

# Ontology-Based Data Discovery and Integration

**Shawn Bowers**

UC Davis, Genome Center  
([sbowers@ucdavis.edu](mailto:sbowers@ucdavis.edu))

**Deana Pennington**

LTER Network Office

The SEEK Team

(<http://seek.ecoinformatics.org>)



# Overview

---

## Introduction: Problems in discovery and integration (~10min)

- *Exercise: Data integration (~35min)*

## Ontologies

- Basics (~25min)
- Ontology Engineering (~5min)
- Representation Languages (~30min)
- *Exercise: Ontology development (~75min)*

## The Protégé Editor

- Overview (~15min)
- *Exercise: Protégé (~30min)*

## Using Ontologies (~25min)

# Overview

---

## ➤ **Introduction: Problems in discovery and integration (~10min)**

- *Exercise: Data integration (~35min)*

## Ontologies

- Basics (~25min)
- Ontology Engineering (~5min)
- Representation Languages (~30min)
- *Exercise: Ontology development (~75min)*

## The Protégé Editor

- Overview (~15min)
- *Exercise: Protégé (~30min)*

## Using Ontologies (~25min)

# Sharing data can be a good thing\* (even in Ecology)

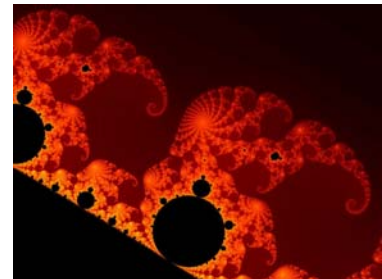
---

Data from observations and experiments remains at the core of Ecological research, but ...

**Open access** and **synthesis** can increase the value of data

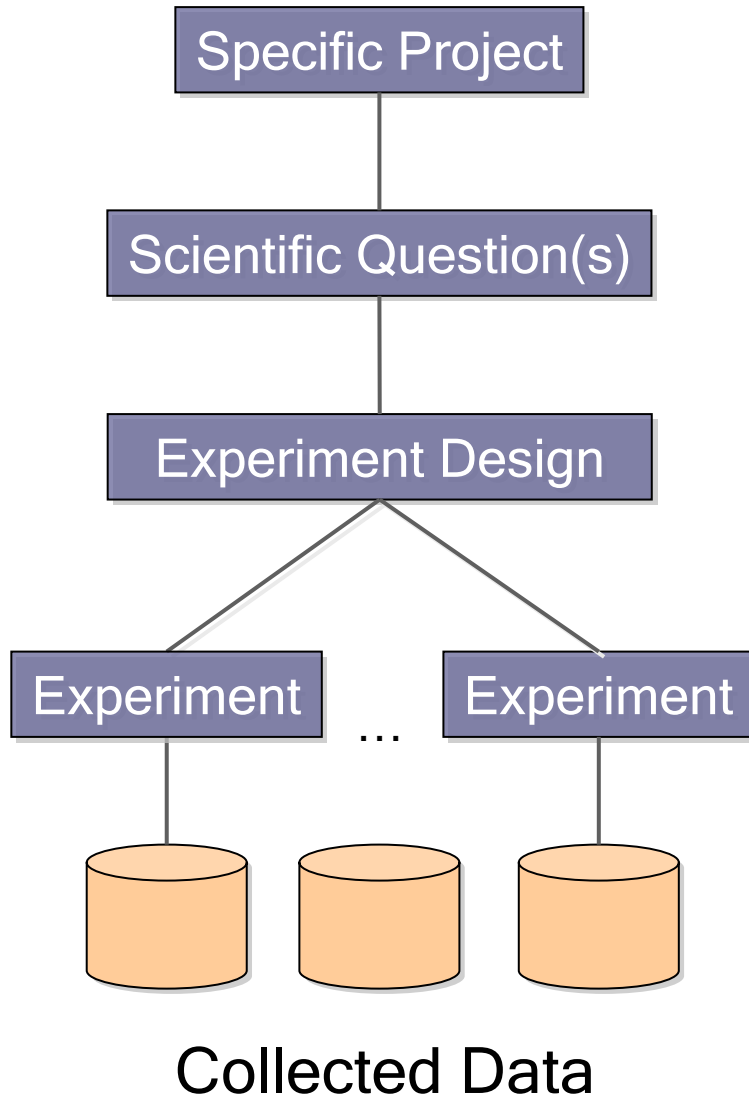
- broader perspective than possible from a single study
- reveal important regional and global patterns (and even new theories)
- data can be re-purposed, beyond the original intent of collection (recycling!)

\* see Bioinformatics



# “One-world” scenarios

---



... LTER project ABC

... Plant Productivity

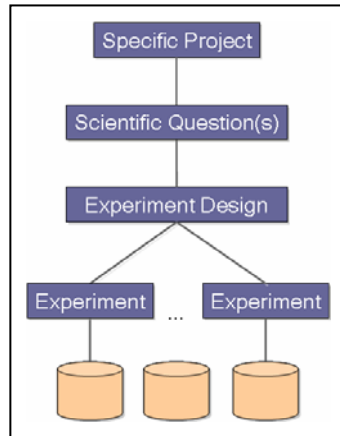
... Nitrogen Fertilization over specific plots

... Different times, plots, samples, etc.

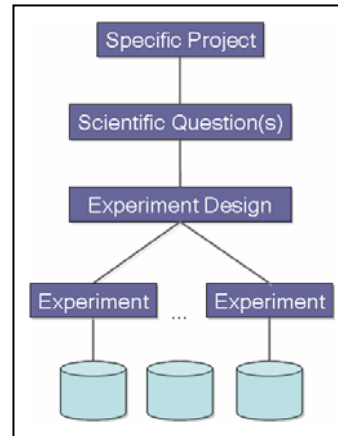
The database designs (tables) are typically uniform

# “Multi-world” scenarios (challenges)

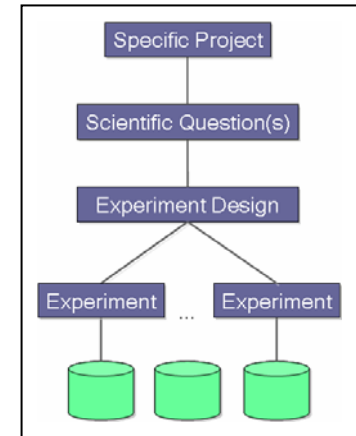
---



project x



project y



project z

Across projects (even experiments), different ...

- assumptions, scientific questions, standards (protocols, species, landscape, etc.), and **data structure & semantics** (schema)

The increasing emphasis on broad, synthetic studies underlines the need for greater coordination among scientists in **documenting** and archiving their data

# Data Discovery

---

You want to do a large-scale, cross-site Ecological analysis

- questions are typically based on an *intuitive* notion of what data is available
- relying on **ad hoc** and **inefficient** data-discovery approaches ... “social networks” or known work reported in the literature
- An initial challenge for data integration is simply determining what data is “out there”



# Data Integration

---

Once relevant data is found, now ... integration

Some issues in data integration:

- What should the integrated product look like?
- What is actually in the found datasets?
- How do the datasets conceptually relate?
- How to map each dataset to the integrated product?
- How to carry out the mappings?
- How to validate and document the decisions and process?

## Federated databases (CS)

- An integrated data warehouse/repository
- Common, integrated schema
- Issues often more challenging, e.g., no control of target schema

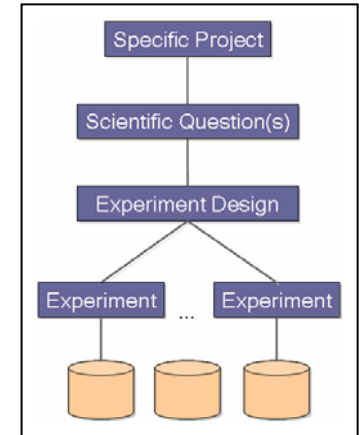




# Digression: Up-front Data Design

## Design Processes

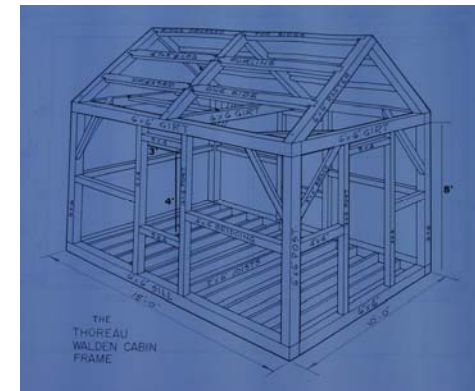
- Significant (and successful) software and engineering projects employ formal design processes ...
- Well-known processes exist for database design
- Typically go from conceptual to concrete



## Why use a design process for scientific data?

- Leads to “better” table designs (e.g., normalized)
- Identify potential problems early
- Help factor out unneeded information
- Design “for change”
- Leads to better documented projects
- Integration later can be easier
- Help organize project data and artifacts
- Opportunities for automation (tools)

We won't say much about different processes ...



# Overview

---

Introduction: Problems in discovery and integration (~10min)

➤ **Exercise: Data integration (~35min)**

## Ontologies

- Basics (~25min)
- Ontology Engineering (~5min)
- Representation Languages (~30min)
- *Exercise: Ontology development (~75min)*

## The Protégé Editor

- Overview (~15min)
- *Exercise: Protégé (~30min)*

## Using Ontologies (~25min)

# Overview

---

Introduction: Problems in discovery and integration (~10min)

- *Exercise: Data integration (~35min)*

## ➤ **Ontologies**

### ➤ **Basics (~25min)**

- Ontology Engineering (~5min)
- Representation Languages (~30min)
- *Exercise: Ontology development (~75min)*

## The Protégé Editor

- Overview (~15min)
- *Exercise: Protégé (~30min)*

## Using Ontologies (~25min)

# What is an “Ontology”?

---

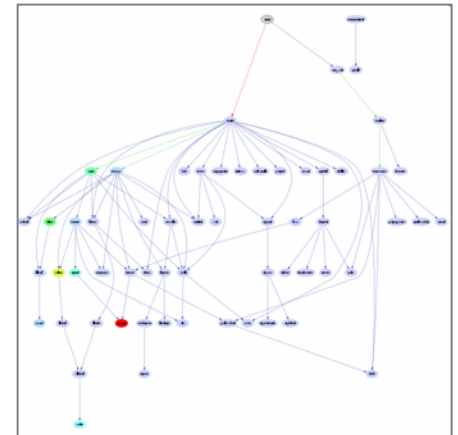
A vocabulary of **important terms**:

- Terms are surrogates for concepts
- Terms related to other terms (a “network”)
- For the purpose of “defining” concepts

Captures assumptions, current knowledge, and “theories” for a domain

For classifying and relating things

May be abstract or highly specialized



# Why Ontologies?

---

When we look at a dataset, we are only seeing part of the story ...

- How was the data collected?
  - What is assumed about the “variables” (attributes)?
  - How are variables related (dependencies)?
  - What is the scope and context of the dataset?
  - ...
- Metadata is meant to capture this (and more) ...  
... but usually uses informal terms and descriptions that can be ambiguous

# Looking Ahead

---

Why consider ontologies?\*

**Formal Metadata:** *metadata values, annotating data*

**Data Discovery:** *“structured” searching*

**Data integration:** *merging, uniform querying*

**Data/Analysis Design:** *high-level to implementation*

**Checking Assumptions:** *consistent “domain model”?*

And especially: **Automation!**

\* Or conceptual modeling in general ...

# “Informal” Ontologies

---

List of terms (*controlled vocabulary*)

– Natural language definitions (*glossary*)

List of terms and relationships (*thesaurus*)

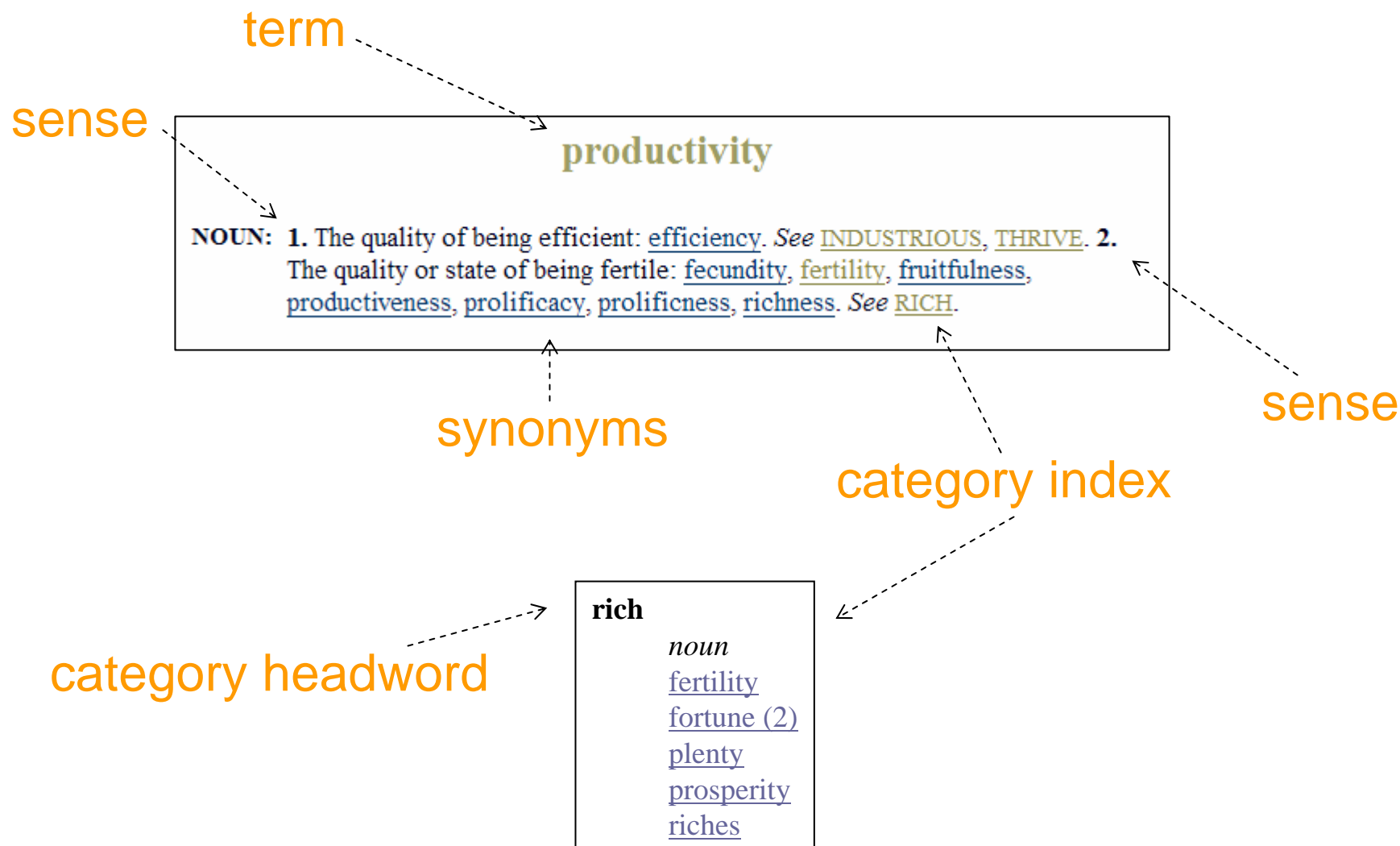
- synonyms
- antonyms
- related terms
- broader-narrower

E.g., see the NBI Thesaurus

(<http://thesaurus.nbi.gov/>)



# Thesaurus Example





# “Formal” Ontologies

---

- Very similar to thesauri\* .... Except:
  - Less on (superficially) relating meanings
    - Terms A and B mean similar things (related)
    - Where “related” is imprecise
  - More on capturing (focused) *meanings*
    - Every A is composed of one B and multiple C’s
    - Theorizing about the real-world or a system

\* In fact, Z39.19 is quite formal and “ontological”

# “Formal” Ontologies

---

- Definitions are precise (set-based)
  - Can make ontologies *easier* to define than thesauri (not as many combinations, but still a lot)
  - Requires a clear understanding of the domain

## productivity

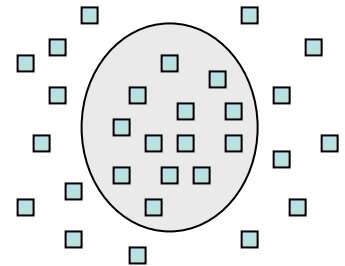
**NOUN:** 1. ~~The quality of being efficient: efficiency. See INDUSTRIOUS, THRIVE.~~ 2. The quality or state of being fertile: fecundity, fertility, fruitfulness, productiveness, prolificacy, prolificness, richness. See RICH.

# Basic Ontology Building Blocks

---

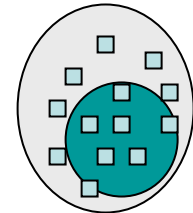
## Classes (concepts)

- A collection of objects that share certain *characteristics* (“if it looks like a duck ...”)
- Assigned names (symbols)



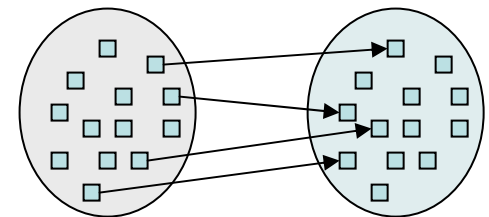
## Subsumption (*is-a*)

- “A subsumes B” means every B object an A object (... B is a subclass of A)
- Additional characteristics; more restrictions



## Properties (*has-a* / *part-of*)

- Represent a characteristic
- Assigned names (symbols)
- e.g., *has* Wings, *has-color* Yellow



# Tip 1: Classes versus Names

---

Classes represent *concepts*, **not** their names

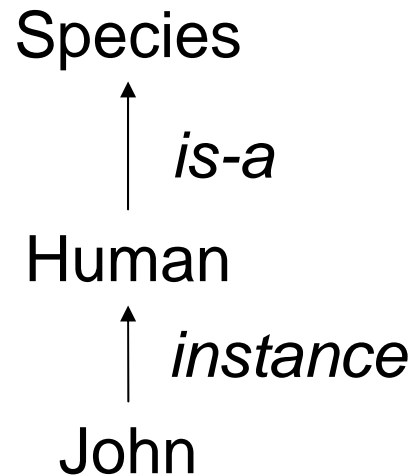
- Although, the name typically reflects the concept ... and partially helps to define it
- The class name can change, but still refer to the same concept
- Synonyms for the same concept are not different classes

Common problems:

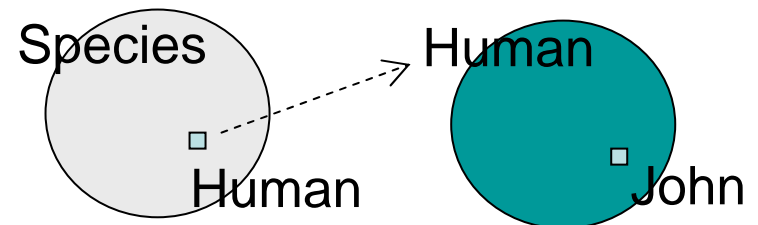
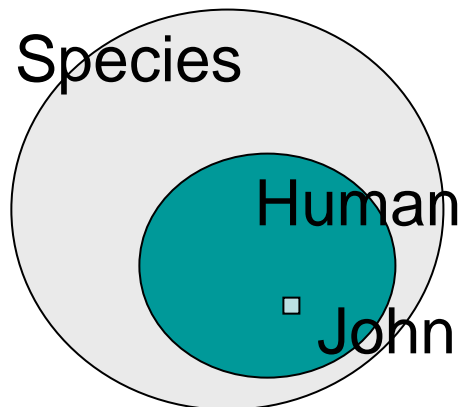
- The system uses the structure, not the names ...
- But people often “overload” names with implicit meaning

## Tip 2: Subsumption is not instantiation (membership)

---

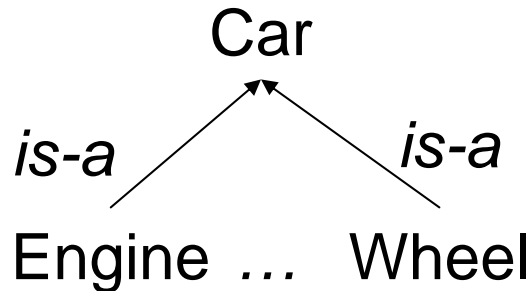


- If A subsumes B, then every B is an A
- Every human, in this case, must also be a species
- But “John” is not a species



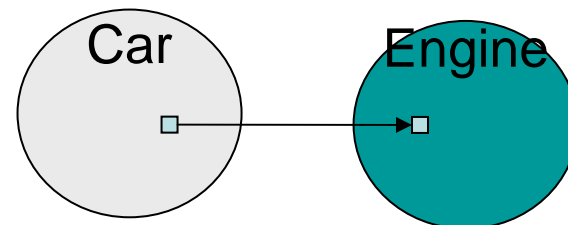
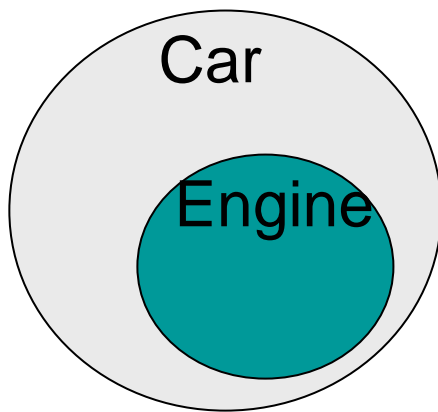
# Tip 3: Subsumption is not part-of

---



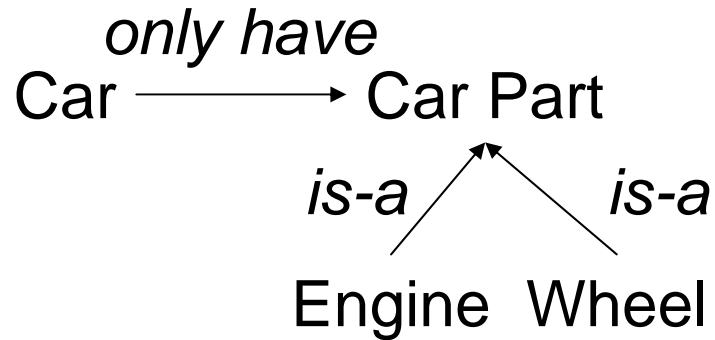
The difference between subsumption and part-of can be tricky ...

- What are essential properties of Cars?
  - E.g., that they accommodate people?
- Are these also essential for Engines?

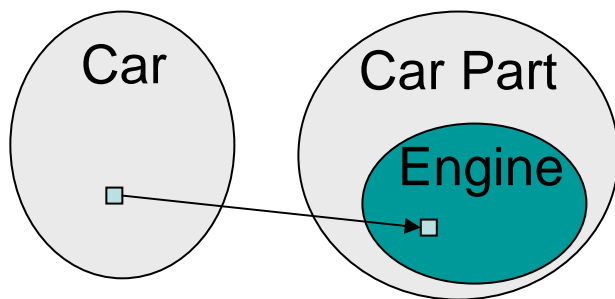


## Tip 4: Subsumption is not disjunction

---

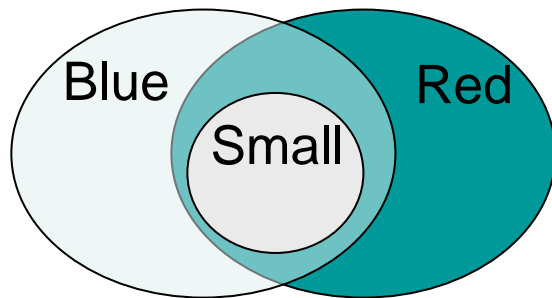
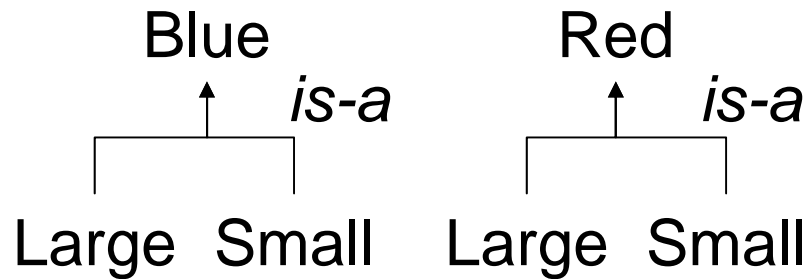


- “Car Part” is an artificial concept to denote all car parts (engines, doors, etc.) using subsumption
- The introduction of artificial classes in this way can lead to problems:
  - Is every engine necessarily a car part?
  - E.g., someone may use it in an airplane

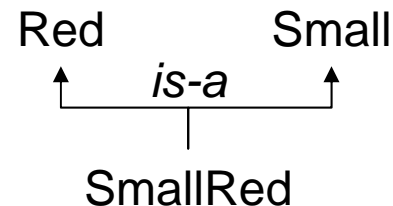


# Tip 5: Subsumption is not cross-classification

---



- We want say that there are small red things and small blue things
  - But Blue and Red subsume Small
  - So, all Small things are both Blue and Red
- Color and Size are unrelated (independent) classifications





# Overview

---

Introduction: Problems in discovery and integration (~10min)

- *Exercise: Data integration (~35min)*

## ➤ **Ontologies**

- **Basics (~25min)**

### ➤ **Ontology Engineering (~5min)**

- Representation Languages (~30min)
- *Exercise: Ontology development (~75min)*

## The Protégé Editor

- Overview (~15min)
- *Exercise: Protégé (~30min)*

## Using Ontologies (~25min)

# How do we build ontologies?

---

First, we have to answer some important questions:

**What is the **purpose** of the ontology?**

- For capturing a particular use of terms, knowledge, a theory, ...

**What is the **scope** of the ontology?**

- What is the target domain, what is not included, ...

**How will input **knowledge** be acquired?**

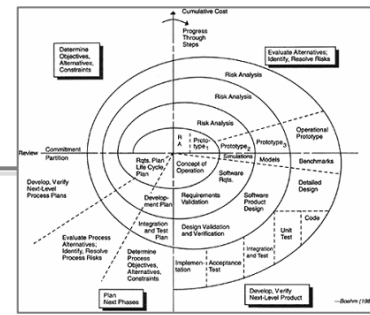
- Who will build it, what sources will be used, ...

**How will the ontology be **evaluated**?**

- How will we know it is correct, complete, ...

# How do we build ontologies?

## General stages of ontology development



**Requirements:** Typically, prose describing what it should contain, what questions it should answer (scope)

**Conceptualization:** Informally define domain concepts and basic relationships. Possibly give natural-language definitions.

**Formalization:** Formalize the conceptualization, defining is-a relations and “axioms”. Independent of a representation language.

**Implementation:** Select a representation language and use it to capture formalization.

**Testing:** Check consistency, completeness, conciseness. Verify against requirements and conceptualization.

**Documentation:** Document stages, assumptions, etc. Each concept and relationship should have a description

– Application of stages typically **iterative**, and may involve **refactoring**

# Overview

---

Introduction: Problems in discovery and integration (~10min)

- *Exercise: Data integration (~35min)*

## ➤ **Ontologies**

- **Basics (~25min)**
- **Ontology Engineering (~5min)**
- **Representation Languages (~30min)**
- *Exercise: Ontology development (~75min)*

## The Protégé Editor

- Overview (~15min)
- *Exercise: Protégé (~30min)*

## Using Ontologies (~25min)

## The Resource Description Framework (RDF)

- a W3C recommendation (... the makers of HTML)
- An extremely simple model\* that consists of **individuals** (objects), **properties**, and **values**

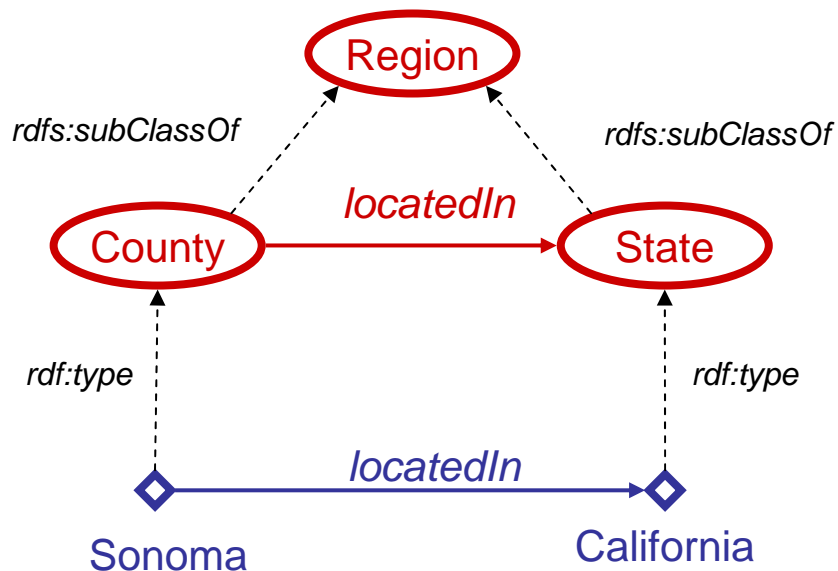


- Individuals and properties are uniquely named
- Properties are directed
- Properties link an individual to another individual or a value

\* We use OWL conventions

## RDF Schema adds class and property definitions

- A detail: These are actually denoted via special properties



- ... 'County' and 'State' are classes
- ... 'locatedIn' is a property from 'County' to 'State'
- ... 'Sonoma' is a particular 'County'
- ... 'California' is a particular 'State'
- ... 'County' and 'State' are both kinds of 'Region'

*rdfs:subClassOf* = is-a

*rdf:type* = instance

Significantly extends RDF Schema with lots and lots of **constraints** on class membership and properties

## Class property restrictions

- Constrains the participation of individuals in properties
- For example, min and max cardinality



... Every 'County' is located in *at most one* 'State' (max cardinality = 1)

... Every 'County' is located in *at least one* 'State' (min cardinality = 1)

Sometimes we write this 1:1 (meaning min:max) ... but note that that not all constraints can be conveniently shown graphically

# OWL (Web Ontology Language)

---

Are these correct?

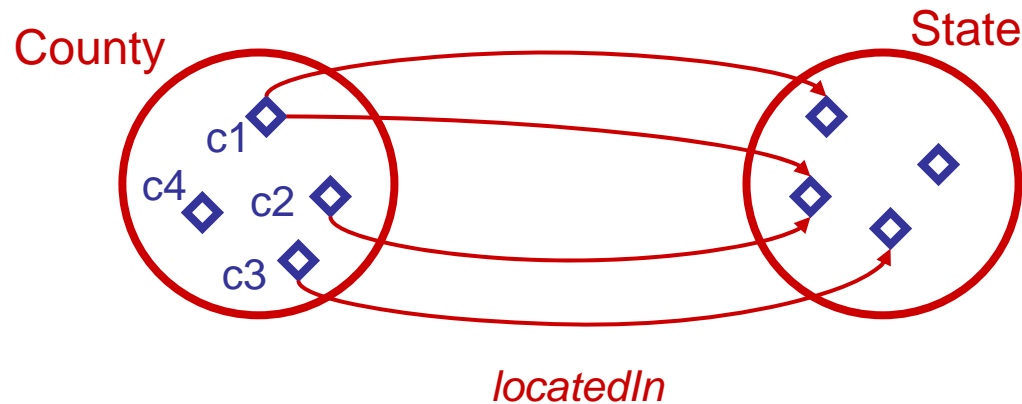




# More on constraints

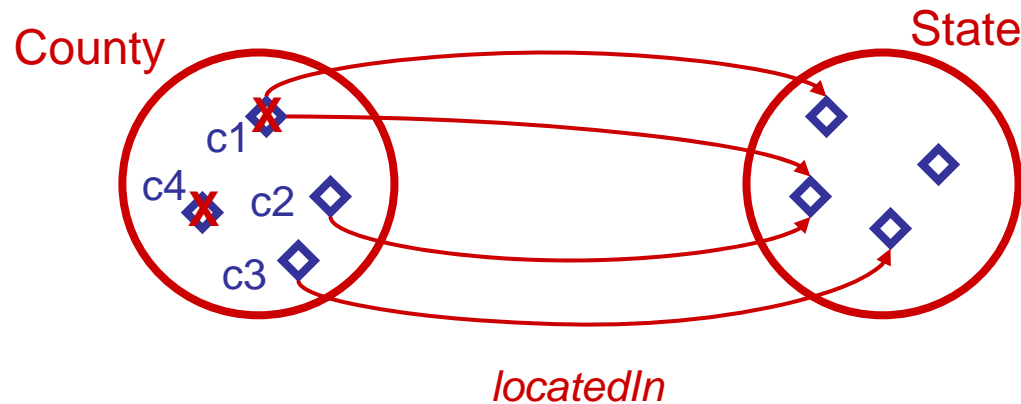
---

- Given the 1:1 cardinality constraints on `locatedIn` for 'County', which of the following 'County' individuals is incorrect?



# More on constraints

- Given the 1:1 cardinality constraints on `locatedIn` for 'County', which of the following 'County' individuals is incorrect?



To be a 'County', must have exactly 1 `locatedIn` property to a 'State' ...

# Exercise: Formalizing wine

---

An (informal) ontology of wine:

- Wines are potable liquids made by wineries within regions and with specific vintages
- Wines are characterized by the type of grape they are made with, their color (white, rose, red), their sugar (dry, offdry, or sweet), their body (light, medium, full), and their flavor (delicate, moderate, strong)
- Sauvignon Blanc, Merlot, Pinot Noir, and Riesling are types of wines

# Exercise: Formalizing Wines

---

With a partner, take 10 minutes and try to define a “formal” ontology for the wine example

- Select two or three concepts
- Identify some relationships between them
- List any cardinality constraints that exist between them

# Other types of OWL constraints

---

## Class property restrictions

- allValuesFrom (“only”,  $\forall$  locatedIn.State)
  - someValuesFrom (“some”,  $\exists$  locatedIn.State)
  - minCardinality (“at least n”,  $\geq n$ )
  - maxCardinality (“at most n”,  $\leq n$ )
  - Cardinality (“exactly n”,  $= n$ )
  - hasValue (“equals x”,  $\in x$ )
- Class expressions (constructors)
    - Necessary ( $\sqsubseteq$ ) and sufficient ( $\sqsupseteq$ ) conditions
    - Class intersection, union, and complement (constructors)

# Examples (Description Logic)

---

$\text{Wine} \sqsubseteq \text{PotableLiquid} \sqcap \exists \text{hasColor}.\{\text{Red}, \text{Rose}, \text{White}\}$

The class Wine is a sub-class of PotableLiquids that have at least one (exists one) hasColor property whose values are either Red, Rose, or White (necessary)

$\text{WhiteWine} \equiv \text{Wine} \sqcap \exists \text{hasColor}.\{\text{White}\}$

WhiteWines are exactly Wines whose color is White (necessary and sufficient)

$\text{WhiteBurgandy} \sqsubseteq \text{WhiteWine} \sqcap \text{Burgandy}$

The set of WhiteBurgandy wines is a subset of the set of WhiteWines intersected with Burgandy wines

$\text{SauvignonBlanc} \sqsubseteq \text{WhiteWine} \sqcap =1 \text{ madeFromGrape.SauvignonBlancGrape}$

# Digression: “Ontological” questions

---

(Philosophy) An ontological theory can answer “ontological” questions

- Is Merlot a potable liquid?
- Are there wines made of things other than grapes?
- How are Pinot Gris and Pinot Noir related?
- Are there white wines that are dry, full, and strong made in Napa Valley?

There are more uses of ontologies ...

# Overview

---

Introduction: Problems in discovery and integration (~10min)

- *Exercise: Data integration (~35min)*

## ➤ **Ontologies**

- **Basics (~25min)**
- **Ontology Engineering (~5min)**
- **Representation Languages (~30min)**
- ***Exercise: Ontology development (~75min)***

## The Protégé Editor

- Overview (~15min)
- *Exercise: Protégé (~30min)*

## Using Ontologies (~25min)



# Ontology Engineering

---

## Develop an Ontology

1. Form groups of 3-4 people
2. Use the BEAM project discussed earlier
3. Capture (a part of) an ontology for the project on whiteboard/poster board as one or more diagrams
  - Define the concepts, properties, and constraints
  - Try to relate the data sets to your ontology ... how can someone use your ontology to understand the assumptions behind the experiment and data sets?

# Overview

---

Introduction: Problems in discovery and integration (~10min)

- *Exercise: Data integration (~35min)*

## Ontologies

- Basics (~25min)
- Ontology Engineering (~5min)
- Representation Languages (~30min)
- *Exercise: Ontology development (~75min)*

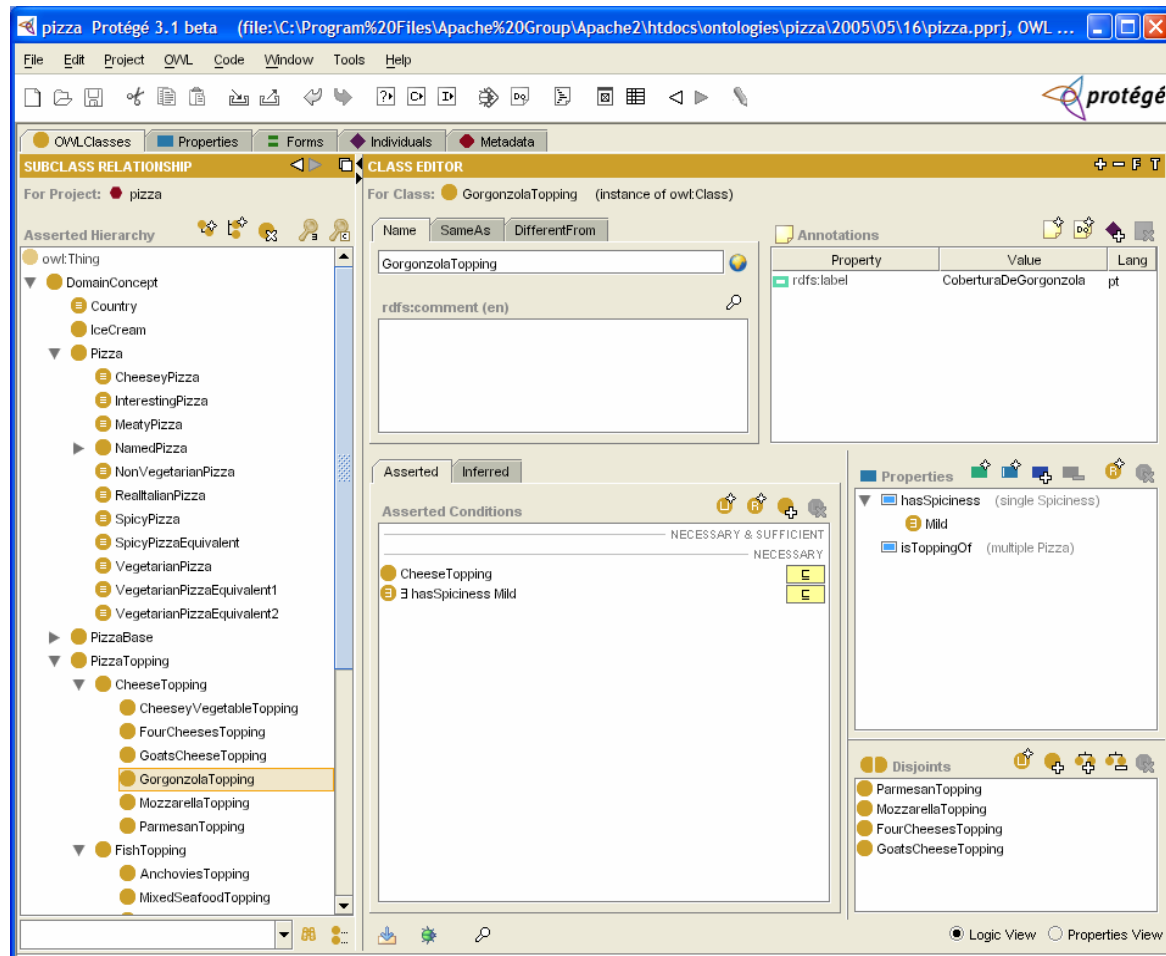
## ➤ The Protégé Editor

- **Overview (~15min)**
  - *Exercise: Protégé (~30min)*

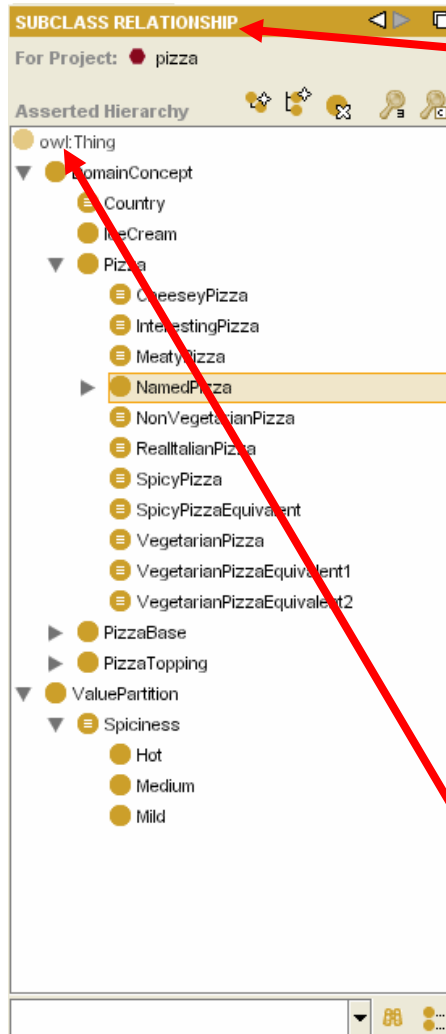
Using Ontologies (~25min)

# Protégé

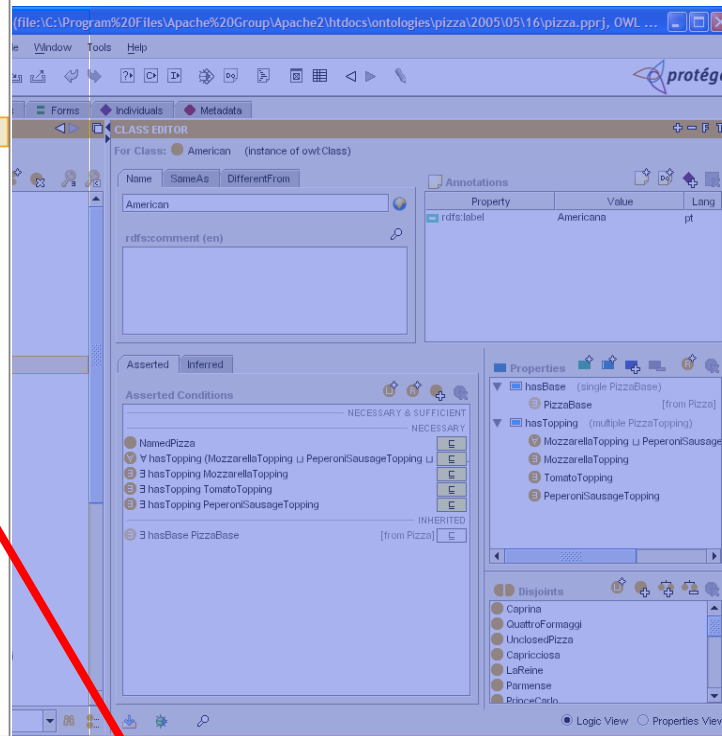
- An open-source, freely available ontology editor



# Class Hierarchy

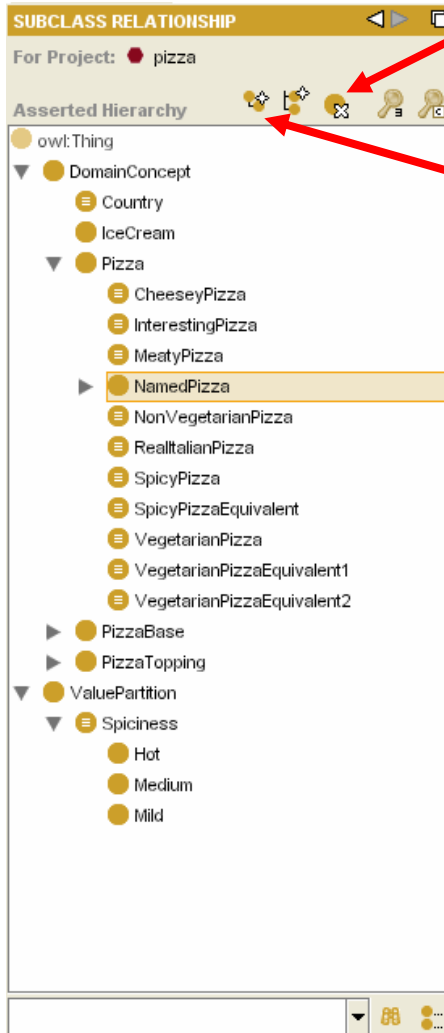


Subsumption hierarchy



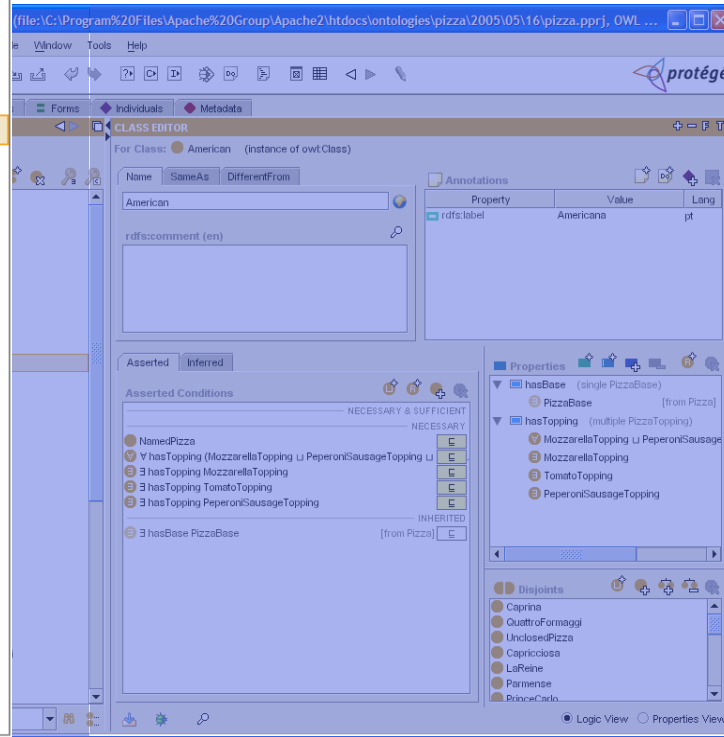
owl:Thing is the root class

# Adding Classes



Delete selected Class

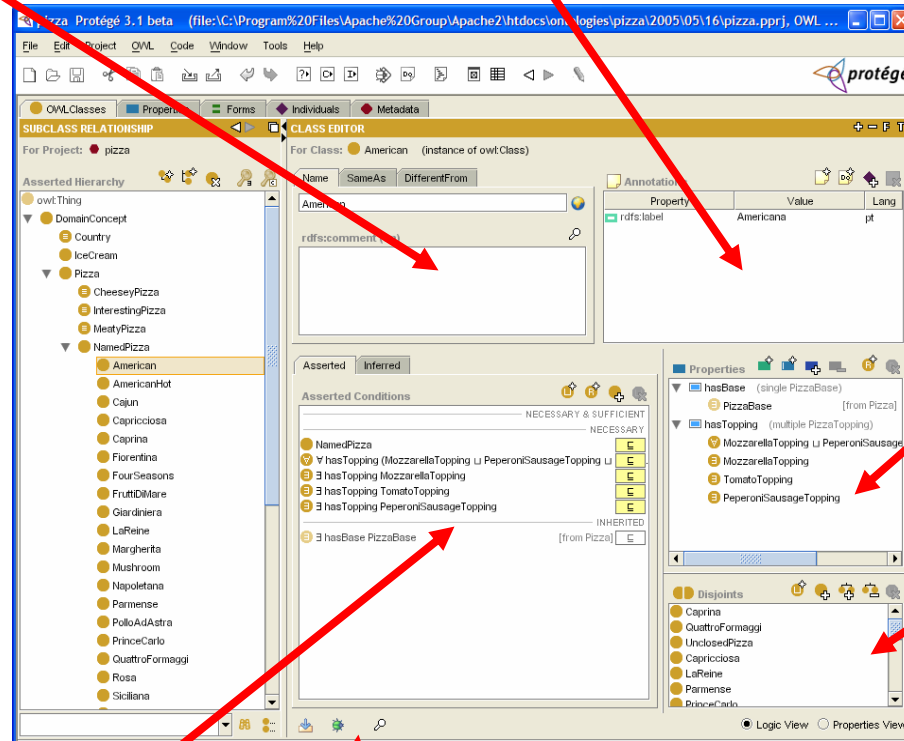
Create New (sub) Class



# Class Editor (Tab)

Class annotations (for class metadata)

Class name and documentation



Properties  
“available” to  
Class

Disjoints  
widget

Conditions Widget

Class-specific tools (find usage etc)

# Saving OWL Files

OWL = easy to make mistakes = save regularly



1. *Select File → Save Project As*  
*A dialog (as shown) will pop up*

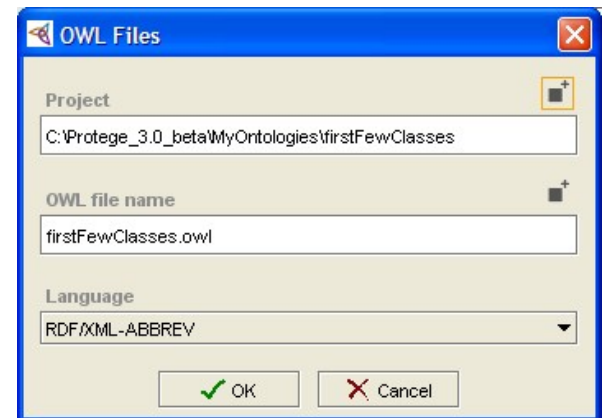
2. *Select a file directly by clicking the button on the top right*  
*You will notice that 2 files are created*  
*.pprj – the project file*

*this just stores information about the GUI  
and the workspace*

*.owl – the OWL file*

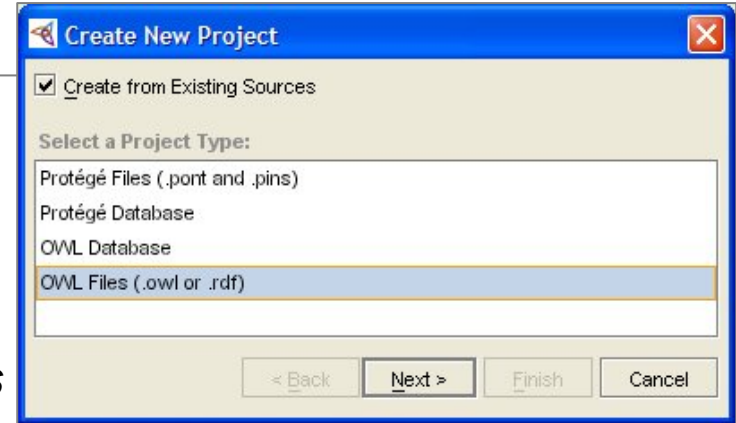
*this is where your ontology is stored in  
RDF/OWL format*

3. *Select OK*



# Loading OWL Files

1. *If you only have an OWL file:*
  - **File** → **New Project**
  - Select **OWL Files** as the type
  - Tick **Create from existing sources**
  - **Next** to select the .owl file

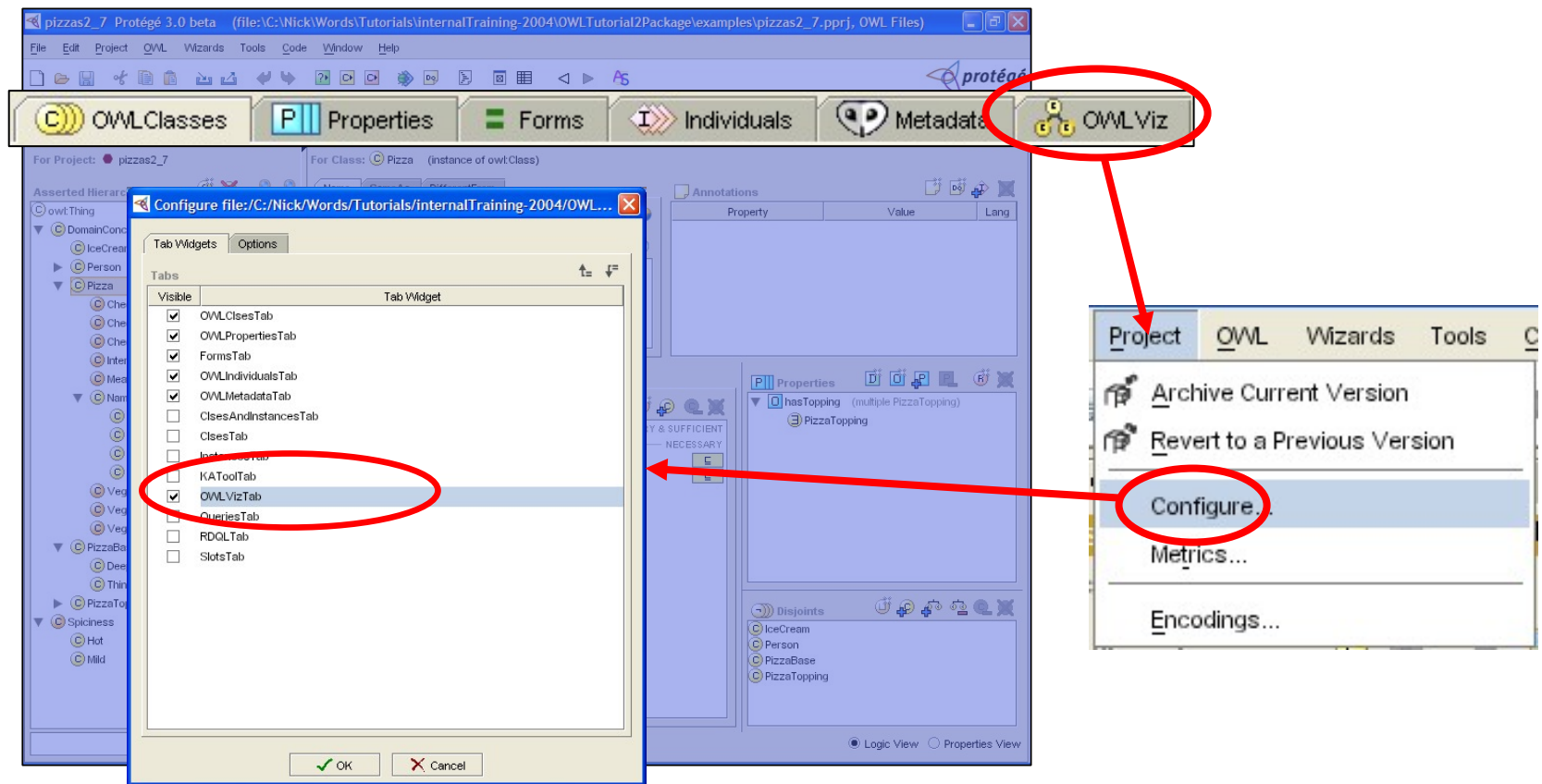


2. *If you've got a valid project file\*:*
  - **File** → **Open Project**
  - select the .pprj file

\* ie one created on this version of Protégé - the s/w gets updated once every few days, so don't count on it unless you've created it recently— safest to build from the .owl file if in doubt

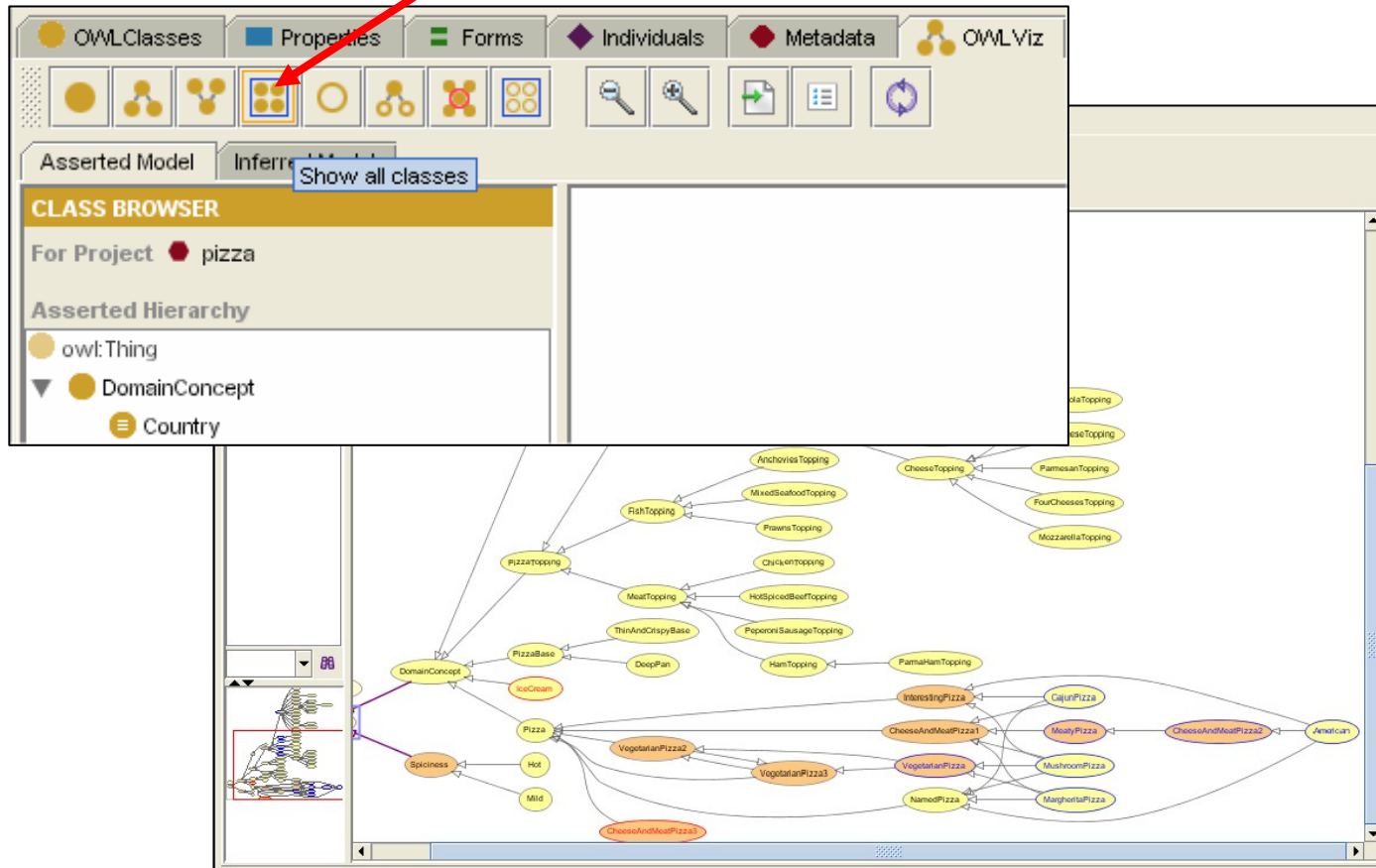


# Viewing the Hierarchy Graphically (OWL Viz)



# OWLViz Tab

Show All Classes



# Relationships In OWL

---


We can restrict how these Properties are used:

- **Globally** ... by stating things about the **Property itself**
- **Locally** ... by restricting their use for a given **Class**

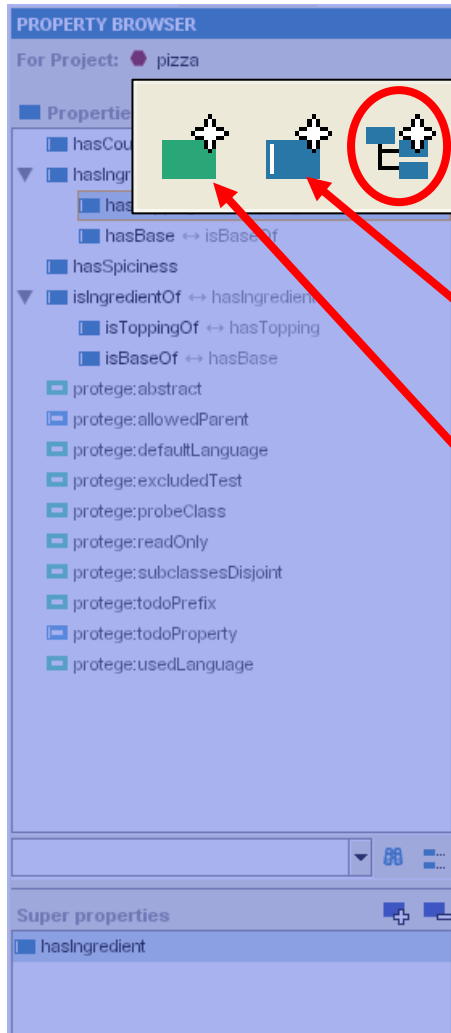
## OWL Properties

- **Object Property** ... relates individuals
- **Datatype Property** ... relates individuals to data
- **Annotation Property** ... attaching “metadata” to items

The screenshot shows the Protege OWL editor interface. At the top, the 'Properties' tab is selected for a project named 'pizza'. A list of properties is displayed, including 'hasCountryOfOrigin', 'hasIngredient', 'hasTopping', 'hasBase', 'hasSpiciness', 'isIngredientOf', 'isToppingOf', 'isBaseOf', and several 'protege:' prefixed properties. The 'hasTopping' property is highlighted with a red arrow. Below the list, the 'Super properties' section shows 'hasIngredient'.



# Creating Properties



Delete Property

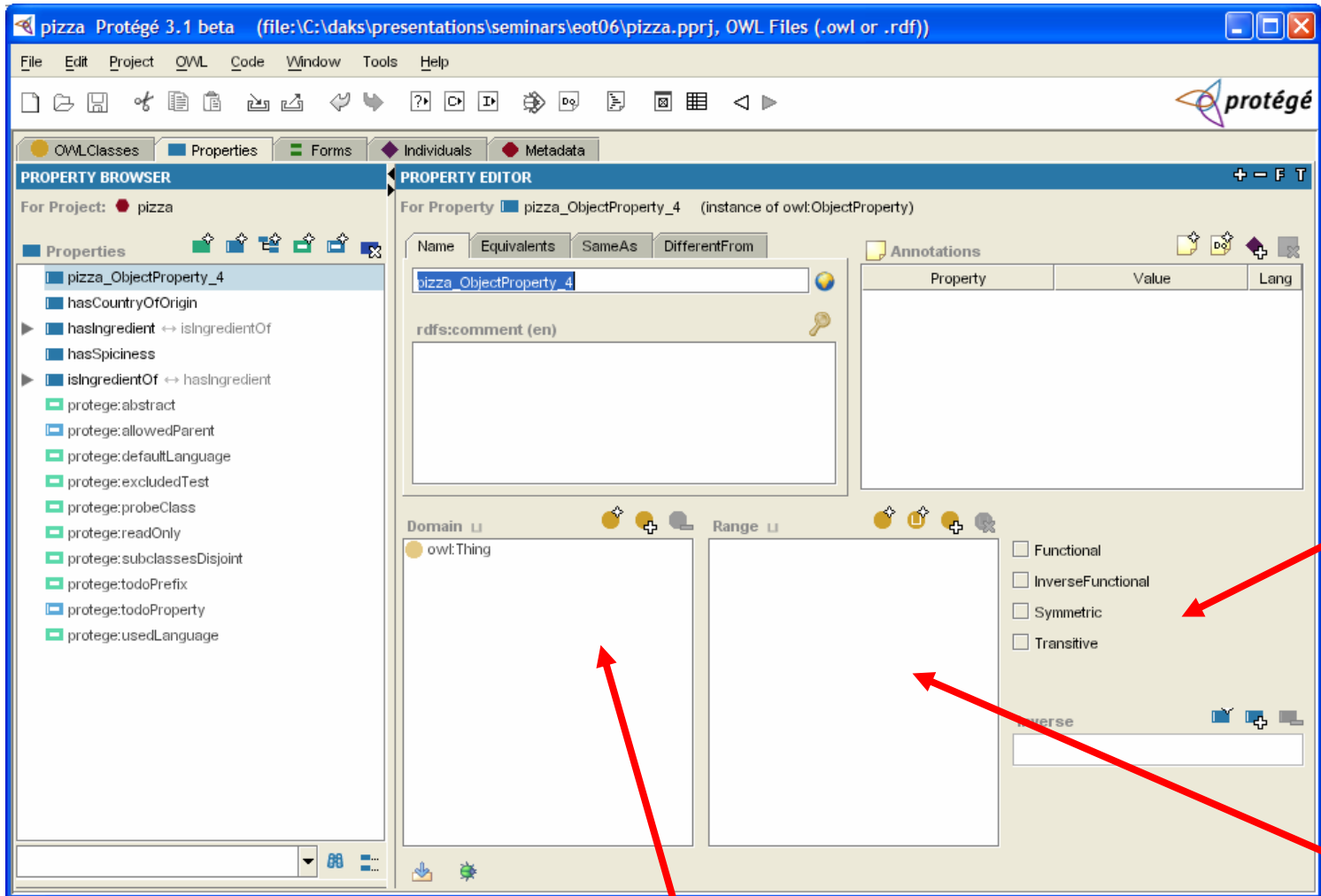
New Object Property:

Associates an individual to another individual

not used today:

- New Datatype Property (String, int etc)
- New Annotation Properties for metadata

# Globally Restricting a Property



Additional  
Constraints

Restrict Range

Restrict Domain

# Conditions Widget (Classes) – Local Restriction

Conditions asserted by the ontology engineer

Add different types of condition

Definition  
of the class  
(later)

Description  
of the class

Asserted

Inferred

Asserted Conditions

NECESSARY & SUFFICIENT

Pizza

NECESSARY

∃ hasTopping CheeseTopping

NECESSARY

∃ hasGreasyness VeryGreasy

INHERITED

∃ hasBase PizzaBase

[from PizzaBase]

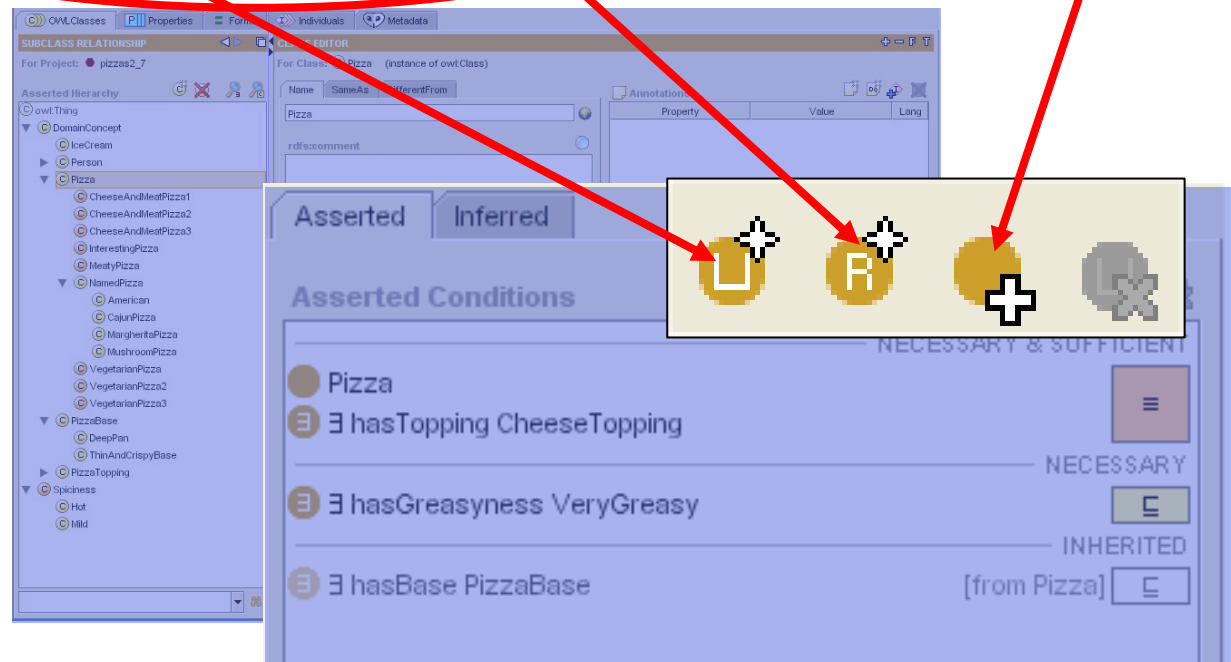
Conditions inherited from superclasses

# Condition Types

Create Class Expression

Create Restriction (next)

Add Named Superclass





# Creating Restrictions (Class restrictions)

Restricted Property

Filler  
Expression

Expression  
Construct  
Palette

Restriction  
Type

Cardinality  
Constraints

Syntax  
check

The screenshot shows the 'Create OWLRestriction' dialog box. It has a title bar with a close button. The main area is divided into several sections:

- Restricted Property:** A list box containing properties like 'hasBase', 'hasCountryOfOrigin', 'hasGreasiness', 'hasIngredient', 'hasSpiciness', 'hasTopping', 'isBaseOf', and 'isIngredientOf'. A red arrow points to this list.
- Filler Expression:** A text field containing 'PizzaBase'. A red arrow points to this field.
- OWLRestriction:** A list box containing restriction types: 'allValuesFrom', 'someValuesFrom', 'hasValue', 'cardinality', 'minCardinality', and 'maxCardinality'. A red arrow points to this list.
- Expression Construct Palette:** A horizontal bar with various icons for building expressions, including logical operators and quantifiers. A red arrow points to this palette.
- Syntax check:** A small icon of a person with a question mark, used for checking the syntax of the restriction. A red arrow points to this icon.

At the bottom of the dialog are 'OK' and 'Cancel' buttons.

# Protégé Exercise:

---

- With a partner, take 10 minutes to:
  - Explore the pizza ontology ...
    - Find 2 of your favorite pizzas
    - Find 2 “strange” pizzas or toppings
  - Add a new topping and named pizza that uses your topping

# Overview

---

Introduction: Problems in discovery and integration (~10min)

- *Exercise: Data integration (~35min)*

## Ontologies

- Basics (~25min)
- Ontology Engineering (~5min)
- Representation Languages (~30min)
- *Exercise: Ontology development (~75min)*

## ➤ The Protégé Editor

- Overview (~15min)
- *Exercise: Protégé (~30min)*

Using Ontologies (~25min)

# Exercise: Using the Protégé Editor

---

- With the same groups you built your ontology with:
  - Enter the concept hierarchy
  - Enter a few properties
  - Enter some cardinality constraints for the concepts
    - Note that you may need to make some modifications of your ontology ...

# Overview

---

Introduction: Problems in discovery and integration (~10min)

- *Exercise: Data integration (~35min)*

## Ontologies

- Basics (~25min)
- Ontology Engineering (~5min)
- Representation Languages (~30min)
- *Exercise: Ontology development (~75min)*

## The Protégé Editor

- Overview (~15min)
- *Exercise: Protégé (~30min)*

➤ **Using Ontologies (~25min)**

# Basic uses of ontologies

## Ontologies for metadata keywords

{cabernet sauvignon, sonoma region, ...}

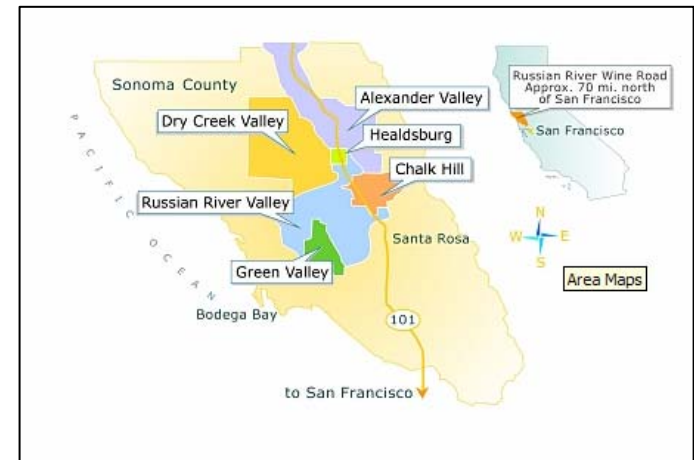


{medium, red, dry, ...}

The screenshot shows a web browser displaying a table of wine quantities and prices. The table has columns for Year, Quantity, Total Wt, and Price. The data is as follows:

Year	Quantity	Total Wt	Price
1997	831	3	53
1998	864	0	62
1999	851	-	-
2000	1,015	-	-
2001	972	-	-
2002	1,338	-	-
2003	1,015	-	-
2004	851	-	-
2005	1,323	-	-
2006	991	0	46
2007	894	2	77
2008	1,000	5	76
2009	540	0	40
2010	799	13	80
2011	1,304	2	44
2012	1,596	1	79
2013	716	2	80
2014	478	3	64
2015	833	4	43
2016	790	4	74
2017	1,036	10	76

{sonoma region, ...}



# Basic uses of ontologies

## Ontologies for metadata keywords

Find information about  
*dry california red wine*

{cabernet sauvignon, sonoma region, ...}

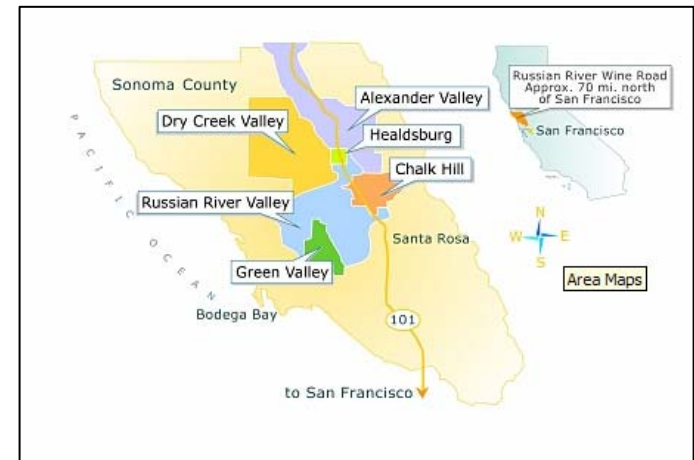


{medium, red, dry, ...}

Table with 5 columns: Year, Quantity, Total W. (%), (QA)(%), and Qd(P%). The table lists data for various years from 1997 to 1999.

Year	Quantity	Total W. (%)	(QA)(%)	Qd(P%)
1997	831	3	53	44
1998	864	0	62	38
1999	851	-	-	-
2000	1,015	-	-	-
2001	972	-	-	-
2002	1,338	-	-	-
2003	1,015	-	-	-
2004	851	-	-	-
2005	1,323	-	-	-
2006	991	0	46	54
2007	894	2	77	21
2008	1,000	5	76	19
2009	540	0	40	60
2010	799	13	80	7
2011	1,304	2	44	54
2012	1,590	1	79	20
2013	716	2	60	38
2014	476	3	64	33
2015	831	4	43	54
2016	710	4	74	22
2017	1,036	10	76	14

{sonoma region, ...}



Plain text search finds only the middle page ...

# Basic uses of ontologies

## Ontologies for metadata keywords

Find information about  
*dry california red wine*

{cabernet sauvignon, sonoma region, ...}

{sonoma region, ...}

{medium, red, dry, ...}

A screenshot of a Microsoft Internet Explorer window displaying a table of wine data. The table has columns for Year, Quantity, Total W, and Quality. The data is organized into a table with multiple rows and columns.

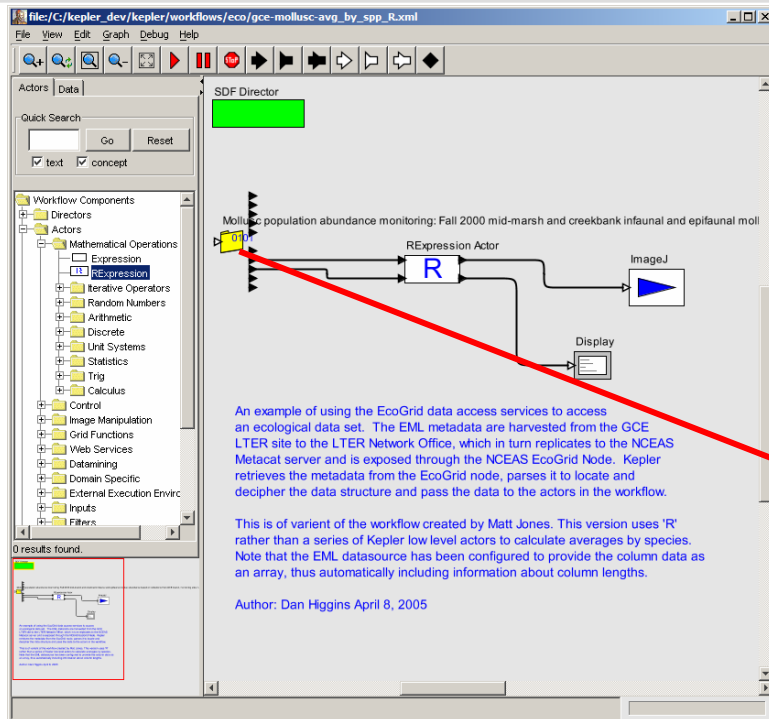
Year	Quantity	Total W	Quality
1997	831	3	53
1996	864	0	62
1995	851	-	-
1994	1,015	-	-
1993	972	-	-
1992	1,338	-	-
1991	1,015	-	-
1990	851	-	-
1989	1,323	-	-
1988	991	0	46
1987	894	2	77
1986	1,000	5	76
1985	540	0	40
1984	799	13	80
1983	1,304	2	44
1982	1,590	1	79
1981	716	2	80
1980	476	3	64
1979	831	4	43
1978	710	4	74
1977	1,036	10	76



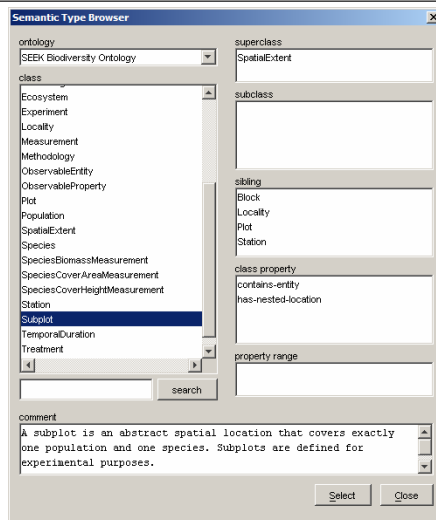
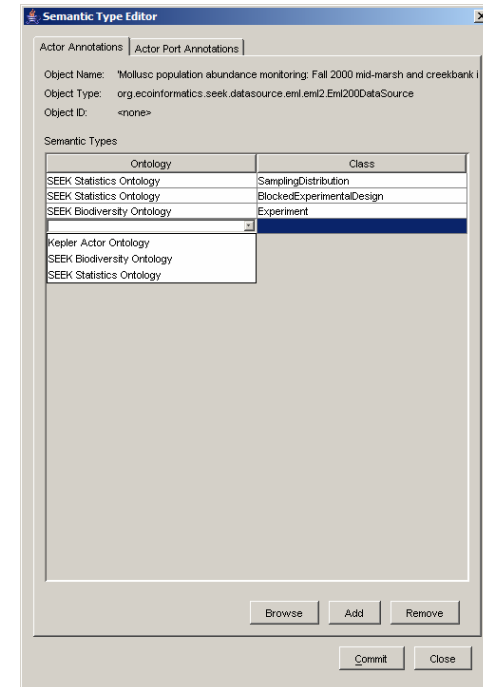
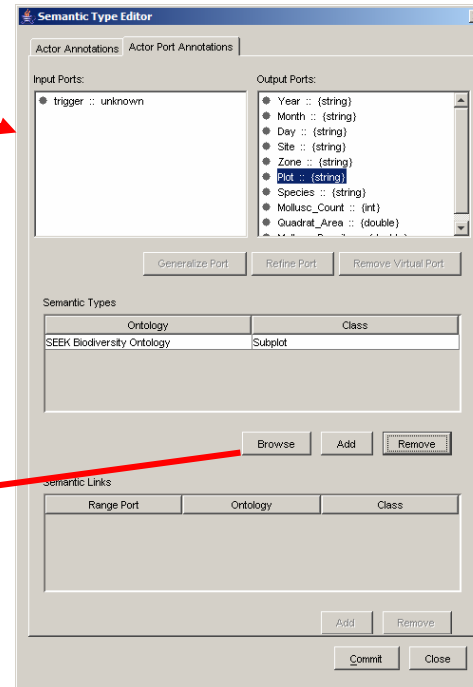
We use the ontology to “expand” the query, e.g., that cabernet sauvignon is red and dry; sonoma is a region in california



# Kepler: Typing Workflow Components



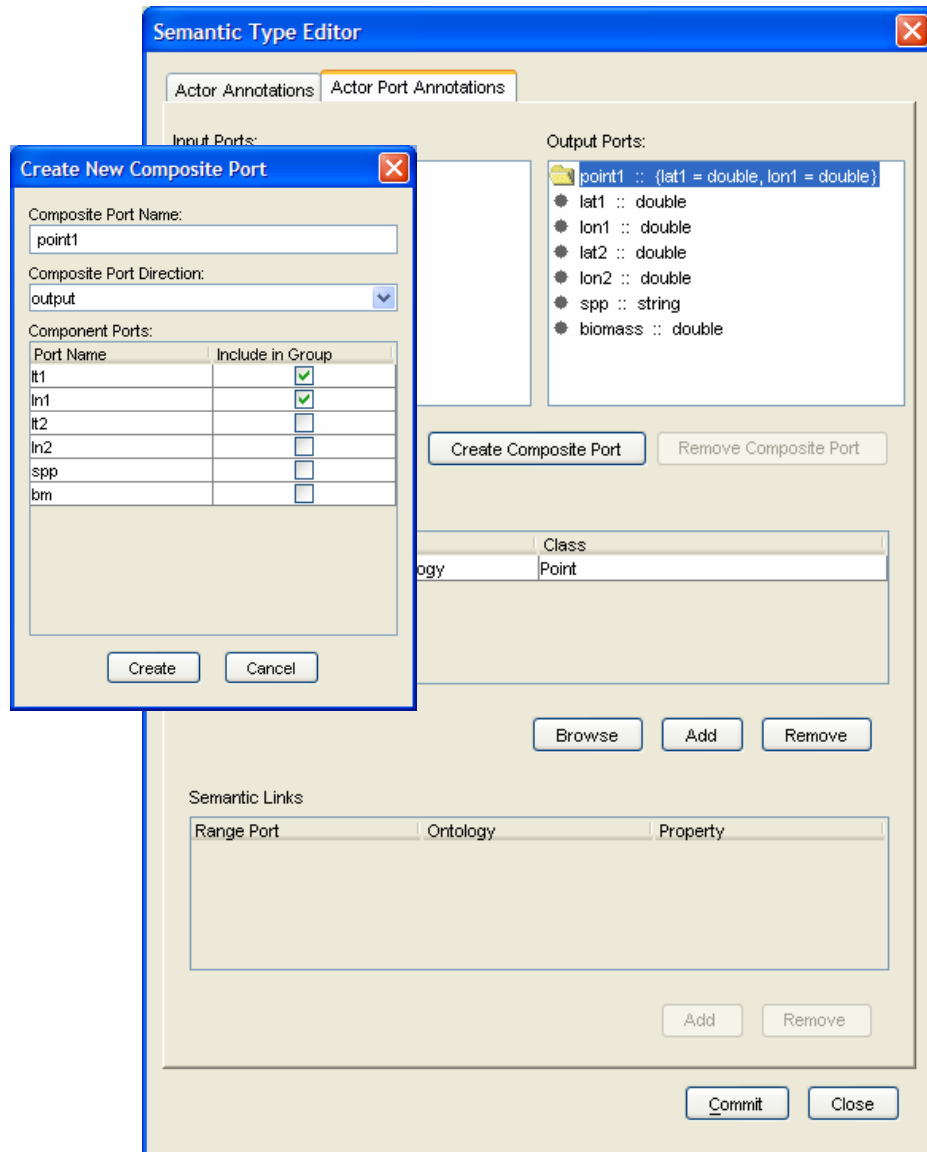
**Semantic Type Editor** is used to assign one or more semantic types to the component or to the component's input and output ports. In the simplest case, a semantic type is a class taken from an OWL-DL ontology. Multiple types define a conjoined concept expression.



A simple **ontology browser** is provided in Kepler to navigate a classified OWL-DL ontology. Classes can be searched for and selected as a semantic type.

# Kepler: More on Semantic Annotation

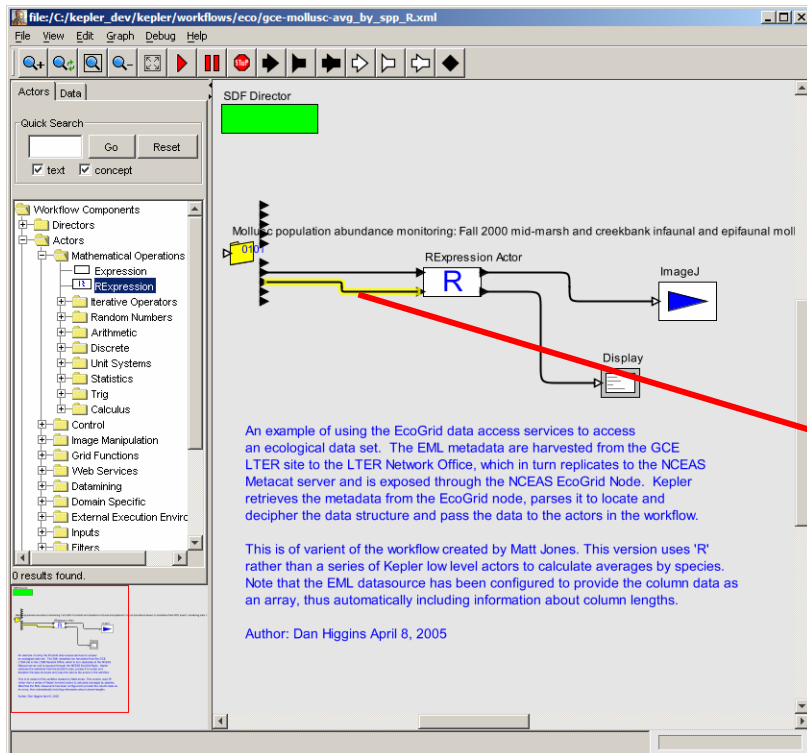
- **Actor-level** and **port-level** annotations
- Annotations are stored in actor's **MoML** definition (as new “semantic type” properties)
- Creation of **composite ports** (i.e., “virtual” ports grouping a set of underlying ports)
- Regular and composite ports may have **multiple annotations** (conjunction)
- Annotations can be drawn from **multiple ontologies**
- More stuff in the works ...



**An annotated composite port**

# Kepler: Checking Type Constraints

Kepler can **statically perform semantic and structural type checking** of connections. A type checker allows the user to see potentially mismatched port connections as well as known type conflicts before workflow execution.



### Structural and Semantic Type Checker

**Channels**

Unsafe Channels:

Output Port	Input Port
'Mollusc population abundance ...'	'RExpression Actor'.cnt

Potentially Unsafe Channels:

Output Port	Input Port
'RExpression Actor'.output	Display.input
'RExpression Actor'.graphicsFil...	ImageJ.input
'Mollusc population abundance ...'	'RExpression Actor'.spp

**Structural Types**

channel status: safe (green)

output: {int}

input: {int}

**Semantic Types**

channel status: error (red)

Ontology	Class
SEEK Biodiversity Ontology	Population

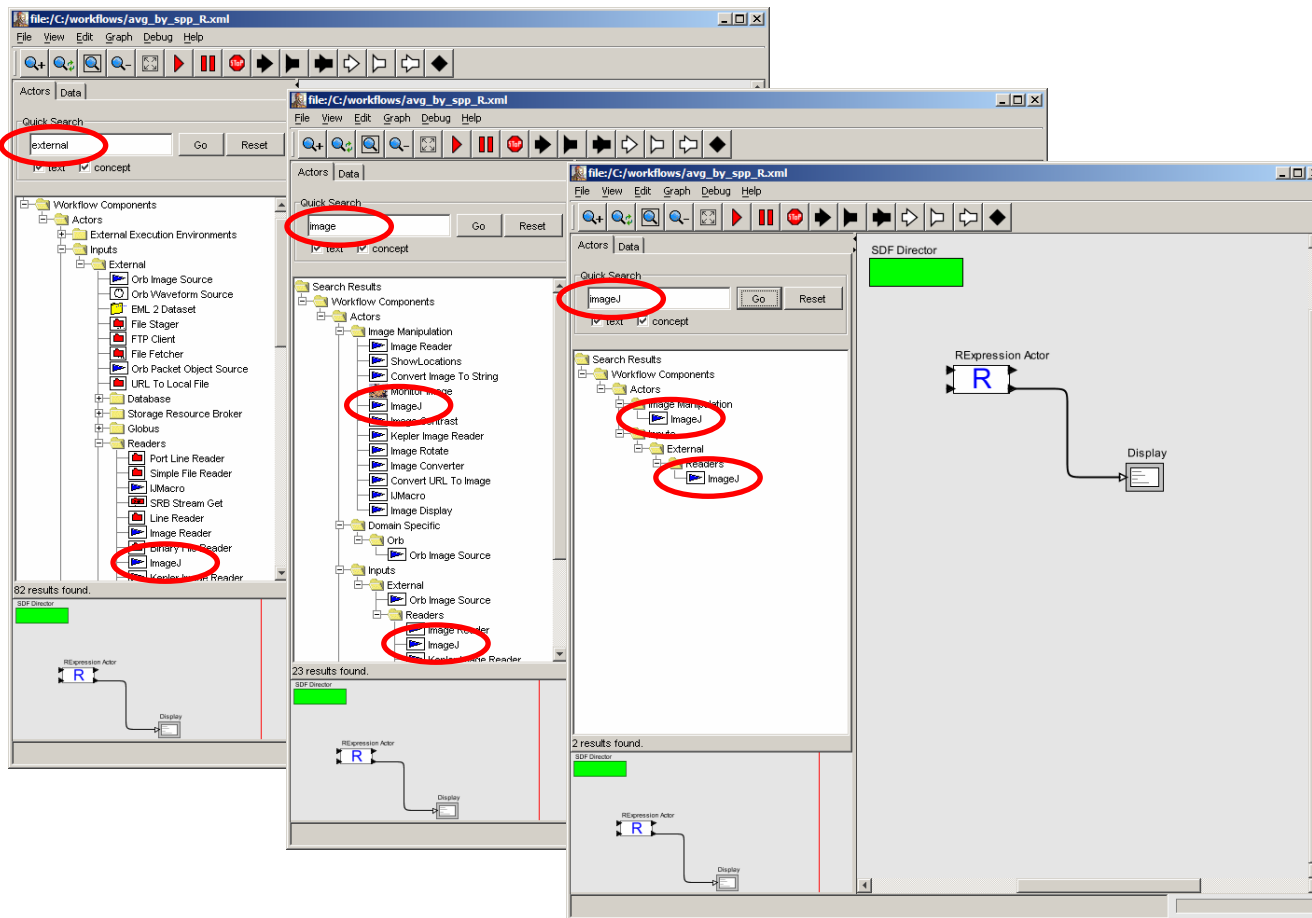
Ontology	Class
SEEK Biodiversity Ontology	CoverArea

Insert Adapters Close

The user can **navigate** the **unsafe and potentially unsafe** channels using the Kepler Type Checker dialog. When a channel is selected: (a) it is highlighted on the canvas, (b) the structural type and status is shown (here, the channel is structurally well typed), and (c) the semantic type and status is shown (here, the connection produce a semantic type error).

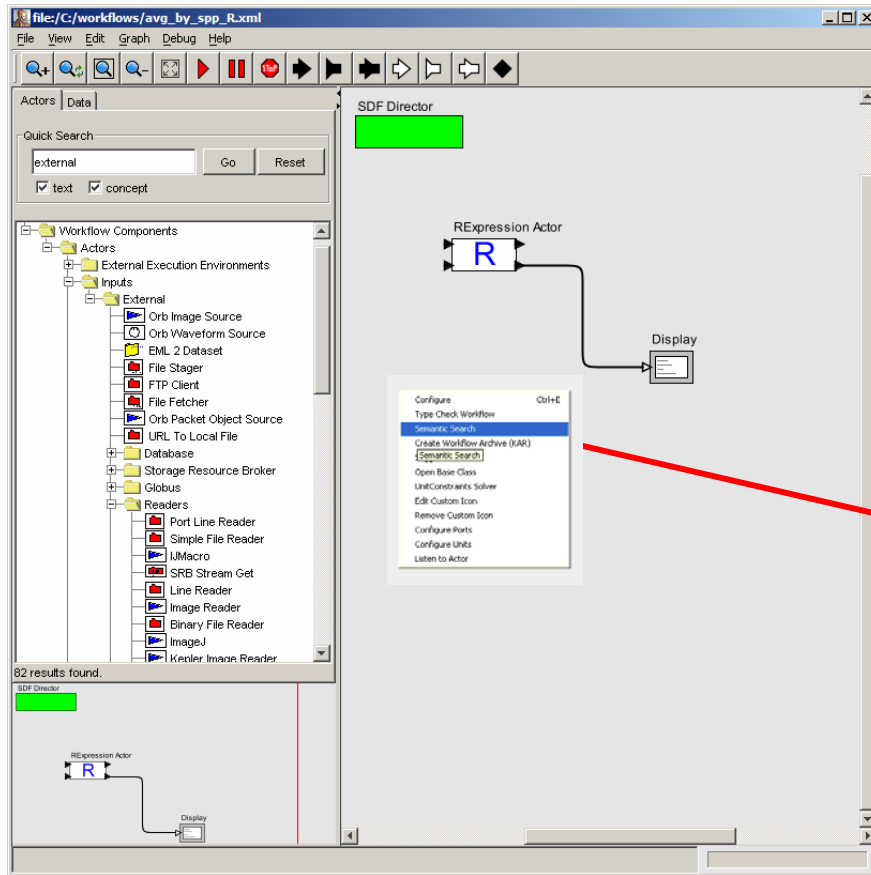
# Kepler: Searching for Actors

- Ontology-based actor organization / browsing
- Customizable libraries based on ontologies
- Text search with concept-based expansion

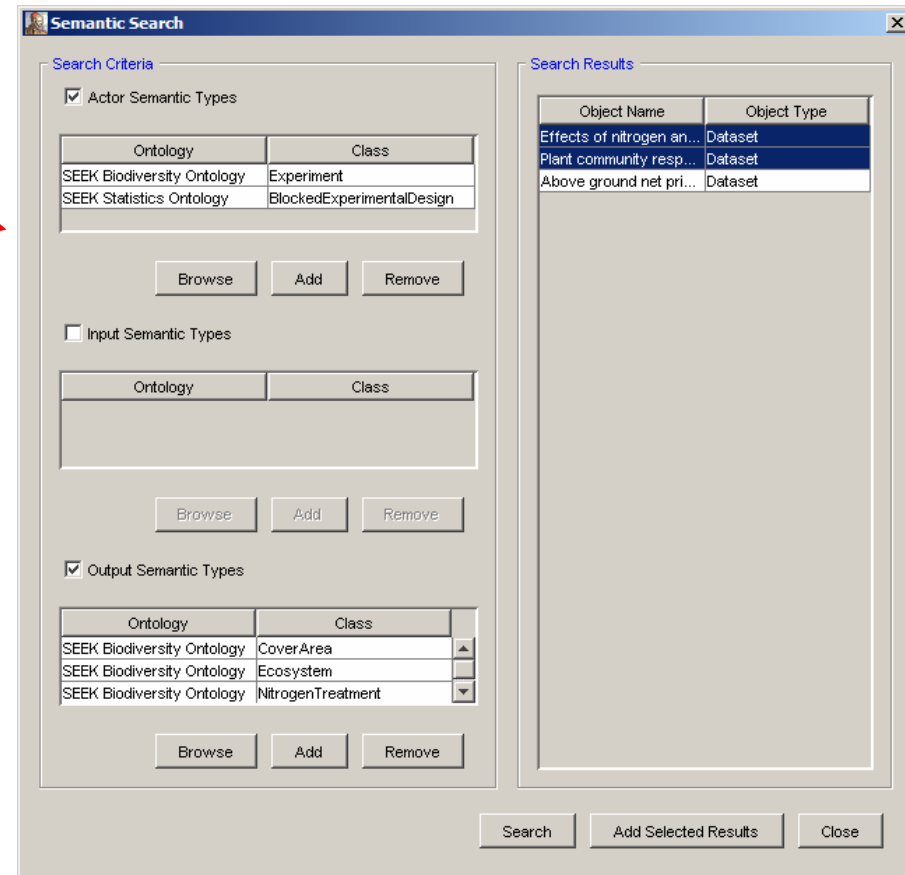


Users can discover ImageJ using various search terms. Here, ImageJ shows up in multiple tree locations based on its given annotations. The library search permits text-based matching against the component's metadata (its given name and certain properties), expanded with concept matches.

# Kepler: More advanced ontology searching

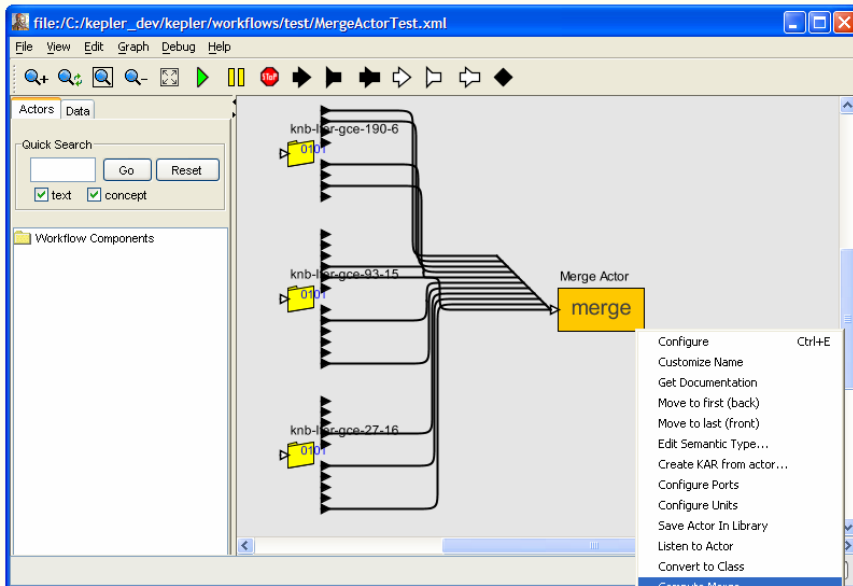


Kepler provides a more advanced ontology-based search mechanism. Users can start the **Semantic Search** dialog, where components can be searched based on their semantic types.

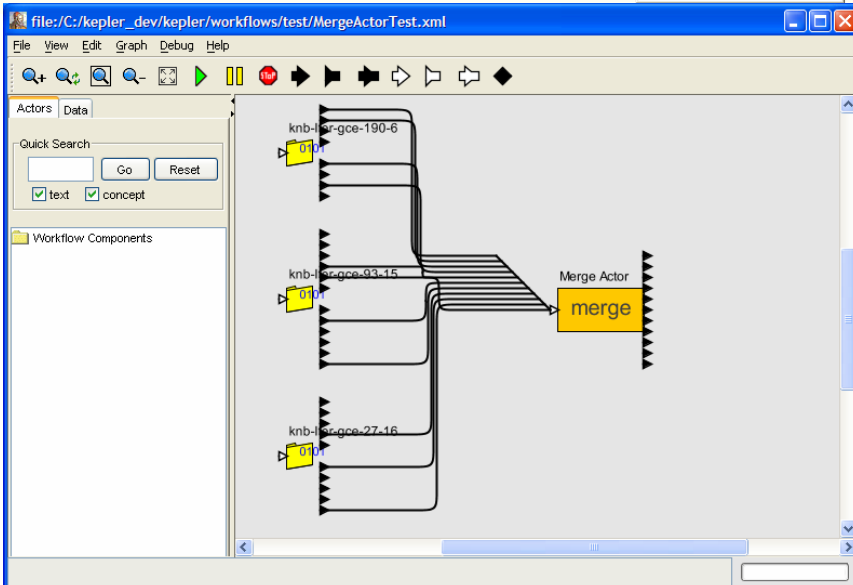


The Semantic Search dialog allows a user to search components by any combination of actor, input, and output semantic types.

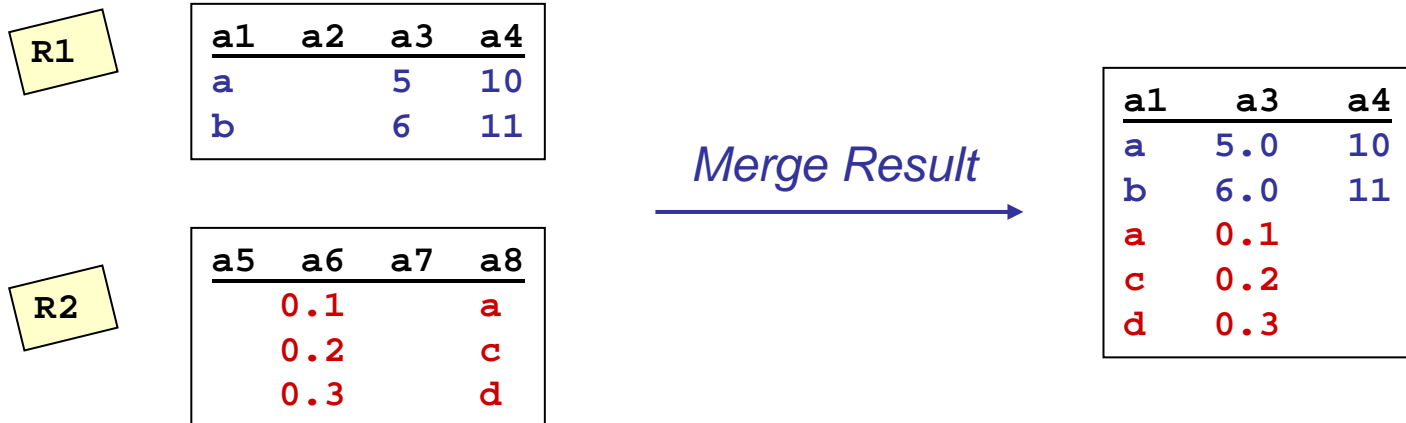
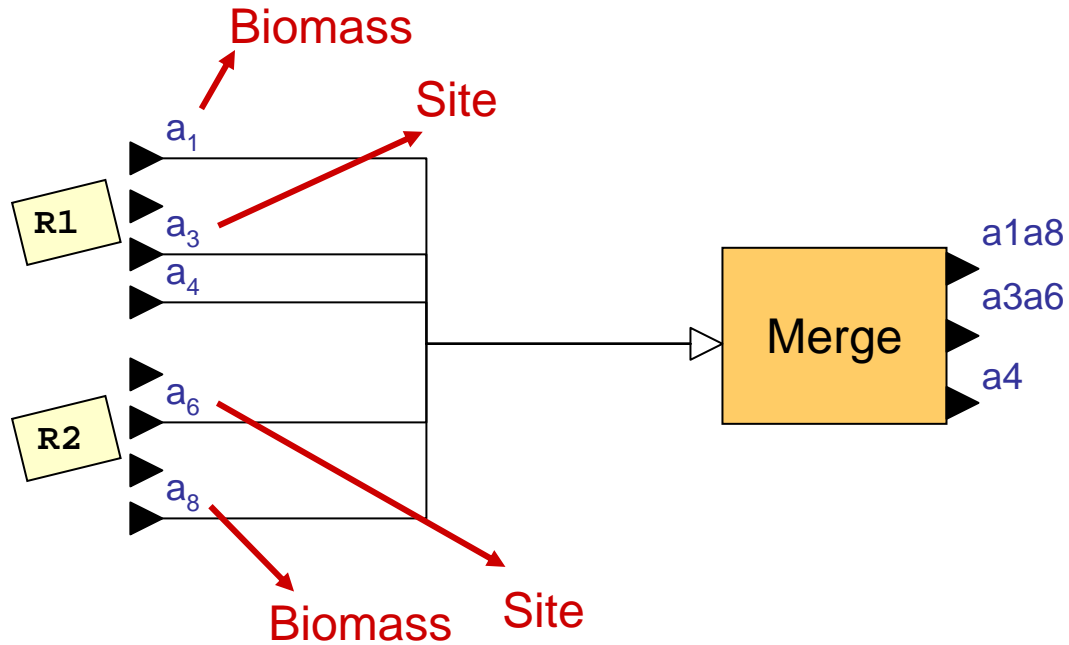
# Kepler: Automating On-the-fly Merge



- Select datasets (or actors) of interest
- Connect desired merge attributes to the merge actor
- Select “compute merge”:
  - Aligns attributes via annotations
  - Dialog for user refinement (not shown)
  - Stores merge “mapping” as configured actor (via new MoML properties) ... and is then executable

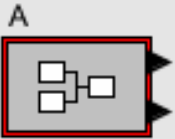


# Simple example of Merge

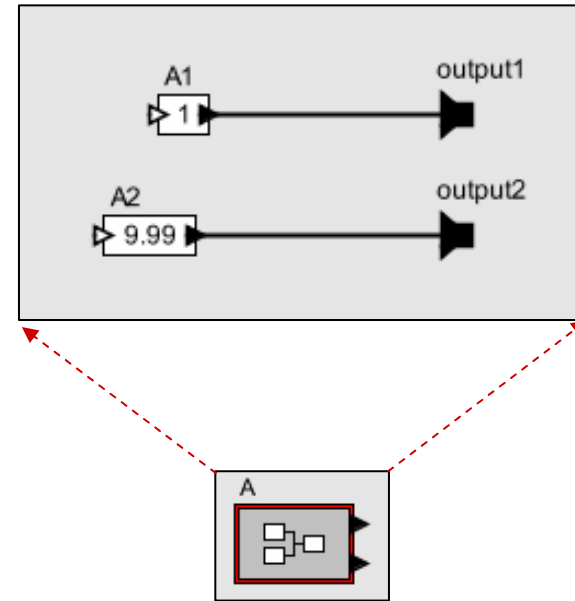


# Kepler: Using Merge

SDF Director



Merge Actor



Here are four “data sources” we want to merge

- These are just constant actors to keep things simple
- One is a composite
- Any actor can be used, **including EML datasets**

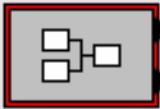


# More on the Merge Actor

SDF Director



A



B



C



Merge Actor

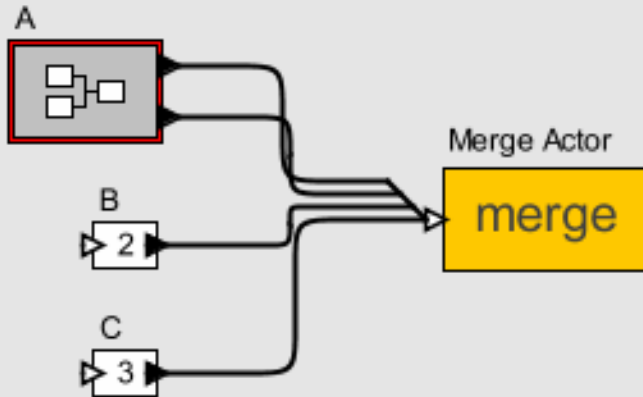
merge

We first **connect** the actor output ports we want to merge ...

... to the input port of the merge actor

# More on the Merge Actor

SDF Director

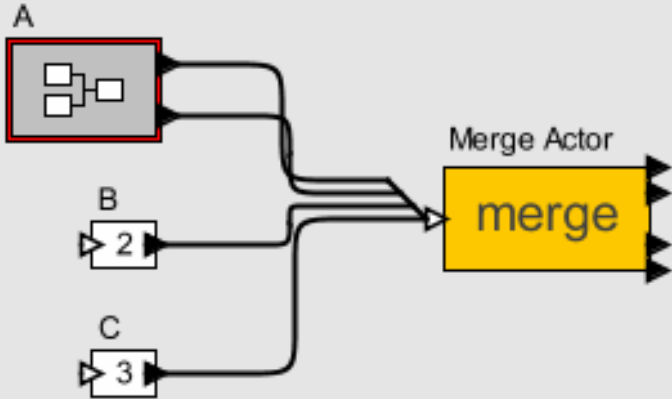


We then select the **compute merge** menu item for the merge actor

- Configure Ctrl+E
- Customize Name
- Get Documentation
- Move to first (back)
- Move to last (front)
- Edit Semantic Type...
- Create KAR from actor...
- Configure Ports
- Configure Units
- Save Actor In Library
- Listen to Actor
- Set Breakpoints
- Convert to Class
- Compute Merge**
- Edit Merge Mappings
- Look Inside Compute Merge Ctrl+E
- Edit Custom Icon
- Remove Custom Icon

# More on the Merge Actor

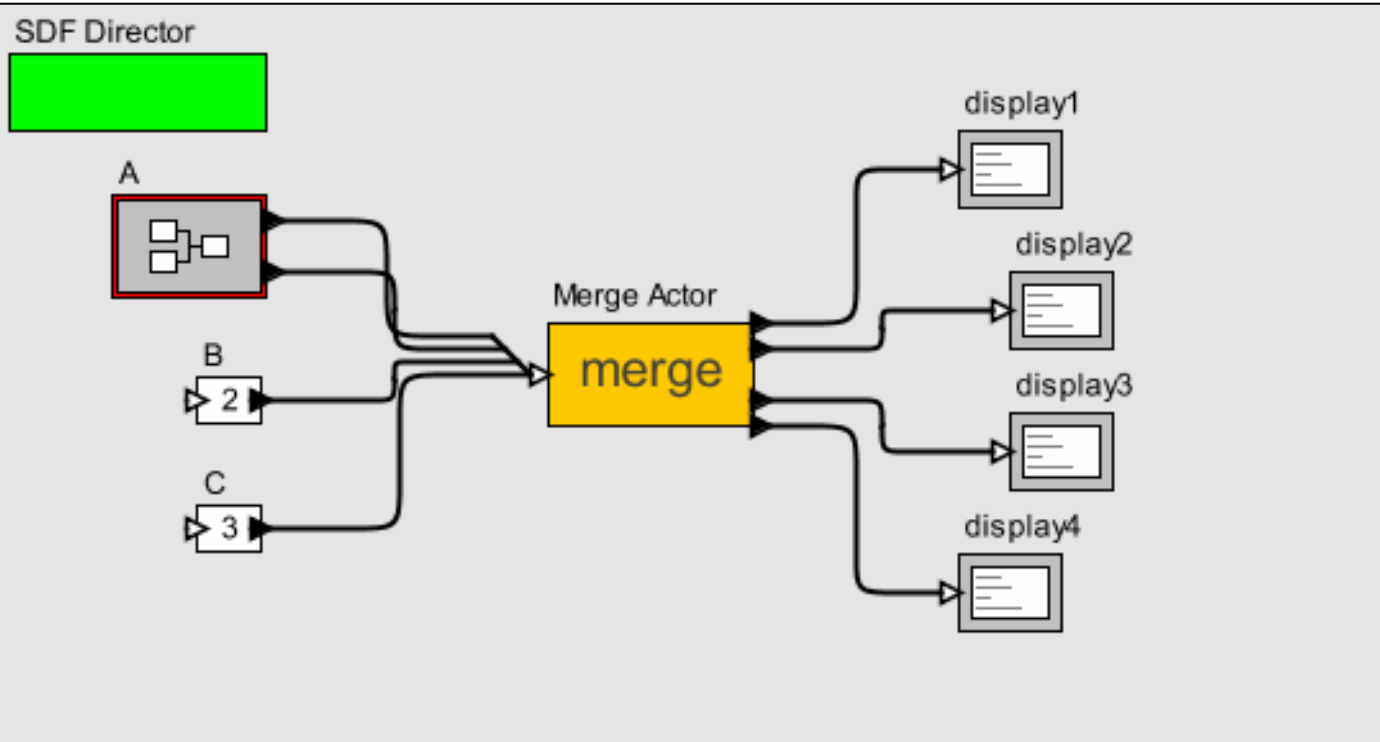
SDF Director



The merge actor creates a set of **target output ports**

In this case, because no semantic types are given, a target output port is created for each actor port

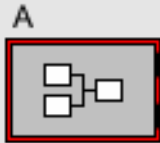
# More on the Merge Actor



We connect displays for each of the target output ports

# More on the Merge Actor

SDF Director



Merge Actor

merge

display1

display2

display3

display4

And execute it ...

Each row is one  
output of merge;  
Zero values  
denote null

File	Help
1	
0	
0	

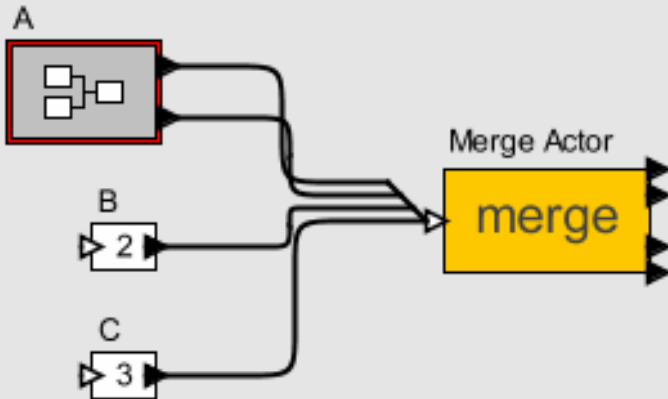
File	Help
9.99	
0.0	
0.0	

File	Help
0	
2	
0	

File	Help
0	
0	
3	

# More on the Merge Actor

SDF Director

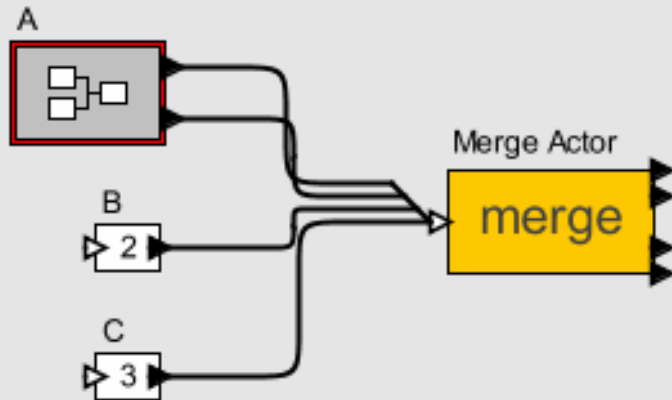


After computing a merge, we can also **edit** the resulting mappings from actor output ports to target ports ...

- Configure Ctrl+E
- Customize Name
- Get Documentation
- Move to first (back)
- Move to last (front)
- Edit Semantic Type...
- Create KAR from actor...
- Configure Ports
- Configure Units
- Save Actor In Library
- Listen to Actor
- Set Breakpoints
- Convert to Class
- Compute Merge
- Edit Merge Mappings**
- Look Inside Ctrl+L
- Edit Custom Icon
- Remove Custom Icon

# More on the Merge Actor

SDF Director



The editor allows mappings to be:

1. Navigated
2. Semantically Annotated (targets)
3. Refined

Compute Merge Results

Input-Output Mapping

merge input ports

- Folder A
  - output1
  - output2
- Folder B
  - output
- Folder C
  - output

merge output ports

- target1
- target2
- target3
- target4

Refine Mappings

Prune Output Ports

Add Output Port

Conversion Functions

Function

Output Port

Output Port Semantic Type

Ontology

Class

Browse

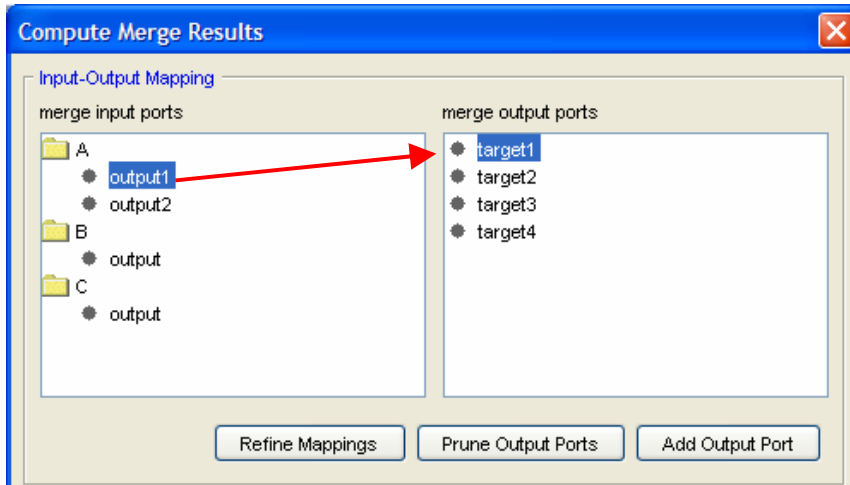
Add

Remove

Commit

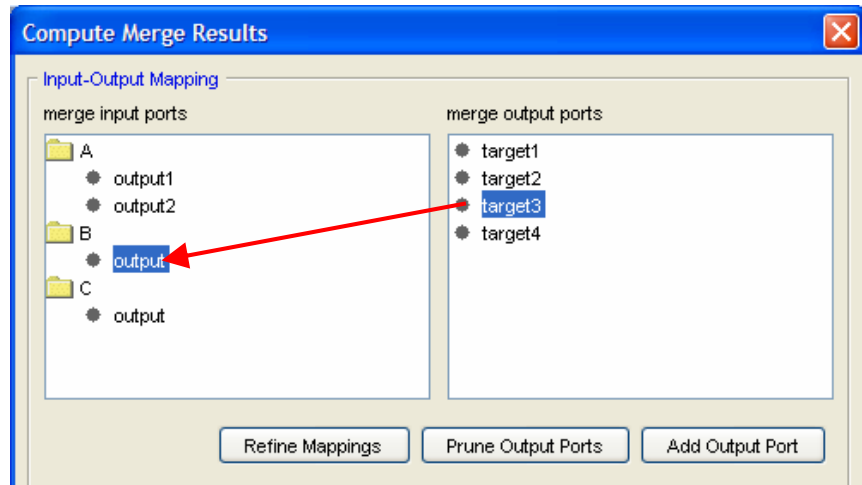
Cancel

# More on the Merge Actor



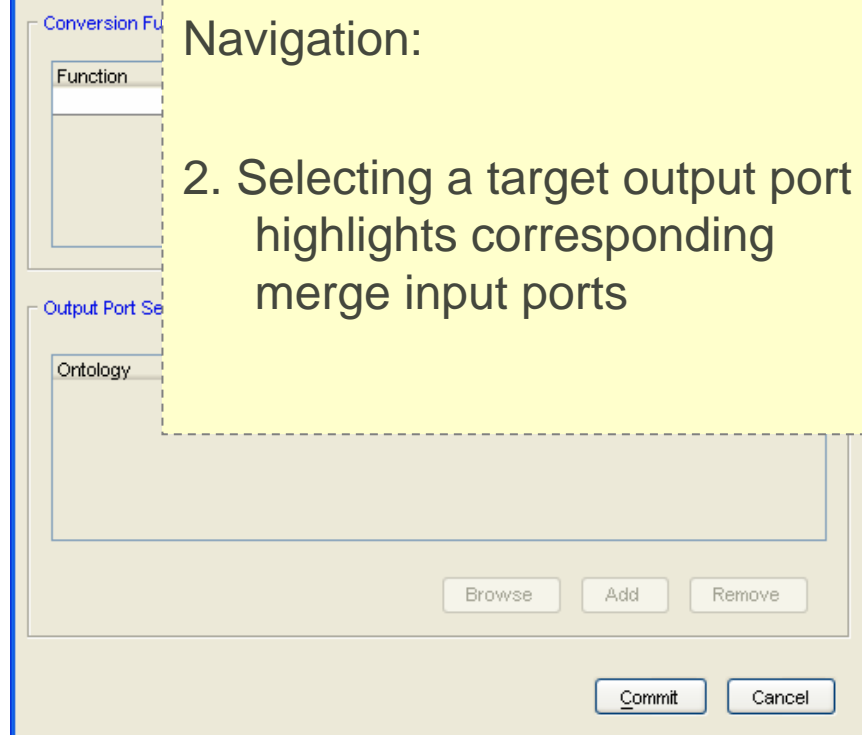
## Navigation:

1. Selecting a merge input port (actor output port) highlights corresponding target ports



## Navigation:

2. Selecting a target output port highlights corresponding merge input ports





# More on the Merge Actor

**Refine Input-Output Mappings**

Source Actor  
A

Source Actor Port  
output1

Output Port	Target
target1	<input checked="" type="checkbox"/>
target2	<input type="checkbox"/>
target3	<input checked="" type="checkbox"/>
target4	<input checked="" type="checkbox"/>

OK Cancel



**Compute Merge Results**

Input-Output Mapping

merge input ports

- A
  - output1
  - output2
- B
  - output
- C
  - output

merge output ports

- target1
- target2
- target3
- target4

Refine Mappings Prune Output Ports Add Output Port

Conversion Functions

Function	Output Port
	target1

Output Port Semantic Type

Ontology	Class
----------	-------

Browse Add Remove

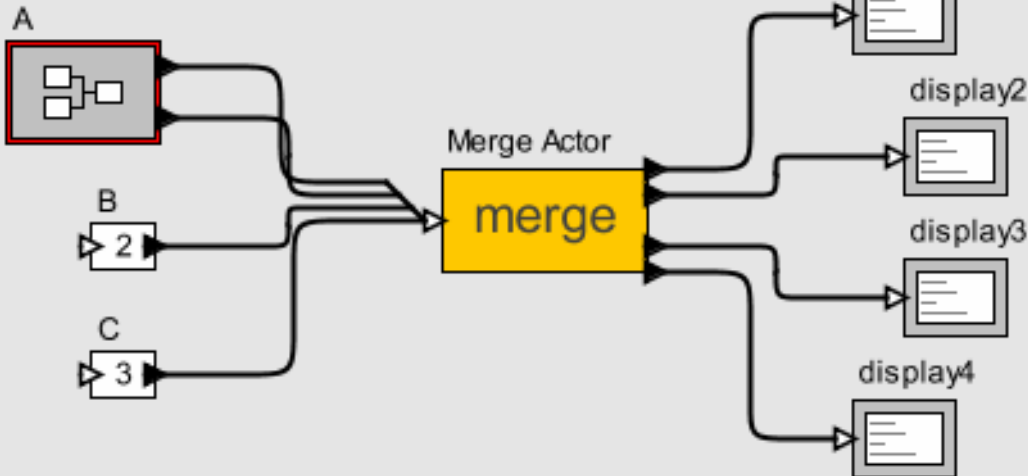
Commit Cancel

Mappings can be further refined ...

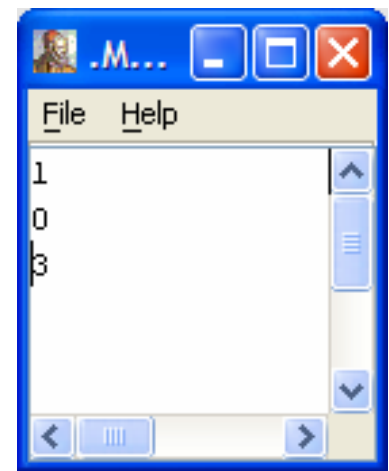
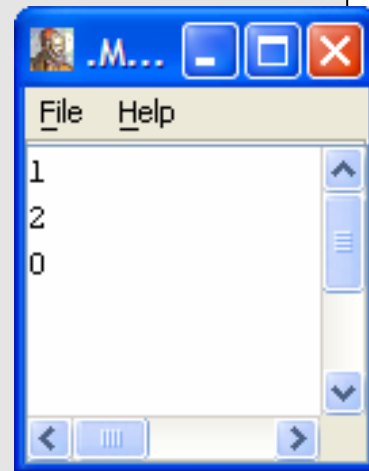
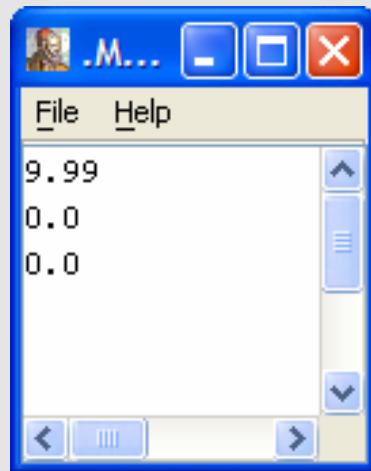
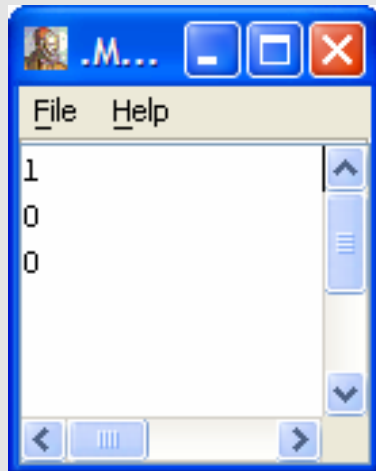
New targets can be added and “dangling” targets, i.e., targets without an associated input, “pruned” (not shown)

# More on the Merge Actor

SDF Director

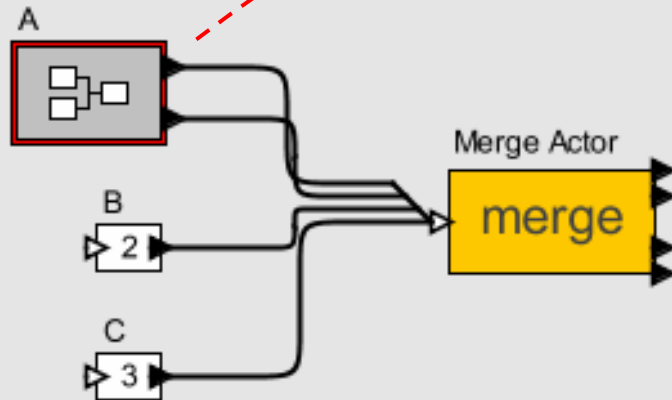


And the result can be executed ...



# More on the Merge Actor

SDF Director



If actor ports are **semantically annotated** ... merge can use them to create “better” initial mappings

Semantic Type Editor

Actor Annotations

Actor Port Annotations

Input Ports:

Output Ports:

- output1 :: int
- output2 :: double

Create Composite Port

Remove Composite Port

Semantic Types

Ontology

SEEK Field Observation Ontology

Class

Biomass

Browse

Add

Remove

Semantic Links

Range Port

Ontology

Property

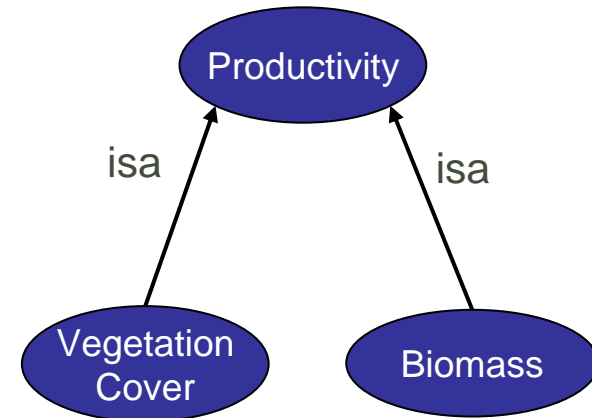
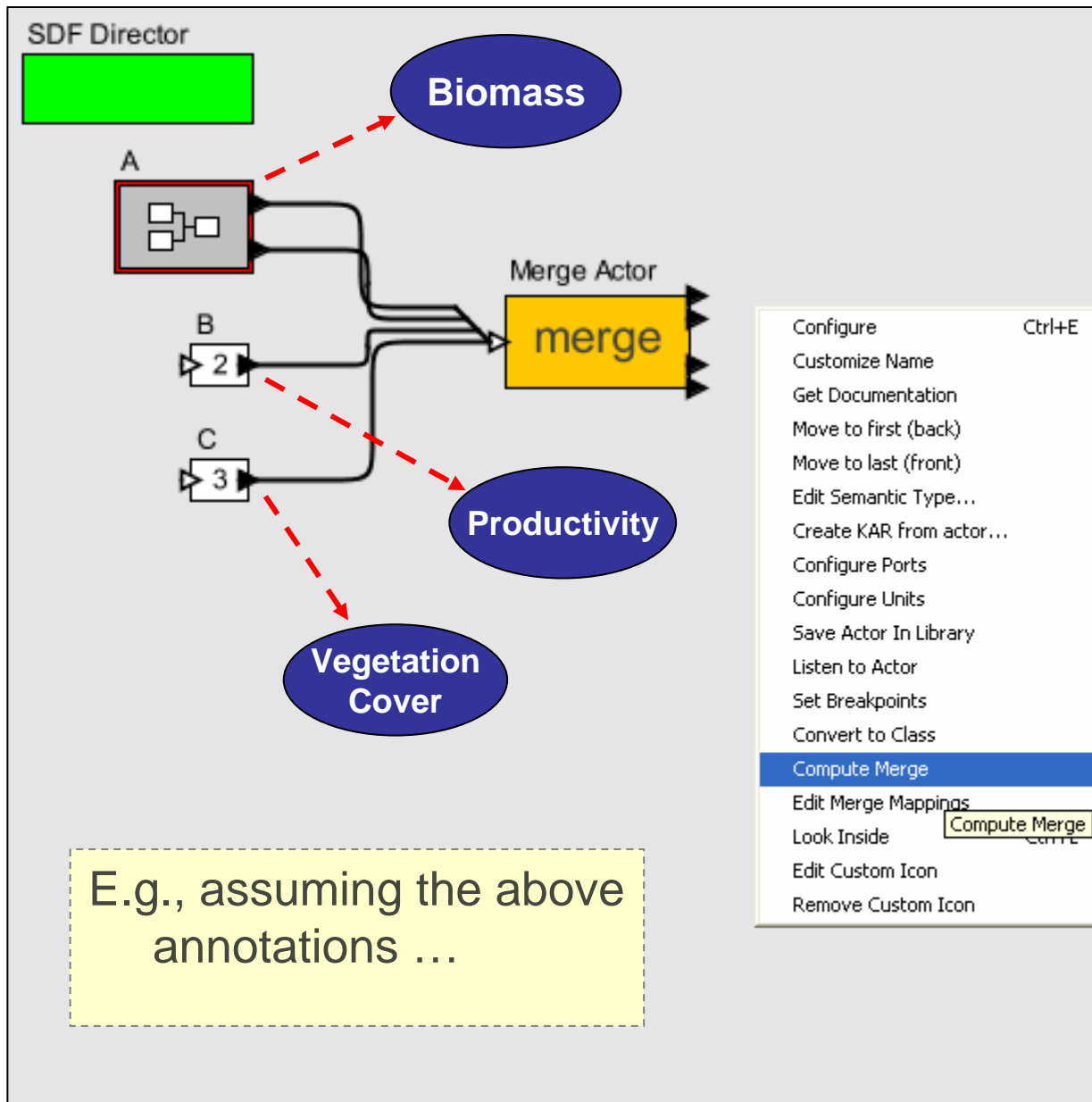
Add

Remove

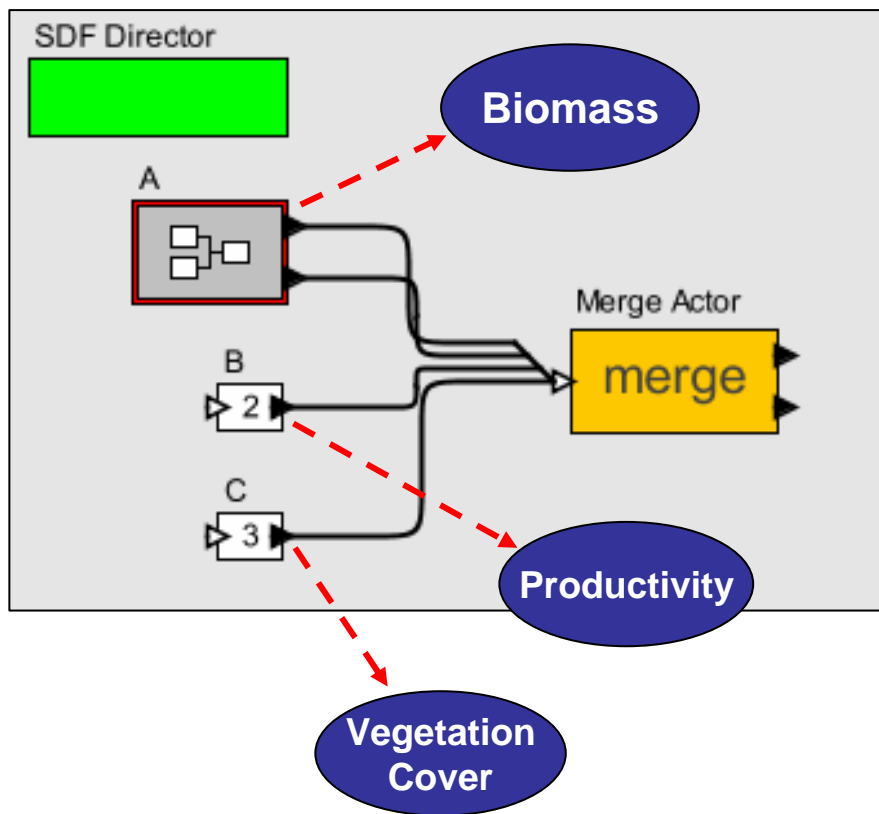
Commit

Close

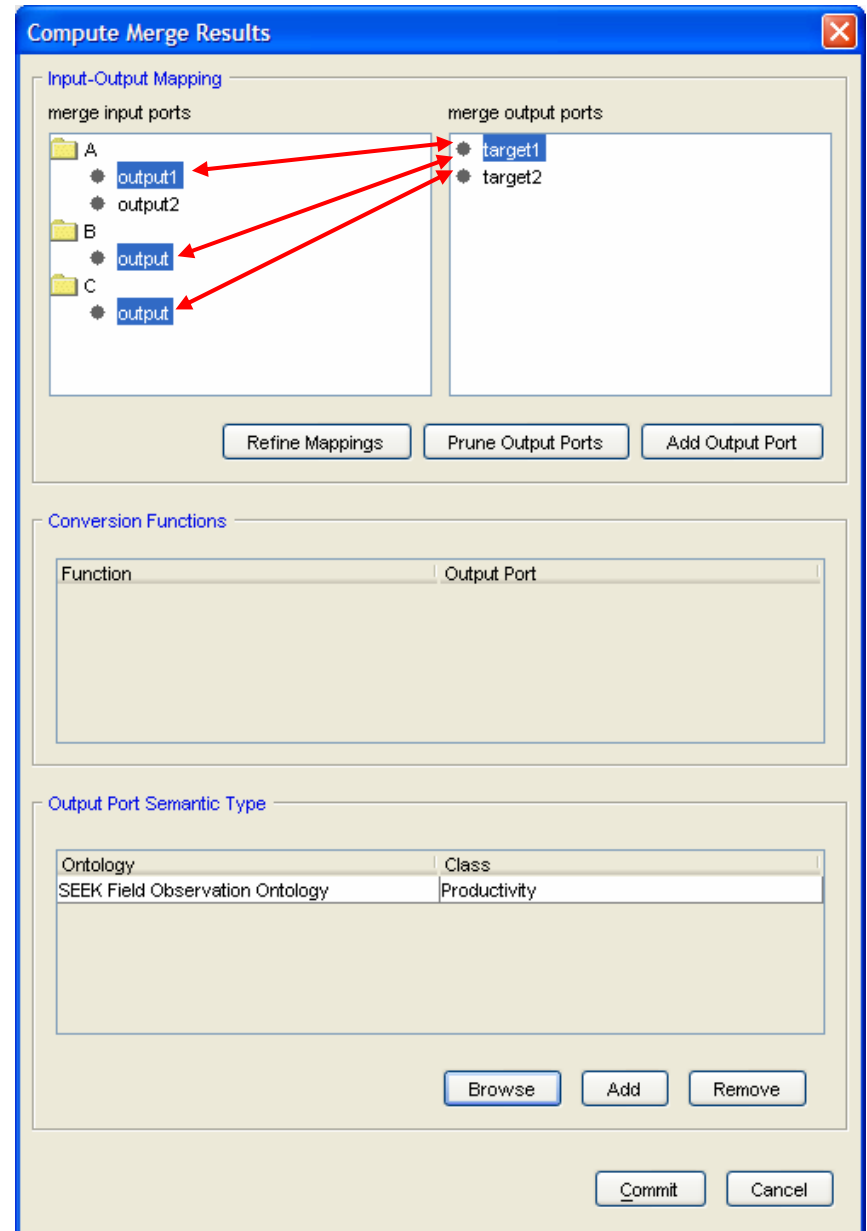
# More on the Merge Actor



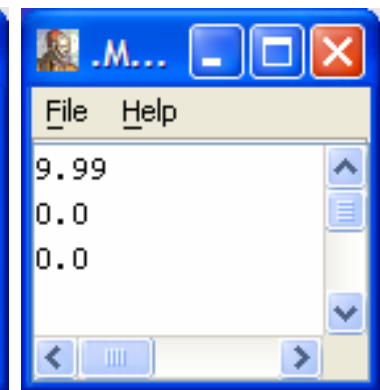
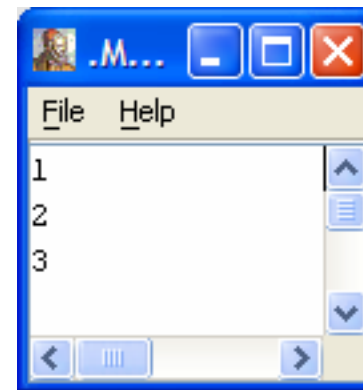
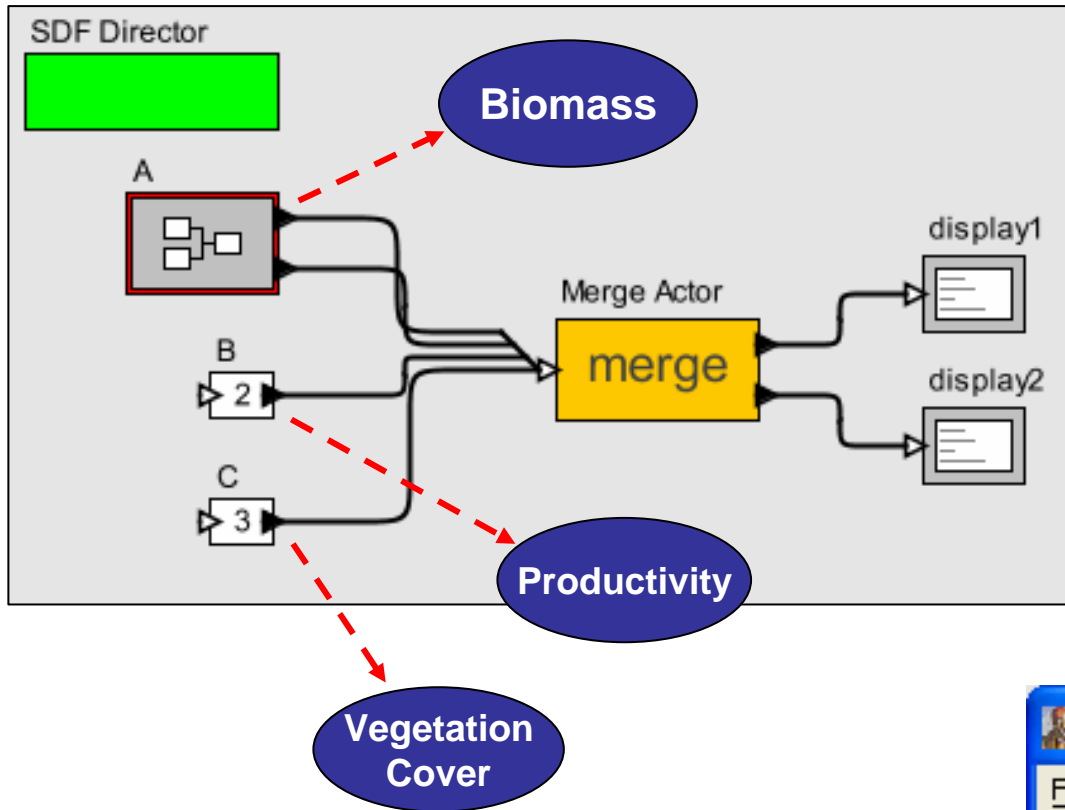
# More on the Merge Actor



merge computes the mappings ...



# More on the Merge Actor



And when executed  
produces ...

# References

---

- Practical guides/references
  - **Protégé**. Open source ontology editor. <http://protege.stanford.edu/>
  - **CO-ODE**. Various resources on ontologies, tutorials, best-practices, etc. <http://www.co-ode.org/>
  - **W3C Semantic Web Activity**. Various pointers, standardization efforts, etc. <http://www.w3.org/2001/sw/>
  - **OWL Resources**: OWL-Guide (<http://www.w3.org/TR/owl-guide/>), OWL-Reference (<http://www.w3.org/TR/owl-ref/>)
  - **Pizza Tutorials**. <http://www.co-ode.org/resources/tutorials/>
- Academic Papers/Collections
  - Barry Smith, <http://ontology.buffalo.edu/smith/>, various papers on ontologies (even in ecology)
  - Collection of ontology engineering references. <http://www.aifb.uni-karlsruhe.de/WBS/cte/ontologyengineering/>
  - Mario Bunge. *Treatise on Basic Philosophy, Vol. 3, Ontology I: The Furniture of the World*. D. Reidel Publishing Company, 1977.
  - Nicola Guarino. Formal ontology and information systems. In *Proc. of Formal Ontology in Information Systems*, IOS Press, pp. 3-15, 1998.
  - Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. In *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Kluwer Academic Publishers, 1993.
  - Jeffrey Parsons and Yair Wand. Emancipating instances from the tyranny of classes in information modeling. In *ACM Transactions on Database Systems*, 25(2):228-268, 2000.