Abstracting Behavior in Ontology Engineering

Werner Kuhn
Center for Spatial Studies
University of California, Santa Barbara

spatial@ucsb
CENTER FOR SPATIAL STUDIES

The Ontology Engineering Bottleneck

meaningful communication

concepts

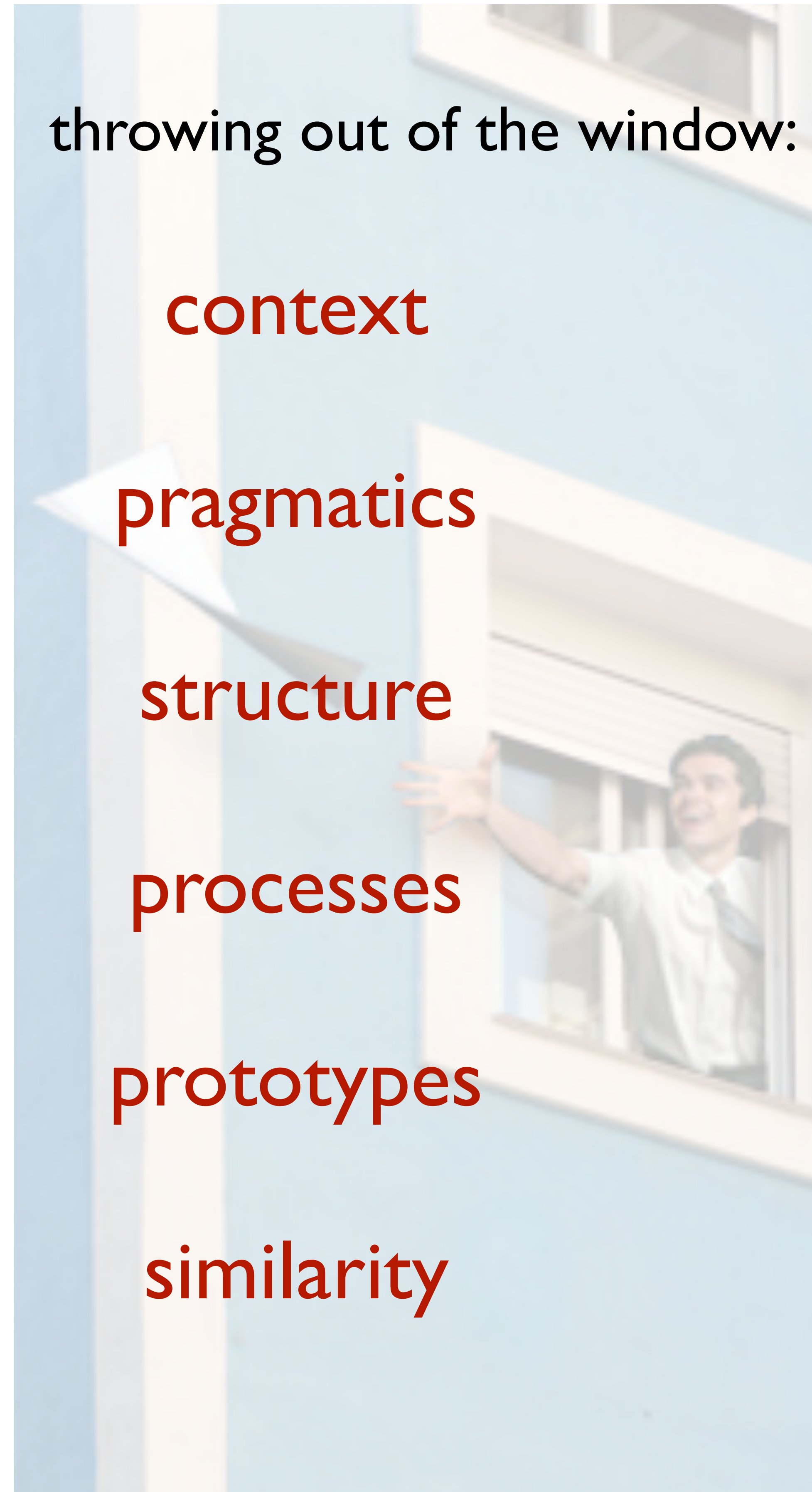
words

predicates

sortals

sets

$\sqcap \sqcup \sqsubseteq$



throwing out of the window:

context

pragmatics

structure

processes

prototypes

similarity

schema.org

Thing > Place > Landform > Mountain

A mountain, like Mount Whitney or Mount Everest

Property	Expected Type	Description
Properties from Thing		
description	Text	A short description of the item.
image	URL	URL of an image of the item.
name	Text	The name of the item.
url	URL	URL of the item.
Properties from Place		
address	PostalAddress	Physical address of the item.
aggregateRating	AggregateRating	The overall rating, based on a collection of reviews or ratings, of the item.
containedIn	Place	The basic containment relation between places.
event	Event	Upcoming or past event associated with this place or organization.
events	Event	Upcoming or past events associated with this place or organization (legacy spelling; see singular form, event).
faxNumber	Text	The fax number.
geo	GeoCoordinates or GeoShape	The geo coordinates of the place.
interactionCount	Text	A count of a specific user interactions with this item—for example, `20 UserLikes`, `5 UserComments`, or `300 UserDownloads`. The user interaction type should be one of the sub types of UserInteraction.
map	URL	A URL to a map of the place.
maps	URL	A URL to a map of the place (legacy spelling; see singular form, map).
photo	Photograph or ImageObject	A photograph of this place.
photos	Photograph or ImageObject	Photographs of this place (legacy spelling; see singular form, photo).
review	Review	A review of the item.
reviews	Review	Review of the item (legacy spelling; see singular form, review).
telephone	Text	The telephone number.

Acknowledgments to Ross Purves

The Promise of ODP

Design patterns support ontology engineering by

- ... increasing modularity and reusability
- ... providing building blocks for ontology design
- ... capturing semantics that may get lost in ontologies.

How should we identify, specify, and evaluate ODP ?

Requirements

Theories and tools to develop ODP should be

- ... generic (independent of domains and of ontology languages)
- ... expressive (more than ontology languages or visual languages)
- ... supporting composition
- ... theoretically sound
- ... easy to use (as easy as possible)
- …?

An old idea: abstracting behavior

Grouping concepts by shared behavior avoids some of these losses and modeling errors.

"If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck."

http://commons.wikimedia.org/wiki/File:Mallard2.jpg

http://upload.wikimedia.org/wikipedia/commons/a/a3/The_Spirit_of_43-Donald_Duck%2C_cropped_version.jpg

Ontology engineering with behavioral abstraction

To avoid vocabulary inflation and obesity…

- ... document actual uses of vocabularies as triples
  `showing who calls what a “duck”`

- ... treat these uses as inconsequential type declarations
  `same animal or toy could be called “mallard”`

- ... define type classes for shared behavior as ontology patterns
  `classes SWIM, QUACK, TALK,...`

- ... inherit behavior to types playing roles and reason about them
  `duck instantiates SWIM, QUACK, but not TALK`

The formal view: multi-parameter type classes

```
class Equal a where
    equal :: a -> a -> Bool
instance Equal Int where
    equal = primEqInt
instance Equal Point where
    equal = coordEqPoint
instance ...
```

- provide variables for concepts (type variables)
- treat concepts as role fillers (think of frames)
- specify them through shared behavior (think of interfaces)

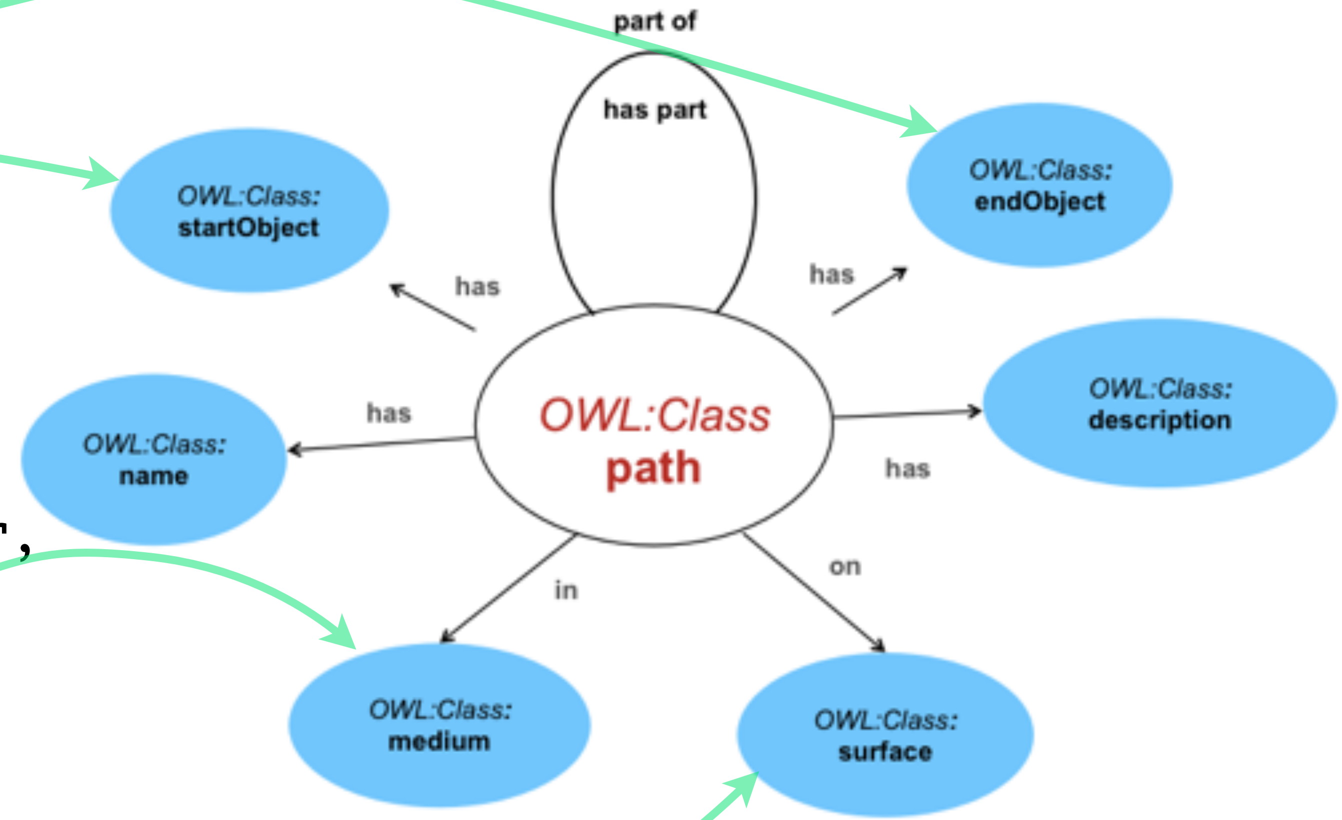
A Modeling Language: Haskell

The standard modern functional language

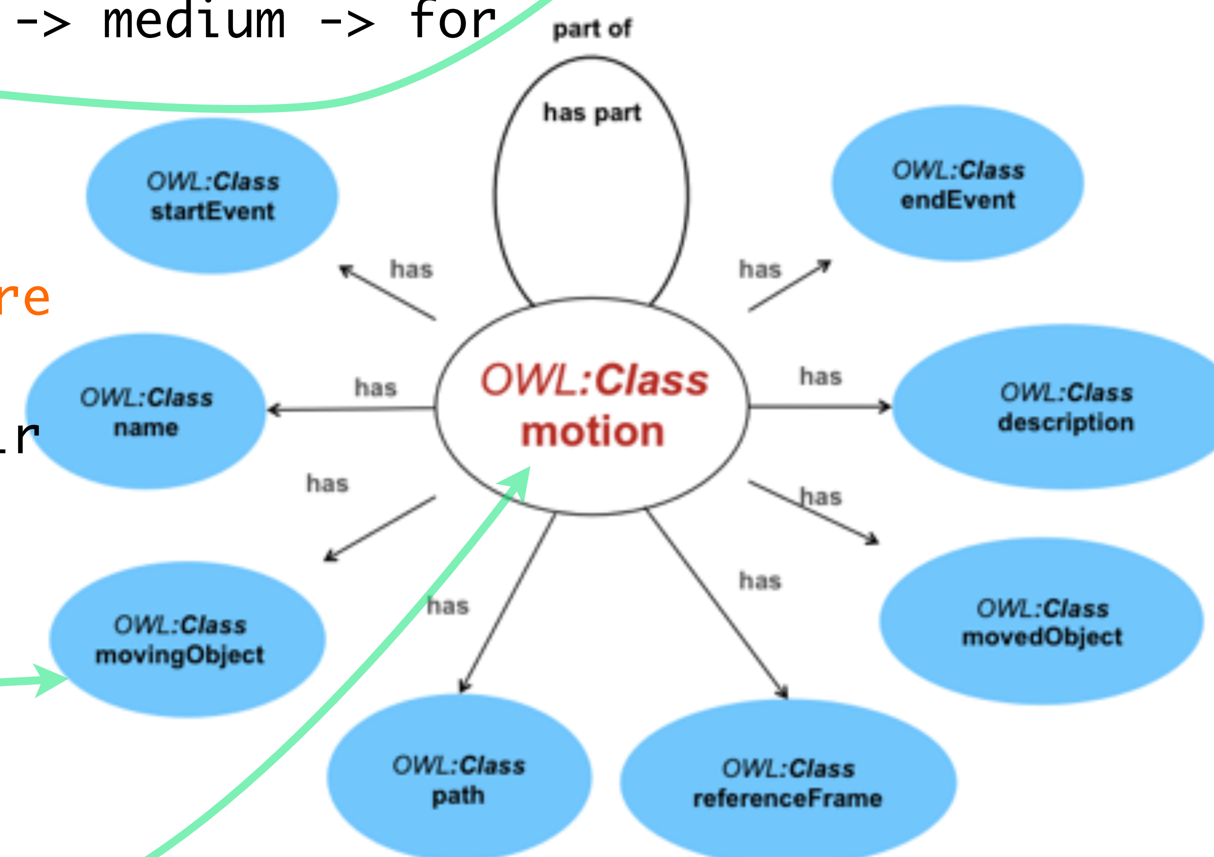
- clean, higher order type system
- executable algebraic specifications
- multi-parameter type classes

```
class (LINK link from to, SUPPORT from for, SUPPORT to for, CONTAINMENT medium link)
   => PATH for link from to medium where
  move :: for -> link -> from -> to -> medium -> for

instance PATH Car Link Node Node Air
```

Example: Path Pattern




```
class (LINK link from to, SUPPORT surface for,
      CONTAINMENT medium link) =>

  PATH for link from to surface medium where

    move :: for -> link -> from -> to -> surface -> medium -> for


instance PATH Car Highway Exit Exit Lane Air where

    *move car{@exit1} highway exit1 exit2 lane air

      = car{@exit2}


class LINK link from to where

    links :: link -> from -> to -> Bool
```



Conclusions

1. Ontologies in the semantic web emphasize set-based (and often highly context-dependent) typing
2. There is no good theory or practice to model higher level structure in ODP
3. I propose to design ODP around shared behavior, using type classes
4. The resulting ODP provide small theories, easily combinable, for big data
5. Specifications can be found, for example, in the GeoVoCamp series, but the idea is not limited to “geo” or spatial.

But, will this solve the
fax-number-of-mountains
problem?

Yes!   mountains are
endObjects of PATHs for hiking (not faxing)

Thank You!

Some References and Pointers

- Kuhn, W. (2010). Modeling vs encoding for the Semantic Web. *Semantic Web - Interoperability, Usability, Applicability*, *1*(1), 11–15
- http://www.haskell.org/haskellwiki/Haskell
- SWISH http://www.ninebynine.org/RDFNotes/Swish/Intro.html
- GeoVoCamps http://vocamp.org/wiki/GeoVoCampSB2014