Abel sibling Cain*" or "*area of MyCircle*".

    b) Temporarily put *print* built-in into rule conclusions to create a trace of rule condition satisfaction or firing, giving insight into order of reasoning events.

    c) Use "*Explain: <graph pattern>.*" and/or *print* in a rule conclusion to better understand why a rule did or did not fire.

7) SADL supports **Indefinite and Definite Articles**

    a) In English, *a, an* are indefinite articles, *the* is a definite article. These indicate *unbound* and *bound* variables, respectively.

    b) Example: "If I find **a book** I like then I will certainly read **the book**."

    c) Articles and class names can be used instead of variables in SADL rules (if enabled in Preferences).

       i) Consider how the rule for area of a circle can be rewritten

```
(1) Rule AreaOfCircle: if c is a Circle then area of c is PI * radius of c ^ 2.
(2) Rule AreaOfCircle: then the area of a Circle is PI * the radius of the Circle
    ^2.
```

       ii) See rule *findPadFillet1* below as a more complex example.

8) SADL supports creation of blank nodes using **there exists**

    a) Used to create new instances in a rule conclusion, possibly using the instance in additional triples. Example: **(PFE)**

```
Rule findPadFillet1:
if a Blending has edge (a first Intersection with edgeAdjacencyType TANGENT) and
    the Blending has edge (a second Intersection with edgeAdjacencyType TANGENT) and
    the second Intersection != the first Intersection and
    a first AbstractFace has edge the second Intersection and
    concave of the first AbstractFace is false and
    isFloorFace of the first AbstractFace is false and
    the first AbstractFace is a Cylindrical and
    a second AbstractFace has edge the first Intersection and
    the first Blending != the second AbstractFace and
    the Blending has adjacentFace a third AbstractFace and
    the Blending has facesShareEndPoint the third AbstractFace and
    not the first AbstractFace has facesShareEndPoint the second AbstractFace and
    the second AbstractFace has edge (an AbstractEdge with edgeAdjacencyType CONVEX) and
```

```
        the AbstractEdge != the first Intersection
then
        there exists a PadFillet with featureFace the Blending and
        the PadFillet has otherFace the second AbstractFace and
        the PadFillet has bottomFace the first AbstractFace and
        the PadFillet has bottomEdge the second Intersection and
        the PadFillet has otherFace "Pad Fillet".
```

a) In your "people model" create the class *Parent* as a subclass of *Person*.

b) Write a rule that says that if a person is a parent then there exists another person who is the child of the first person.

c) Create a new instance of *Parent* in your "people data" and add a query to find all parents and children. Test the model. Verify that the new person has an unnamed person as child.

d) Now give the new *Parent* a named child. Re-test the model. How is the result different?   (PMD6)

9) Logical axioms vs rules: compare these ways of defining Uncle. How are they different? Which do you find easiest to understand?

a) `A Man is an Uncle only if sibling has at least one value of type (child has at least 1 value).`

b) `Rule UncleRule: if a Man has sibling a Person and the Person has child a second Person then the Man is an Uncle.`

c) `Rule UncleRule: if a Man has sibling a Person and the Person has child a second Person then the Man is an Uncle and the second Person has uncle the Man.`

d) `Rule UncleRule: if x is a Man and x has sibling y and y has child z then x is an Uncle and z has uncle x.`

10) SADL has an **Expression Language**

a) The entire SADL grammar can be viewed at http://sadl.sourceforge.net/sadl3/SadlGrammar.pdf. The expression grammar begins near the bottom of page 8.

b) Which operator has precedence, *and* or *or*?

c) What operators have precedence over PropOfSubject, e.g., *age of Methuselah*?

d) SADL Constants

   i) PI

   ii) e

   iii) *known* (*not known*)

e) To assist in debugging expressions, use the *Expr: <epxr>.* Statement.

   i) SADL expressions are translated in steps

      (1) SADL -> "raw" Intermediate Form (IF-R)

      (2) IF-R -> "cooked" Intermediate Form (IF-C)

      (3) IF-C -> target language, e.g. SPARQL, Jena Rules, Prolog

   ii) Both IF-R and IF-C are displayed as Info markers on Expr statements (see http://sadl.sourceforge.net/sadl3/SadlIntermediateForm.html).

   iii) Try some of these in your "people data":

      (1) Expr: 1+2.

      (2) Expr: PI*3.

      (3) Expr: age of x.

      (4) Expr: age of a son of p.

          (5) Expr: age of the brother of Cain.   **(PMD6b)**

11) The **SADL Implicit Model**

    a) The SADL Implicit Model (<Project>/ImplicitModel/SadlImplicitModel.sadl) contains the definition of concepts that are <u>automatically</u> included, as if imported (actually imported in the OWL translation), by every .sadl file in the project.

    b) It is editable so that projects can have additional implicit concepts.

    c) UnittedQuantities

        i) The SADL grammar supports the inclusion of a "unit" word or quoted phrase after numbers

        ii) Depending on the preference "Ignore Unitted Quantities…", the numeric value and its units are included in the OWL translation as an instance of the class UnittedQuantity.

        iii) In your "people data", add a unit to *Adam's age*. What is the result? Why?

        iv) Revert your change to your "people data". Make a valid copy of your "people model" giving it a new name, e.g., PeopleModelUQ.sadl.
           (1) Note that you must give it a unique "uri" and "alias".
           (2) Change the range of *age* to *UnittedQuantity* in this new model.

        v) Create a new "people data", e.g., PeopleDataUQ.sadl and import the new metamodel.

        vi) Create an instance of person in your new "people data UQ" and assign an age with units. Save the model. Look at the OWL model for the new data model.

        vii) In preferences, set "Ignore Unitted Quantities…" to true. Clean the project. Now look at the OWL model for the new data model.

    d) Implied/expanded properties (see http://sadl.sourceforge.net/sadl3/ImpliedProperties.html)

        i) Not fully implemented in SADL V3.3.0

    e) Equations and semantic augmentation

        i) Equations are a type of knowledge capture finding its way into SADL

        ii) There are two types of equations supported by the SADL grammar

           (1) Equation has a signature and a body that specifies the computation

           (2) External [equation] has a signature and a URI/URL but no body; the specification is outside the project

        iii) Open SadlBuiltinFuncitons.sadl in the ImplicitModel folder

        iv) Open your SADL model containing the AreaOfCircle rule (from Part 1). Add an equation to your model to compute the area of a circle.
           (1) Equation areaOfCircle(float radius) returns float : radius^2*PI.

        v) Save your model and examine the OWL translation.   **(S2)**

        vi) Look in the SadlImplicitModel.sadl and find the property augmentedType.

        vii) What kinds of things can have the property augmentedType? (What is its domain?)

        viii)   What kinds of things can be described by DataDescriptors?

        ix) Examine the range of augmentedType. Examine the subclasses of AugmentedType, especially SematnicConstraint.

        x) Note: extensions to the SADL grammar and model and inference processors have reasoned about equations but equations aren't yet made executable by the available reasoners.

12) **Typed Lists** in SADL (see http://sadl.sourceforge.net/sadl3/SadlConstructs.html#TypedLists)

a) OWL does not fully support the concept of a typed list—an ordered group of similar things

b) In SADL, any class or primitive data type can be followed by the keyword "List" to declare a new typed list subclass or to declare an instance of a typed list

    i) **ChildrenList** is a type of **Person** List length 1-*.

    ii) **children** describes **Person** with values of type **ChildrenList**.

    iii) **MyFriends** is the **Person** List [Mark, Samantha].

    iv) See documentation for operations on lists.

    v) Note: typed lists were implemented for reasoning about knowledge and are not fully implemented in SADL Version 3.3.0.

13) **Default Values** in SADL

a) Not a part of OWL, but useful in many circumstances; if, after reasoning, members of a specified class do not have a value for a specified property, assign the default value.

b) *Note: default values were fully supported in older versions of SADL but the reasoner supporting default values is not currently part of SADL V3.*

14) Converting **OWL to SADL** using Import -> SADL -> Owl Files

a) An objective of the Semantic Web is to achieve semantic models shared across domains and organizations => if there is an accepted ontology that meets your needs, reuse it!

b) Sometimes people want to see an OWL ontology they are using in SADL syntax.

c) **Import the time-entry.owl file previously downloaded from http://sadl.sourceforge.net/owl/time-entry.owl into your project.**

    i) **File -> Import… -> SADL -> Owl Files, browse to download folder, select time-entry.owl, select destination project**

    ii) **What errors do you see?**

d) Note: The OwlToSadl code is a work in-progress.

15) Read and Write statements

a) **Write** (send data normally going to the console window to a file as well)

b) **Read** (read OWL or CSV data via a template (see below) into the current model; like an import but provides data ingestion in various formats and without mapping)

SADL Is Also an Integrated Development Environment

1) The SADL Language is an Xtext-based domain specific language (DSL). Xtext provides an integrated development environment (IDE) called the SADL IDE.

2) The SADL IDE provides the framework for "building" models to create OWL files and rule files

3) Whenever a valid SADL model is saved, a corresponding OWL model is saved in the OwlModels folder.

a) **Do you have a SADL model with an error in it? If not, add an erroneous statement and save.**

b) **Do you see the OWL for this model in the OwlModels folder?**

4) The OWL file can be serialized as RDF/XML, RDF/XML-ABBREV, N3, or N-TRIPLE (see Preferences)

a) **Change the OWL format to "N3". Do a clean/build on the project.**

b) **Look at the files in the OwlModels folder. Examine one of the .n3 files.**

c) **Note**: a bug currently prevents inference from working with other than one of the XML formats.

5) The SADL IDE provides authoring assistance including:
   a) Semantic coloring
      i) Keywords are magenta
      ii) Class names are dark blue bold
      iii) Property names are green bold
      iv) Annotation properties are green, not bold
      v) Instances are blue
      vi) User-defined datatypes are dark blue, not bold
      vii) Equation names are magenta bold
      viii) Variable names are pink
   b) Content assistance—control-space provides help in the form of templates and possible statement completions
   c) Quick fix—some problems have offered solutions which can be implemented with one click
   d) Hyperlinking of concepts within and across models
      i) **Put the cursor in a class or property name in your "people data" and press F3 or right-click and pick Open Declaration from the menu.**
      ii) **Open Temporal.sadl. Put the cursor in IntervalThing and press F3.**
   e) Renaming of concepts across a project
      i) **Select a class or property name in your one of your people models, right-click, and select "Rename Element". Change the name.**
      ii) **Look in models imported by or importing this model to verify the change across the project.**
      iii) **Select Edit -> Undo Rename Element.**
   f) Tiled and floating windows
      i) **Make an editor windows larger by double-clicking on the tab with the file name on it.**
      ii) **Open another editor window. Drag one of the windows, by its tab, to the left side or bottom of the screen.**
      iii) **Now drag it out of the main window. (If you don't have two monitors, make the main Eclipse window smaller than full-screen.)**
   g) Semantic validation of model structures
   h) Folding
      i) Useful to hide details and get the larger picture of large, complex statements.
      ii) **Go to the definition of Marriage. Right-click in the left margin of the editor window and select Folding -> Collapse All.**
6) Note: any keyword in the SADL grammar may also be used as a model-defined concept by "escaping it" with a preceding "^".
   a) **Open SadlImplicitModel.sadl. Examine the definition of UnittedQuantity and notice that "value", a keyword in the SADL grammar, is defined as a property.**
7) The SADL IDE provides test suites and a test suite editor to facilitate regression testing.
   a) **Create a file named Regression.test.**
   b) **Request Content Assistance, repeat and pick a .sadl file containing one or more Test statements.**

       c) **Select Test Model from the SADL menu.**

8) The SADL IDE architecture supports pluggable reasoner/translator pairs. Currently the following are available:

    a) Jena-based

        i) Reasoner: JenaReasonerPlugin or extension JenaGEReasonerPlugin

        ii) Translator: JenaTranslatorPlugin or extension JenaOptimizingTranslator (PoC)

    b) Prolog-based

        i) Reasoner: PrologReasonerPlugin (tuProlog)

        ii) Translator: PrologTranslatorPlugin

9) If the reasoner supports it, derivations can be recorded and reviewed, showing each inferred statement and how (which rules) were involved in the inference.

10) If GraphViz (or another implementation of the IGraphVisualizer interface) is installed, some visual graphing capability is supported by the SADL IDE.

11) The Eclipse framework provides additional useful capability:

    a) Project management including easy exporting and importing of projects

        i) To export a project:

           (1) File -> Export -> General -> Archive File

           (2) Select project and content (usually all), provide name and location of archive file. (Make sure to maintain directory structure.)

           (3) Click Finish.

        ii) To import a project:

           (1) File -> Import -> General -> Existing Projects into Workspace

           (2) Pick "Select archive file:", Browse to location of and select archive file.

           (3) Check project to be imported.

           (4) Click Finish.

    b) Integration with source code control and versioning systems like SVN and Git

        i) Extremely valuable, even if working alone. Creates a record of changes, insures against loss of work.

    c) Eclipse also maintains some history as set in Preferences -> General -> Workspace -> Local History

        i) **Right-click on one of your .sadl files that has had multiple edits**

        ii) **Select Compare With -> Local History…**

        iii) **Pick a timestamp, then step through the changes with Next Difference**

12) One SADL Project can reference another and thereby have its models available.

    a) Project -> Properties -> Project References

13) The SADL IDE has also been implemented with a browser user-interface, see http://sadl.sourceforge.net/sadl3/WebSADL.html.

Adding New Rule Built-ins

    1. Built-in mechanism is reasoner-dependent

    2. For Jena, user can define new built-ins by creating subclass of Jena BaseBuiltin class, providing required methods including bodyCall and/or headAction, building JAR file which includes Java

Services information, and placing on classpath (see
http://sadl.sourceforge.net/CustomJenaBuiltins.html)

It's a Tabular World: Mapping Tabular Data to OWL

1. Tabular data is stored in relational databases, Excel files, and CSV files. The first two can usually be converted to CSV with little effort.
2. Tabular data does not have complete and unambiguous semantics.
   a. Table name may supply meaning to a human user
   b. Column names may supply meaning to a human user
   c. Relationships can only be inferred (guessed) by someone based on table and column names
3. If each row of the table is independent of other rows, mapping is relatively easy and the data can be ingested into an OWL file or a triple store. We call such a mapping a **template** (see http://sadl.home.openge.ge.com/CsvDataImporter.html).
   a. A template specifies how to map from table to graph by supplying a set of triple patterns and putting column references in appropriate places in the triples
      i. The triples make the semantics explicit and complete
      ii. It may be necessary to create new nodes in the resulting graph
   b. If rows of data are truly independent, inference may be done during ingestion and the inferred model persisted.
   c. Ingestion may also be parallelized via Haddoup.

Modeling Processes

1. While a process is dynamic (involves change over time), the meta-model of a process is not dynamic for most purposes. Therefore processes can be modeled in a declarative language such as OWL.
2. An event may be considered as the smallest unit of a process.
3. Processes may:
   a. Consume
   b. Produce
   c. Change
   d. Have agents
   e. Have catalysts
   f. ….

Knowledge Server

1. While the SADL IDE supports model development, maintenance, testing, version control, collaboration, etc. It is often not a convenient mechanism to deploy knowledge-based applications.

2. The knowledge server, AKA SadlServer, provides a Java API-based or Web Services-based service for knowledge deployment (see http://sadl.home.openge.ge.com/SadlServerWebService.html and http://sadl.sourceforge.net/sadl3/KnowledgeServer.html).
   a. Knowledge bases are deployed as named services.
   b. A client connects to the service and specifies a service by name. The named service implies the knowledge base and entry point model for the session.
   c. The client sends scenario data to the service and queries for information, which may include inferences.
3. To facilitate deployment, the SADL IDE's configuration file, which identifies what reasoner and reasoner configuration to use, is used by the knowledge server.
4. For Web Services, a "kbase root" is specified and knowledge bases can be deployed simply by "dropping" them onto the server's file system.
5. The knowledge server, like the SADL IDE, can provide derivation information if supported by the reasoner.
6. Knowledge servers other than SadlServer can and are used
   a. Virtuoso: no inference capability—must infer on data ingestion and/or by SPARQL update
   b. Fuseki: part of the Apache Jena project

Sparql Graph: Exploring Semantic Models and Semantically Marked-up Data