# Ontological Resolution and Object-Oriented Software Frameworks

(part the third)

# Knowledge

Interrogative Theory of Knowledge

- **Data**
  - symbols and tokens
- **Information**
  - Who? What? When? Where?
- **Knowledge**
  - How?
- **Understanding**
  - Why?

D/I/K is per Quigley and Debons

U is per Ackoff

I is gathered passively by a society of sensors

K is produced by reasoning about {K U I*} (also called procedural knowledge)

K is of higher utility than I since the # of problems that can be solved by K is greater than I.

U is of higher utility than K as the number of problems that can be solved by U is greater than K.

## Software as K Representation

Objects shelved in persistent storage (OODB) record experience.

Classes abstract instances and allow reasoning to be done on sets rather than singletons.

Programs (digraphs of classes, objects, and operations) encode procedural knowledge.

This is information.

This is the first step of knowledge.

This is procedural knowledge.

# Software as K Representation

Interrelated sets of classes and global objects that are specific to some domain or task are called **frameworks**

Frameworks embody the utility of a software development system.

Ie. OO-Prolog and Smalltalk as a **language** are almost completely **syntax** and **no semantics**.

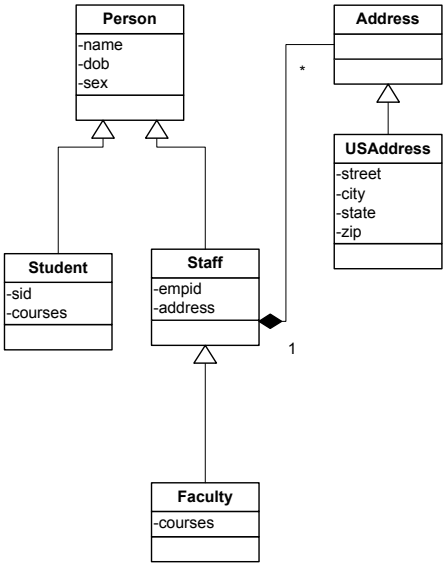
# Software as K Representation

The semantics of these (and other) languages are relegated to object-oriented frameworks.

Object-Oriented software is essentially frame-based logic.

- Frames have ***attributes*** and ***operations***
- Frames are related by ***generalization***, and ***aggregation***.

# Software as K Representation




UML is a network of frames that describe object-oriented software.

# Software as an Asset

Modern enterprises typically have a vast amount of resources allocated to software maintenance and development.

We would like to share software as knowledge, to lower the cost of creating maintaining and encoding new procedural knowledge.

This includes also software extensions and customizations such as workflows, spreadsheets, simulations, etc.

# Intuitive Goal of Resolution

Given two agents $\mathbf{A}_1$ and $\mathbf{A}_2$ that know ontologies $O_1$ and $O_2$, respectively.

$\mathbf{A}_1$ believes some degree of resolution has been achieved when a class or object known to $\mathbf{A}_2$ can be described by $O_1$.

Note that resolution is not commutative! Ie. (O1 ~ O2) != (O2 ~ O1)

# Intuitive Goal of Resolution

Slightly more formally,

From agent $\mathbf{A}_1$'s perspective, some degree of resolution between $O_1$ and $O_2$ occurs when some entity $\mathbf{y}$ in $O_2$ is equivalent to some entity $\mathbf{x}$ in $O_1$ or some entity $\mathbf{b}$ in $O_2$ is generalized by $\mathbf{a}$ in $O_1$.

# eog Relations

Intuitively we understand resolution in terms of of equivalence or generalization (***eog***).

**Ontological** resolution will operate on the subgraphs of $O_1$ and $O_2$ induced by the ***generalization*** relation

In both $O_1$ and $O_2$ ***generalization*** relations will be replaced by ***eog*** relations.

Ontological is in bold, because the outer (ontological) resolution will require an inner (entity) resolution that will be discussed later.

# Goal of Resolution

Graphically, an object-oriented class hierarchy (frame logic network) is a directed forest of **k** components

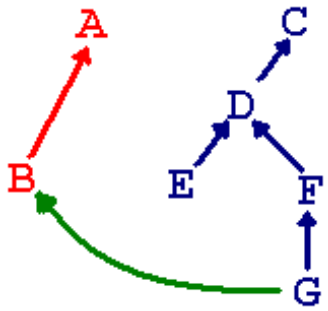
Given two disjoint ontologies, $O_1$ and $O_2$, with $k_1$ and $k_2$ components, respectively…

Resolution **(O3 = O1~O2)** is achieved when $k_3 < (k_1 + k_2)$.

I'm using the tilde for the resolution operator.

# Complete Resolution

$O_3$ is the complete resolution of $O_1$ and $O_2$ if for all frames $\mathbf{a} \in \mathbf{V}(O_1)$, there exists some ***eog*** path $\mathbf{r} \rightarrow \ldots \rightarrow \mathbf{a}$ where $\mathbf{r} \in \mathbf{V}(O_2)$

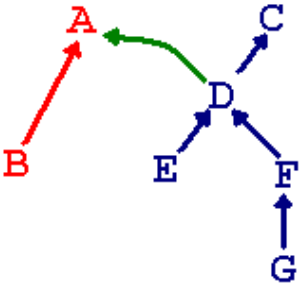



Every abstraction in $\mathbf{O_1}$ can be realized in $O_2$

Here we have ontologies O1 (red) and O2 (blue) and we can show that B is eog to G. Thus, every entity in O1 can be realized in O2.

# Partial Resolution

$O_3$ is the partial resolution of $O_1$ and $O_2$ if there exists frames $\{a_1 \ldots a_n\} \in V(O_1)$, such that there exists some eog path $r \rightarrow \ldots \rightarrow a_i$, $r \in V(O_2)$ and $n < |V(O_1)|$

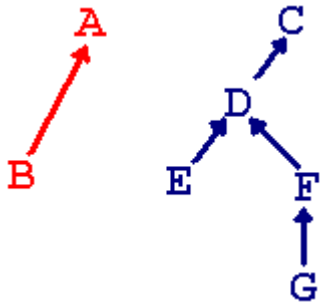



Here A in $O_1$ can be realized as D, E, F, or G in $O_2$ but B cannot be realized in $O_2$.

Here B cannot be shown to eog any entity in O2. Thus from A1's persepctive, O1 is not completely resolvable with O2.

# Null Resolution

$o_3$ is the null resolution of $o_1$ and $o_2$ if there does not exist $\mathbf{a} \in \mathbf{V}(o_1)$, such that there exists some eog path $\mathbf{r} \rightarrow \ldots \rightarrow \mathbf{a}_i$, $\mathbf{r} \in \mathbf{V}(o_2)$




From $\mathbf{A}_1$'s perspective no resolution is achieved between $o_1$ and $o_2$

# Inner Resolution

So far, all that has been shown is that two ontologies can be (outer) resolved if ***eog*** relations exist between entities **x** in $O_1$ and y in $O_2$.

Inner resolution is needed to find ***eog*** relations between two entities in disjoint ontologies.

How does one know if an ***eog*** relation exists between two entities?

# Inner Resolution

To demonstrate that class (**X eog Y**), we must define a mapping m from the attributes of **Y** **onto** the attributes of **X**.

Then, defining **Y** by extension, **Y** is the set of all instances $\{y_1 \ldots y_n\}$.

$(X\ eog_m\ Y)$ iff there is no $y_i$ with attribute $a_k$ that violates the $m_{jk}$ constraint on the **j**th attribute of **X**.

# Attribute Constraints

As an abstraction, a class covers objects that possess not only similar properties, but also similar constraints on those properties.

Such constraints can be formally defined with types and sets:

# Attribute Constraints

```
type
  TAge = 0..200; { A little optimistic }
  TCity = (Tucson, Phoenix, Yuma);
  Person = class(Tobject)
     name:string;
     age:TAge;
     city:TCity;
  end;
end;
```

This code fragment in Object Pascal (Delphi) – which has *excellent* type features illustrates that there is a class Person and the attributes of person are constrained to the range 0 to 200 for the age attribute and to an element the set of cities Tcity for the attribute city.

# Example

```
type
  TMaturityRange = 0..150;
  TWeightRange = 1..600;
  Patient = class
      first:string;
      last:string;
      maturity:TMaturityRange;
      weight:TWeightRange;
      address:string;
  end;
end;
```

# Example

$m_1$ = {Person.age = Patient.weight}

$m_2$ = {Person.age = Patient.maturity}

Given two instances of **Patient**

$y_1$ = {age: 24, weight: 180}

$y_2$ = {age: 41, weight: 412}

($y_2$ $m_1$ X) violates the **TAge** constraint on **Person**. For $m_2$ there is no $y_i$ that violates any constraints on **Person**.

This isn't entirely true, because the mapping functions m1 and m2 are not complete mappings from Y onto X…but for purpose of illustration, hopefully this works.

# Inner Resolution

Note that a valid eog mapping must be **onto `X`**.

A mapping m must define a relationship for every attribute of **`X`**, but not necessarily a relationship for every attribute of **`Y`**.

This mapping can be many to one, but not one to many.

(Y must contain at least as much information as X.)

That is attributes Y.a1 and Y.a2 can map onto X.a1, but *not* Y.a1 maps onto X.a1 and X.a2

# Implicit Semantic Constraints

If ontolgies $o_1$ and $o_2$ share some primitive sets (ie, the ordinal integers), proof techniques could be used to construct what would appear to be a valid mapping.

Beyond the formal constraints, there are often implied semantic constraints.

For a mapping to be correct, these constraints must be honored as much as the formal constraints.

# Implicit Semantic Constraints

Consider as a counter example attributes **`X.age`** and **`Y.height`**

If Y.height is measured in inches, the range of **`X.age`** would subsume **`Y.height`**

Thus a mapping could be constructed **`X.age=Y.height`** yet such a mapping would not honor the implied semantics of age and height.

# More Problems

There may exist several range and set consistent mappings from **`Y`** onto **`X`**.

Which of these is the **correct** mapping?

The ubiquitous use of INTEGER for quantitative data.

# More Problems

Many class definitions are not decomposed into ranges and sets, making formal analysis difficult.

Furthermore, enumerating some qualitative attributes may be theoretically possible, but impractical.

The ubiquitous use of INTEGER for quantitative data.

# More Problems

Consider an attribute **`name`**, which is constrained to be the name of some living person.

A set of the names of all living people could be theoretically be constructed, but is it feasible?

# Mapping by Experiments

It is not likely that real-world class definitions will be fully resolvable by constructing mappings that are logically consistent with explicit type constraints.

In the method of sketch-interpret, we can construct interpreters with domain specific knowledge to gather evidence for fuzzy reasoning.  (ie, BBN, DS, etc)

## Mapping by Experiments

Examples:

- Domain specific thesaurus to look for common terms in strings
- Analysis of variance on quantitative attributes of a large set of candidate instances
- Exploiting common objects, including global objects and co-occurrences in OODB.

Ie. A thesaurus that relates “video” to “graphics”

If we have two attributes X.height and Y.altitude, we may conclude that these are equivalent if we have sufficient samples to construct statistical measurements.

An example of this might be linking x in A1’s OODB to y in A2’s OODB if x.ssn = y.ssn (and we have already shown that x.ssn and y.ssn are eog) then, from that we can assist in resolving the remaining mappings.

# Strategies for Outer Resolution

Outer resolution requires as much as $\mathbf{v(o_1) \cdot v(o_2)}$ inner resolution attempts.

Each inner resolution attempt may have to explore as many as $\mathbf{m^{(n+1)}}$ mappings, where m is the number of (flattened) attributes of **Y** and n is the number of attributes of **X**.

A strategy to minimize computation would be advantageous.

The +1 is for the possible NULL mapping that is assigned to attributes of Y that have no semantic equivalent in X.

I don't envision the resolution process as occurring in "real" time.

# Strategies for Outer Resolution

A good strategy for resolution would minimize the **`n`** attributes in each resolution attempt.

A good strategy would also try to resolve classes that are deep in the hierarchy since all the classes above them are resolved as a corollary.

# Outline of a Good Strategy

With both generalization and aggregation links considered, reverse topologically sort $O_1$ into a queue **q**.

While **q** is not empty

1. remove entity **x** from **q**
2. resolve **x** with $O_2$
3. on **failure** remove all entities in q that have aggregation relations with **x**.
4. on **success** preserve mapping

The ordering is then deepest to shallowest.

# Outline of a Good Strategy

When resolving **x** with $o_2$, start at leaves of $o_2$ then work up, stopping on failure.

If no resolution was done, then **x** fails completely, if at least one resolution was successful, then **x** was resolved and the resolution returns success.

The ordering is then deepest to shallowest.

# Contributions

- Defined ontological resolution as applied to object-oriented software hierarchies.
- Identified pitfalls of reasoning about formal types when the class contains implicit semantic constraints.
- Suggested methods for fuzzy constructing fuzzy mappings
- Outlined a strategy to guide the resolution process, optimizing for computational efficiency.

My methods quite handily skipped the whole bit about operations.

# Conclusions

Object-Oriented software is a vessel of procedural **knowledge**

Object-Oriented databases store **information**.

Integrating software frameworks has significant utility

Software frameworks are amenable to ontological resolution

# Future Work

Improve resolution methodology to include inference about operational similarity.

Continue exploring contributions from…

- Database schema integration
- Programming by example
- Program equivalence