Semantic Web Services and OOP toward Unified Services

Seiji Koide
Galaxy Express Corporation

OMG Model Driven Architecture

- Automatic Code Generation
  - And Reverse Engineering
- Platform Independent Model (PIM)
  - And Platform Specific Model (PSM)
- MetaObject Facility (MOF)
- XML Metadata Interchange (XMI)
- Common Warehouse Metamodel (CWM)
- Object Constraint Language (OCL)
OMG Model Driven Architecture

- Motivation of Code to Model (Mellor, Scott, Uhl, and Weise; MDA Distilled)
  - Desire to Higher Abstraction Level
  - Higher Reusability Level of Software Component
  - Interoperability of Model in Design
  - Costs to Assets

Model Driven Architecture to Ontology Driven Architecture in W3C

- A Lot of Benefits
  - Unambiguous domain models
  - Consistency checking
  - Validated models

Object Class Modeling to Object Behavior Modeling

Semantic Web Services + Method → Unified Service

- Semantic gaps between UML and OWL, especially MOF and OWL Metamodel
- How to generate executable code, procedures, and behaviors of objects
What is SWCLOS?

- An integration of SW and CLOS
- Toward integration of OWL and OOP
  - Class-instance concept in RDFS
  - An OWL entity like object
- OWL semantics is realized on top of CLOS.
Common Lisp Object System (CLOS)

Allegro Common Lisp® is a powerful dynamic object-oriented development system, suited to enterprise-wide, complex application development. Allegro CL's true dynamic object technology allows developers to generate leading-edge, mission-critical applications that are robust, extensible, and easy to evolve and deploy.

What is CLOS?

- A class is not a schema but an object.
- A class is an instance of a meta-class.
- A meta-class is also an object and instance of a meta-class.
- `cl:standard-class` is a native meta-class of every class.
- `cl:standard-object` is a native super-class of every class.
Why CLOS for Semantic Webs?

- rdfs:Class & owl:Class are meta-classes.
- rdfs:Resource & owl:Thing are supers
- CLOS allows to change a class of instance.
- Meta-Object Protocol (MOP) allows to customize class and meta-class behaviors.
- OO reflection conveys great flexibility into system.
Entailment Rules

Entailment Rules

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>If E contains</th>
<th>then add</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs9</td>
<td>uuu rdfs:subClassOf xxx . vvv rdf:type uuu .</td>
<td>vvv rdf:type xxx .</td>
</tr>
<tr>
<td>rdfs10</td>
<td>uuu rdf:type rdfs:Class .</td>
<td>uuu rdfs:subClassOf uuu .</td>
</tr>
<tr>
<td>rdfs11</td>
<td>uuu rdfs:subClassOf vvv . vvv rdfs:subClassOf xxx .</td>
<td>uuu rdfs:subClassOf xxx .</td>
</tr>
<tr>
<td>rdfs12</td>
<td>uuu rdf:type rdfs:ContainerMembershipProperty .</td>
<td>uuu rdfs:subPropertyOf rdfs:member .</td>
</tr>
<tr>
<td>rdfs13</td>
<td>uuu rdf:type rdfs:Datatype .</td>
<td>uuu rdfs:subClassOf rdfs:Literal .</td>
</tr>
</tbody>
</table>

\[
\text{(defclass} \ xxx ( ) ( )) \\
\text{(setq} \ vvv (make-instance} \ (\text{defclass} \ uuu \ xxx) ( )) \\
\text{(typep} \ vvv \ xxx \ \rightarrow \ \text{true})
\]

Subsumption Rule

Transitivity Rule

The Comparison of OWL/RDF and Object-Oriented Languages (from SETF)

<table>
<thead>
<tr>
<th>Classes and Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object-Oriented Languages</strong></td>
</tr>
<tr>
<td>Classes are regarded as types for instances.</td>
</tr>
<tr>
<td>Each instance has one class as its type. Classes cannot share instances.</td>
</tr>
<tr>
<td>Instances can not change their type at runtime.</td>
</tr>
<tr>
<td>The list of classes is fully known at compile-time and cannot change after that.</td>
</tr>
<tr>
<td>Compilers are used at build-time. Compile-time errors indicate problems.</td>
</tr>
</tbody>
</table>

From http://www.w3.org/2001/sw/BestPractices/SE/ODSD/
The Comparison of OWL/RDF and Object-Oriented Languages

<table>
<thead>
<tr>
<th>Classes and Instances</th>
<th>OWL and RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes are regarded as types for instances.</td>
<td>Classes are regarded as sets of individuals.</td>
</tr>
<tr>
<td>Each instance has one class as its type. Classes cannot share instances.</td>
<td>Each individual can belong to multiple classes.</td>
</tr>
<tr>
<td>change-class is applicable to instances at runtime.</td>
<td>Class membership may change at runtime.</td>
</tr>
<tr>
<td>Classes can be created and changed at runtime, because classes are also instances of meta-classes.</td>
<td>Classes can be created and changed at runtime.</td>
</tr>
<tr>
<td>Dynamic compiling and linking in runtime is available.</td>
<td>Reasoners can be used for classification and consistency checking at runtime or build-time.</td>
</tr>
</tbody>
</table>

Domain models consist of classes, properties and instances (individuals). Classes can be arranged in a subclass hierarchy with inheritance. Properties can take objects or primitive values (literals) as values.

SWCLOS Language

<table>
<thead>
<tr>
<th>SWCLOS Language</th>
<th>OWL and RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWL/RDF semantics &amp; entailments are implemented.</td>
<td>Classes are regarded as sets of individuals.</td>
</tr>
<tr>
<td>Multiple classing is implemented.</td>
<td>Each individual can belong to multiple classes.</td>
</tr>
<tr>
<td>change-class is applicable to instances at runtime.</td>
<td>Class membership may change at runtime.</td>
</tr>
<tr>
<td>Classes can be created and changed at runtime, because classes are also instances of meta-classes.</td>
<td>Classes can be created and changed at runtime.</td>
</tr>
<tr>
<td>Consistency checking and proactive entailments in runtime are implemented.</td>
<td>Reasoners can be used for classification and consistency checking at runtime or build-time.</td>
</tr>
</tbody>
</table>

Modified from http://www.w3.org/2001/sw/BestPractices/SE/ODSD/
The Comparison of OWL/RDF and Object-Oriented Languages

<table>
<thead>
<tr>
<th>Properties, Attributes and Values</th>
<th>OWL and RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties are defined locally to a class (and its subclasses through inheritance).</td>
<td>Properties are stand-alone entities that can exist without specific classes.</td>
</tr>
<tr>
<td>Instances can have values only for the attached properties. Values must be correctly typed. Range constraints are used for type checking.</td>
<td>Instances can have arbitrary values for any property. Range and domain constraints can be used for type checking and type inference.</td>
</tr>
<tr>
<td>Classes can encapsulate their members to private access.</td>
<td>Classes make their meaning explicit in terms of OWL statements. No imperative code can be attached.</td>
</tr>
<tr>
<td>Closed world: If there is not enough information to prove a statement true, then it is assumed to be false.</td>
<td>Open world: If there is not enough information to prove a statement true, then it may be true or false.</td>
</tr>
</tbody>
</table>

From http://www.w3.org/2001/sw/BestPractices/SE/ODSD/

---

The Comparison of OWL/RDF and Object-Oriented Languages

<table>
<thead>
<tr>
<th>CLOS Language</th>
<th>OWL and RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties, Attributes and Values</td>
<td>OWL and RDF</td>
</tr>
<tr>
<td>CLOS slots are defined locally to a class (and its subclasses through inheritance).</td>
<td>Properties are stand-alone entities that can exist without specific classes.</td>
</tr>
<tr>
<td>Instances can have values only for the attached properties. Values must be correctly typed. Range constraints may be used for type checking.</td>
<td>Instances can have arbitrary values for any property. Range and domain constraints can be used for type checking and type inference.</td>
</tr>
<tr>
<td>Classes may encode much of their meaning and behavior through imperative functions and methods.</td>
<td>Classes make their meaning explicit in terms of OWL statements. No imperative code can be attached.</td>
</tr>
<tr>
<td>Classes can encapsulate their members to private access.</td>
<td>All parts of an OWL/RDF file are public and can be linked to from anywhere else.</td>
</tr>
<tr>
<td>Closed world: If there is not enough information to prove a statement true, then it is assumed to be false.</td>
<td>Open world: If there is not enough information to prove a statement true, then it may be true or false.</td>
</tr>
</tbody>
</table>

Modified from http://www.w3.org/2001/sw/BestPractices/SE/ODSD/
The Comparison of OWL/RDF and Object-Oriented Languages

<table>
<thead>
<tr>
<th>SWCLOS Language</th>
<th>OWL and RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties, Attributes and Values</td>
<td></td>
</tr>
<tr>
<td>Property definitions are class objects, the extensions of property are object slots.</td>
<td>Properties are stand-alone entities that can exist without specific classes.</td>
</tr>
<tr>
<td>Flexible slot attachment is implemented. Range and domain constraints are also realized, and the entailments are implemented.</td>
<td>Instances can have arbitrary values for any property. Range and domain constraints can be used for type checking and type inference.</td>
</tr>
<tr>
<td>A number of OWL entailments are implemented.</td>
<td>Classes make their meaning explicit in terms of OWL statements. No imperative code can be attached.</td>
</tr>
<tr>
<td>Variables may be public or private, depending on programming.</td>
<td>All parts of an OWL/RDF file are public and can be linked to from anywhere else.</td>
</tr>
<tr>
<td>t means Boolean true, but nil is regarded unknown.</td>
<td>Open world: If there is not enough information to prove a statement true, then it may be true or false.</td>
</tr>
</tbody>
</table>

Modified from http://www.w3.org/2001/sw/BestPractices/SE/ODSD/

Straight-forward Mapping

- RDFS and OWL Class to CLOS Class
- Instance and Individual to CLOS Instance
- rdfs:Class & owl:Class to CLOS meta-class

Gain:

- Lisp native cl:typep and cl:subtypep is available for RDFS
- Linear search in the class precedence list
- Reflective programming by meta-class
Questions on Implementation of OWL

- Unknown Complete Set of Entailments in OWL
- How to Implement Proactive Entailment
  - Lisp is interactive.
  - One by one satisfiable check and entailment
  - Tableau algorithm looks like for Query.
  - What query should we make?
  - Proactive Entailment needs Entailment Rules to make a proper query in context.
  - OWL object by MOP

Satisfiability Checking

```
(defResource TheSpecialCourse (rdf:type owl:Class)
  (owl:intersectionOf
    food:RedMeatCourse
    (owl:Restriction (owl:onProperty food:hasFood)
      (owl:allValuesFrom food:Fruit)))))

(defIndividual No1SpecialCourse (rdf:type TheSpecialCourse)
  (food:hasFood food:Meat food:Bananas))

Error: Unsatisfiable by disjoint pair in
  (#owl:Class food:Fruit> #owl:Class food:RedMeat>) for
TheSpecialCourse food:hasFood
```

If a class is disjoint with a superclass, then the class is also disjoint with the subclasses.
Cardinality Checking

```lisp
(setf (slot-value vin:ElyseZinfandel 'vin:hasMaker) vin:Bancroft)
CARDINALITY CHECK ERROR: THE OPERATION IS NOT EFFECTIVE:
#<vin:Zinfandel vin:ElyseZinfandel> vin:hasMaker <- #<vin:Winery vin:Bancroft>

(get-form vin:Wine) →
{owl:Class vin:Wine (rdfs:label (rdfs:en "wine") (rdfs:fr "vin"))
(rdfs:subClassOf
  food:PotableLiquid
  {owl:Restriction (owl:onProperty vin:hasMaker)
    (owl:cardinality 1))
{owl:Restriction (owl:onProperty vin:hasMaker)
  (owl:allValuesFrom vin:Winery))
  (owl:Restriction (owl:onProperty vin:madeFromGrape)
  (owl:minCardinality 1))
  (owl:Restriction (owl:onProperty vin:hasSugar)
  (owl:cardinality 1))
  (owl:Restriction (owl:onProperty vin:hasFlavor)
  (owl:cardinality 1))
  (owl:Restriction (owl:onProperty vin:hasBody)
  (owl:cardinality 1))
  (owl:Restriction (owl:onProperty vin:hasColor)
  (owl:cardinality 1))
  (owl:Restriction (owl:onProperty vin:locatedIn)
  (owl:someValuesFrom vin:Region))))
```

Cardinality Unsatisfiability

```lisp
(defResource BlendedWine (rdf:type owl:Class)
  (rdfs:subClassOf
    vin:Wine
    (owl:Restriction (owl:onProperty vin:madeFromGrape)
      (owl:minCardinality 2))
    (owl:Restriction (owl:onProperty vin:hasColor)
      (owl:minCardinality 2))))

(defIndividual MyBlendedWine (rdf:type BlendedWine)
  (vin:hasColor vin:Red vin:White))

Error: Unsatisfiability by cardinality:BlendedWine vin:hasColor
```

The minimum maxcardinality and the maximum mincardinality so far are effective.

\[
\text{minCardinality} \leq \text{maxCardinality}
\]
owl::allValuesFrom Constraint

(defResource House (rdf:type owl:Class))
(defIndividual MyHouse (rdf:type House))
(defIndividual MyHomeMadeWine (rdf:type vin:Wine)
  (vin:hasMaker MyHouse))

Warning: Range entail: change class of #<House MyHouse> to #<owl:Class vin:Winery>.
Warning: #<House MyHouse> is additionally classified to (owl:Class vin:Winery).

MyHouse -> #<House.17 MyHouse>
(type-of MyHouse) -> (vin:Winery House)
(get-form vin:Wine) ->
  (owl:Class vin:Wine (rdfs:label (rdf:en "wine") (rdf:fr "vin"))
   (rdfs:subClassOf
    food:PotableLiquid
    (owl:Restriction (owl:onProperty vin:hasMaker)
     (owl:cardinality 1))
    (owl:Restriction (owl:onProperty vin:hasMaker)
     (owl:cardinality 1))
    (owl:Restriction (owl:onProperty vin:makerFromGrape)
     (owl:cardinality 1))
    (owl:Restriction (owl:onProperty vin:hasSugar)
     (owl:cardinality 1))
    (owl:Restriction (owl:onProperty vin:hasBody)
     (owl:cardinality 1))
    (owl:Restriction (owl:onProperty vin:hasFlavor)
     (owl:cardinality 1))
    (owl:Restriction (owl:onProperty vin:hasFlavor)
     (owl:cardinality 1))
    (owl:cardinality 1))

Family Ontology

(defIndividual Female (rdf:type Gender)
  (owl:differentFrom Male))
Warning: Special Entailing with domain: Gender is a owl:Class.
Warning: Entail by range: Male rdf:type owl:Thing.

(defResource Person (rdf:type owl:Class)
  (owl:intersectionOf
   Human
   (owl:Restriction (owl:onProperty hasGender)
    (owl:cardinality 1))))
Warning: Entail by range: hasGender rdf:type rdf:Property.
Warning: Change class by intersectionOf:Human rdf:type owl:Class

(defResource Woman (rdf:type owl:Class)
  (owl:intersectionOf
   Person
   (owl:Restriction (owl:onProperty hasGender)
    (owl:hasValue Female)))))
Family Ontology

(defIndividual QueenElizabethII (rdf:type Person)
(hasGender Female))

Warning: Entailed in refining: #<Person QueenElizabethII> to Woman.
Family Ontology

![Family Ontology Diagram]

Early Work in SWCLOS, A Semantic Web Processor on CLOS

- OWL Entities and CLOS objects are unified.
- SWCLOS can enjoy satisfiability checking and diverse constraints of OWL in OOP.

So, How about UML Modeling?
Please imagine, if program specification is defined by OWL like OWL-S.
A Simple Domain Model in UML Syntax to SWCLOS

(http://www.w3.org/2001/sw/BestPractices/SE/ODSD/)

<table>
<thead>
<tr>
<th>Product</th>
<th>price: float</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>products * 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PurchaseOrder</th>
<th>date: Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>totalPrice: float</td>
</tr>
<tr>
<td></td>
<td>orders * 0</td>
</tr>
<tr>
<td></td>
<td>customer 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer</th>
<th>email: string</th>
</tr>
</thead>
<tbody>
<tr>
<td>country: Country</td>
<td></td>
</tr>
</tbody>
</table>

Orders from countries with a free trade agreement are duty free

Warning: Entail by range: Product rdf:type rdfs:Class.

gx-user(2): (defProperty price (rdf:type owl:DatatypeProperty)
(rdfs:domain Product) (rdfs:range xsd:float))

Warning: Entail by range: date rdf:type rdf:Property.

gx-user(3): (defResource Customer (rdf:type owl:Class))

gx-user(4): (defResource PurchaseOrder (rdf:type owl:Class)
(rdfs:subClassOf (owl:Restriction (owl:onProperty date)
(owl:allValuesFrom Date)))
(rdfs:subClassOf (owl:Restriction (owl:onProperty products) (owl:minCardinality 1)))
(rdfs:subClassOf (owl:Restriction (owl:onProperty customer) (owl:allValuesFrom Customer)))
(rdfs:subClassOf (owl:Restriction (owl:onProperty customer) (owl:cardinality 1)))
(rdfs:comment "Orders from countries with a free trade agreement are duty free")
A Simple Domain Model in UML Syntax to SWCLOS
(http://www.w3.org/2001/sw/BestPractices/SE/ODSD/)

Domain Model as Meta-program that Specifies Programs

- OWL & DL provides a firm ground for programming.
- The OWL realization enables to share program specification between human and machine.
- Machine can support programming and (semi-)automatic code generation. Really?
A Simple Domain Model in UML Syntax to SWCLOS

(defmethod getPurchasesSum ((customer-individual Customer))
  (loop for order in (-> customer-individual orders)
    sum (loop for product in (-> order products)
      sum (coerce (-> product price) 'float))))

But, machines must know code patterns and simple task
ontology for automatic code generation, i.e., what sum is!

The Comparison of SWS and OOP

<table>
<thead>
<tr>
<th>Object-Oriented Languages</th>
<th>Semantic Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedures are locally executed.</td>
<td>Services are remotely executed across networks.</td>
</tr>
<tr>
<td>The execution of procedures are statically controlled by code.</td>
<td>The services are dynamically controlled by agents.</td>
</tr>
<tr>
<td>Procedures are coded as methods by hands. MDA aims automatic code generation.</td>
<td>Services are discovered, composed, and decomposed.</td>
</tr>
<tr>
<td>Methods are defined in class definition and cannot exist independently.</td>
<td>Services independently exist and are chosen by IOPE parameters.</td>
</tr>
</tbody>
</table>
The Comparison of SWS and OOP

<table>
<thead>
<tr>
<th>CLOS</th>
<th>Semantic Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedures are locally executed.</td>
<td>Services are remotely executed across networks.</td>
</tr>
<tr>
<td>The methods are chosen by the name and the parameter classes</td>
<td>The services are dynamically controlled by agents.</td>
</tr>
<tr>
<td>Procedures are coded as methods by hands. MDA aims automatic code generation.</td>
<td>Services are discovered, composed, and decomposed.</td>
</tr>
<tr>
<td>The CLOS Method is a generalized function and exists independently apart from classes.</td>
<td>Services independently exist and are chosen by IOPE parameters.</td>
</tr>
</tbody>
</table>

The Comparison of SWS and OOP

<table>
<thead>
<tr>
<th>Unified Services in SWCLOS</th>
<th>Semantic Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seamless between procedures and services</td>
<td>Services are remotely executed across networks.</td>
</tr>
<tr>
<td>The execution of procedures are dynamically controlled by instantiation and dispatch rules.</td>
<td>The services are dynamically controlled by agents.</td>
</tr>
<tr>
<td>Procedures are automatically coded by ontology. ODA aims automatic code generation.</td>
<td>Services are discovered, composed, and decomposed.</td>
</tr>
<tr>
<td>CLOS Method Protocol may be modified and unified with SWS by Meta-Object Protocol.</td>
<td>Services independently exist and are chosen by IOPE parameters.</td>
</tr>
</tbody>
</table>
Web Service Taxonomy

Method Definition in CLOS

(defmethod GetQValues ((precondition gxdomain:OperationMode) 
  (input gxdomain:SensorNameList) 
  (output QValueList)) 
  (error "This method cannot be invoked."))

(defmethod GetQValues ((precondition gxdomain:PipeCoolDownMode)) 
  (input gxdomain:SensorNameList) 
  (output QValueList))

<Web Service invoking body for CBR1>

(defmethod GetQValues ((precondition gxdomain:PipeCoolDownMode)) 
  (input gxdomain:SensorNameList) 
  (output QValueList))

<Web Service invoking body for CBR2>

(defmethod GetQValues ((precondition gxdomain:PipeCoolDownMode)) 
  (input gxdomain:SensorNameList) 
  (output QValueList))

<Web Service invoking body for CBR3>
Service Definition in SWCLOS

(defservice GetQValues ((precondition gxdomain:OperationMode) (input gxdomain:SensorNameList) (output QValueList))
(error "This method cannot be invoked."))

(defservice CBR1:GetQValues ((precondition gxdomain:PipeCoolDownM (input gxdomain:SensorNameList) (output QValueList))
  <Web Service invoking body for CBR1>
)

(defservice CBR2:GetQValues ((precondition gxdomain:PipeCoolDownM (input gxdomain:SensorNameList) (output QValueList))
  <Web Service invoking body for CBR2>
)

(defservice CBR3:GetQValues ((precondition gxdomain:PipeCoolDownM (input gxdomain:SensorNameList) (output QValueList))
  <Web Service invoking body for CBR3>
)

Can We Get New Dispatching?

Generic functions are instances with meta class funcallable-standard-class. Instances with this meta class are called funcallable-instances (FINs for short). They behave something like lexical closures in that they have data associated with them (which is used to store the slots) and are funcallable. When a funcallable instance is funcalled, the function that is invoked is called the funcallable-instance-function. The funcallable-instance-function of a funcallable instance can be changed.

Associated with each generic function is its discriminating function. Each time the generic function is called, the discriminating function is called to provide the behavior of the generic function. The discriminating function receives the full set of arguments received by the generic function. It must lookup and execute the appropriate methods, and return the appropriate values.
Can We Get New Dispatching?

Unified services are instances with meta class funcallable-standard-class. Instances with this meta class are called funcallable-instances (FINs for short). They behave something like lexical closures in that they have data associated with them (which is used to store the slots) and are funcallable. The function that is called when a funcallable instance is funcalled is called the funcallable-instance-function. The funcallable-instance-function of a funcallable instance can be changed.

Associated with each unified services is its discriminating function. Each time the unified services is called, the discriminating function is called to provide the behavior of the generic function. The discriminating function receives the full set of arguments received by the unified services. It must lookup and execute the appropriate services, and return the appropriate values.

Summary

• Model Driven Architecture in OMG is introduced.
• Emerging Activity on Ontology Driven Architecture in SETF in W3C is introduced.
• Early Work by SWCLOS is demonstrated.
• An example of mapping from UML class diagram to domain model is demonstrated.
• Automatic code generation requires task ontology or code pattern ontology.
• Unified Services of Web Services and OOP are proposed.