Dr. Nabil R. Adam
At Rutgers University in Newark, New Jersey, Dr. Nabil R. Adam is a Professor of Computers and Information Systems; the Founding Director of the Rutgers University Center for Information Management, Integration and Connectivity (CIMIC); past Director of the Meadowlands Environmental Research Institute; and the Director of the Laboratory for Water Security. Dr. Adam has published numerous technical papers in such journals as *IEEE Transactions on Software Engineering*, *IEEE Transactions on Knowledge and Data Engineering*, *ACM Computing Surveys*, *Communications of the ACM*, *Journal of Management Information Systems*, and *International Journal of Intelligent and Cooperative Information Systems*. He has co-authored/co-edited ten books. Dr. Adam is the co-founder and the Executive-Editor-in-Chief of the International Journal on Digital Libraries and serves on the editorial board of a number of journals including *Journal of Management Information Systems*, the *Journal of Electronic Commerce*, and the *Journal of Electronic Commerce Research and Applications*. He is also the co-founder and past chair of the IEEE Technical Committee on Digital Libraries.

Dr. Adam's research work has been supported by over $15 million from various federal and state agencies, including the National Science Foundation (NSF), the National Security Agency (NSA), NOAA, the U.S. Environmental Protection Agency, the Defense Logistics Agency (DLA), the National Library of Medicine, the New Jersey Meadowlands Commission, and NASA.

He has been invited lecture at several national and international institutions/forum including: The first US-China International Workshop on Digital Government Research and Practice (IntDG 2006), Beijing, China, Sponsored by U.S. National Science Foundation, the Chinese Academy of Sciences, and the National Natural Science Foundation of China, 2006; The Seventh World Congress on the Management of e-Business, Halifax, Canada, 2006; ETRIC, International Conference on Emerging Trends in Information and Communication Security, Germany 2006; IEEE Workshop Working Together: R&D Partnership in Homeland Security, the National Conference on Digital Government Research, 2005; Digital Library Colloquium Speaker Series, Carnegie Mellon School of Computer Science AND Lab of Education and Research on Security Assured Information Systems, University of Pittsburgh, 2005; Hungarian US R&D Workshop - Information Society technology and Research Challenges, Sponsored by NSF and ELTE Ithaka, Budapest, Hosted by Hungarian Ministry for Information &Telecommunications and Ministry of Education, Hungary, 2004; The National Conference for Digital Government Research, May 2002; The National Research Council's Workshop on Coping with Increasing Demands on Government Data Centers, 2002; The IEEE/ARL/NASA Workshop on Information Assurance, 2001; The International Symposium on Government and E-commerce Development, Ningbo, China, April 2001, Co-sponsored by the United Nations Department of Economic and Social Affairs, the Ningbo Municipality, the Chinese Academy of Science, the Chinese Academy of Engineering, the Ministry of Information Industry of China, and Zhejiang University of China.

Contact:

E-mail:	adam@adam.rutgers.edu
Telephone:	+1 973 353-5239/1014

Tutorial on Semantic Web and Web Services

The development of the World Wide Web facilitates the advertisement and access of content over the web. The Semantic Web enriches the WWW with content-based descriptions and makes it easier to discover and publish information by machines. The main idea behind the Semantic Web is to make the meaning explicit, thereby allowing machines to process and integrate Web resources intelligently. Beyond enabling quick and accurate web search, this technology may also allow the development of intelligent internet agents and facilitate communication between a multitude of heterogeneous web-accessible devices and services.

The tutorial will be composed of two main parts; the first part shall provide the attendees a detailed understanding of the aims and challenges of Semantic Web, the design of various Semantic Web languages (such as XML, RDF, SHOE, and OWL), the role of ontologies and how to develop them, the knowledge acquisition problem, techniques for scalable reasoning, integrating heterogeneous data sources, web-based agents, and issues in developing semantic-aware applications, especially important for knowledge management applications.; the second part of the tutorial will focus on a specific application and will show how Web Services technology and SOA challenges are resolved with semantics.

Target Audience
The tutorial targets academics, industrial researchers, and developers interested in the next generation web, Semantic Web. Although no specific knowledge is demanded as a pre-requisite for attending the tutorial, basic knowledge about the Web, ontologies, and Service-oriented Architectures will allow attendees to better understand and follow the tutorial.

Topics

Today's Web; From Today's Web to the Semantic Web: Examples; Semantic Web Technologies; A Layered Approach

STRUCTURED WEB DOCUMENTS IN XML
Introduction; The XML Language; Structuring; Namespaces; Addressing and Querying XML Documents; Processing; Summary

DESCRIBING WEB RESOURCES RDF
Introduction; RDF: Basic Ideas; RDF: XML-Based Syntax; RDF Schema: Basic Ideas; RDF Schema: The Language; RDF and RDF Schema in RDF Schema; An Axiomatic Semantics for RDF and RDF Schema; A Direct Inference System for RDF and RDFS; Querying in RQL; Summary

WEB ONTOLOGY LANGUAGE: OWL
Introduction; The OWL Language; Examples; OWL in OWL; Future Extensions; Summary

LOGIC AND INFERENCE: Rules
Introduction; Example of Monotonic Rules: Family Relationships; Monotonic Rules: Syntax; Monotonic Rules: Semantics; Nonmonotonic Rules:
Motivation and Syntax; Example of Nonmonotonic Rules: Brokered Trade; Rule Markup in XML: Monotonic Rules; Rule Markup in XML: Nonmonotonic Rules; Summary

Semantic Web - Ontology and OWL

Nabil R. Adam

Outline

Ontology

- Definition
- Making Use of Ontology, in the Context of Semantic Web, to Make Inferences
- Ontology Language - OWL
- Ontology Development

Ontology – A Basic Component of the SW

- Two databases may use different identifiers for what is in fact the same concept, such as *zip code*.
  - For a program to compare or combine info across the two databases, it has to know that these two terms are being used to mean the same.
  - Ideally, the program must have a way to discover such common meanings for whatever databases it encounters
- Ontologies provide a solution to this problem
- An ontology
  - is a formal conceptualization of a domain that is usable by a computer.
  - aims to make Web resources more readily accessible to automated processes
    - by adding information about the resources that describe or provide Web content

Ontology: Classes, Subclasses, Relations& Properties

- Formally defines the relations among terms.
  - The most typical kind of ontology for the Web has a taxonomy and a set of inference rules.
  - The taxonomy defines classes of objects and relations among them.
    - e.g., an *address* may be defined as a type of *location*, and *city codes* may be defined to apply only to *locations*, and so on.
  - Classes, subclasses and relations among entities are a very powerful tool for Web use.
  - We can express a large number of relations among entities by assigning properties to classes and allowing subclasses to inherit such properties.
- With ontology, solutions to terminology problems begin to emerge.
  - The meaning of terms or XML codes used on a Web page can be defined by pointers from the page to an ontology
  - How about If I point to an ontology that defines *addresses* as containing a *zip code* and you point to one that uses *postal code*?
  - This can be resolved if ontologies (or other Web services) provide equivalence relations

Ontology Vs. XML?

- Note: An ontology differs from an XML schema?
  - Ontology is a knowledge representation, not a message format.
  - Most industry based Web standards consist of a combination of message formats and protocol specifications.
    - e.g., "Upon receipt of this PurchaseOrder message, transfer Amount dollars from Account From to AccountTo and ship Product.“
    - But the specification is not designed to support reasoning outside the transaction context.
    - For example, we won't in general have a mechanism to conclude that because the Product is a type of Chardonnay it must also be a white wine.

Ontology and OWL

- In order to write an ontology that can be interpreted unambiguously and used by software agents we require
  - a syntax and formal semantics for OWL.
- OWL is a vocabulary extension of RDF
- OWL depends on constructs defined by RDF, RDFS, and XML Schema datatypes.
- One advantage of OWL ontologies will be
  - the availability of tools that can reason about them.
  - Tools will provide *generic* support that is not specific to the particular subject domain.

XML – RDF - OWL

- XML- Provides a syntax for structured doc.s.
  - But, no semantic constraints on the meaning of these doc.s
- XML Schema- a language for restricting the structure of XML doc.s and also extends XML with data types
- RDF – A data model for objects (resources) and relations among them
  - Provides a simple semantics for this data model and
    - These data models can be represented in XML syntax
- RDF Schema – A vocabulary for describing properties and classes of RDF resources,
  - With semantics for generalization-hierarchies of such properties and classes
- OWL- Adds more vocabulary for describing properties and classes, including:
  - Relations among classes, e.g., disjointness; cardinality, e.g., exactly 1; equality; rich typing of properties; characteristics of properties, e.g., symmetry; and enumerated classes.

Making Use of Ontology, in the Context of SW
Making Inferences

Wine – Meal Course Example

Example - Wine

- **Wine**

The ontology defines the concept of a wine

According to the specification, a wine is a potable liquid produced by at least one maker of type winery, and is made from at least one type of grape (such grapes are restricted to wine grapes elsewhere in the ontology.)

The full text of the declaration additionally stipulates that a wine comes from a region that is wine-producing and, most importantly to the agent, that a wine has

four properties: color, sugar, body, and flavor.

```
<rdfs:Class rdf:ID="WINE">
 <rdfs:subClassOf rdf:resource="#POTABLE-LIQUID"/>
 <rdfs:subClassOf>
  <daml:Restriction>
    <daml:onProperty rdf:resource="#MAKER"/>
    <daml:minCardinality>
       1
    </daml:minCardinality>
  </daml:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource="#MAKER"/>
    <daml:toClass rdf:resource="#WINERY"/>
   </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction>
     <daml:onProperty rdf:resource="#GRAPE-SLOT"/>
    <daml:minCardinality>
           1       .........
</rdfs:Class>
```

(full text)

Example – Meal Course

- **Meal Course**

The concept of a meal course underlies pairing of a food with a wine.

Each course is a consumable thing comprising at least one food and at least one drink, the latter of which is stipulated to be a wine.

When the user selects a type of course, or an individual food that gets mapped to a type of course,

the agent will consult that course definition for restrictions on the constituent food or wine.

All such course types map back to this concept, like objects to their superclasses in OO programming.

One such example follows...

```
<rdfs:Class rdf:ID="MEAL-COURSE">
 <rdfs:subClassOf rdf:resource="#CONSUMABLETHING"/>
<rdfs:subClassOf>
 <daml:Restriction>
  <daml:onProperty rdf:resource="#FOOD"/>
  <daml:minCardinality>
       1
   </daml:minCardinality>
 </daml:Restriction>
 </rdfs:subClassOf>
 <rdfs:subClassOf>
  <daml:Restriction>
   <daml:onProperty rdf:resource="#FOOD"/>
  <daml:toClass rdf:resource="#EDIBLE-THING"/>
  </daml:Restriction>
  </rdfs:subClassOf>
 <rdfs:subClassOf>
  <daml:Restriction>
    <daml:onProperty rdf:resource="#DRINK"/>
  <daml:minCardinality>
        1      .........
```

Example – Pasta with Spicy Red Sauce

Pasta with Spicy Red Sauce

- Suppose the user selected fra diavolo, or pasta with spicy red sauce directly.
- The concept of such a food is defined elsewhere in the ontology, but most relevant here is the notion of a pasta with spicy red sauce *course*.
- Here the concept is defined as a type of meal course where the food must be a pasta with spicy red sauce.

```
<rdfs:Class rdf:ID="PASTA-WITH-SPICY-RED-SAUCE-COURSE">
    <daml:intersectionOf rdf:parseType="daml:collection">
      <rdfs:Class rdf:about="#MEAL-COURSE"/>
      <daml:Restriction>
        <daml:onProperty rdf:resource="#FOOD"/>
<daml:toClass rdf:resource="#PASTA-WITH-SPICY-RED-SAUCE"/>
      </daml:Restriction>
    </daml:intersectionOf>
    <rdfs:subClassOf
  rdf:resource="#DRINK-HAS-RED-COLOR-RESTRICTION"/>
    <rdfs:subClassOf
  rdf:resource="#DRINK-HAS-FULL-BODY-RESTRICTION"/>
    <rdfs:subClassOf
  rdf:resource="#DRINK-HAS-STRONG-FLAVOR-RESTRICTION"/>
    <rdfs:subClassOf
  rdf:resource="#DRINK-HAS-DRY-SUGAR-RESTRICTION"/>
  </rdfs:Class>
```

•Furthermore, such courses must be a subclass of those with specific restrictions on the properties: of their wines:

- #DRINK-HAS-RED-COLOR-RESTRICTION and the like appear elsewhere, specifying the properties of candidate wines in a straightforward manner.

Example – Chateau Lafite Rothschild Pauillac

Pauillac

- One wine that matches the above restrictions is the Pauillac
- This individual wine is simply defined as a Pauillac whose maker is Chateau Lafite Rothschild.
- Together with other statements in the ontology, this allows the reasoner to infer many additional facts:
  - It is a Medoc wine from Bordeaux, in France, and that it is red, etc.

```
<rdf:Description
rdf:ID="CHATEAU-LAFITE-ROTHSCHILD-PAUILLAC">
      <rdf:type rdf:resource="#PAUILLAC"/>
    <MAKER rdf:resource="#CHATEAU-LAFITE-ROTHSCHILD"/>
  </rdf:Description>

<rdfs:Class rdf:ID="PAUILLAC">
    <rdfs:subClassOf rdf:resource="#FULL-BODY-RESTRICTION"/>
   <rdfs:subClassOf rdf:resource="#STRONG-FLAVOR
    RESTRICTION"/>
    <rdfs:subClassOf rdf:resource=
  "#CABERNET-SAUVIGNON-INDIVIDUAL-GRAPE-SLOT-
    RESTRICTION"/>
    <rdfs:subClassOf rdf:resource=
  "#GRAPE-SLOT-MAX-CARDINALITY-1-RESTRICTION"/>
    <daml:intersectionOf rdf:parseType="daml:collection">
      <rdfs:Class rdf:about="#MEDOC"/>
      <daml:Restriction>
        <daml:onProperty rdf:resource="#REGION"/>
        <daml:hasValue rdf:resource="#PAUILLAC-INDIVIDUAL"/>
      </daml:Restriction>
    </daml:intersectionOf>
  </rdfs:Class>
```

Reasoner

- Following the above example through the ontology reveals
  - a straightforward logical path for pairing
  - the Pauillac with the selected course.
- Because it was specified in a standardized, machine-readable format,
  - it is a straightforward task for any compliant automated reasoner

Ontology Language - OWL

OWL Sublanguages

The OWL language provides three increasingly expressive sublanguages:

- *OWL Lite* supports a classification hierarchy and simple constraint features
  - It should be simpler to provide tool support for OWL Lite than: OWL-DL, OWl Full, and provide a quick migration path for thesauri and other taxonomies.
- *OWL DL* supports users who want the maximum expressiveness without losing computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time) of reasoning systems.
  - OWL DL is so named due to its correspondence with *description logics*, OWL DL has desirable computational properties for reasoning systems.
- *OWL Full* Provides maximum expressiveness and the syntactic freedom of RDF with no computational guarantees.

- Each of these sublanguages is an extension of its simpler predecessor, both in what can be legally expressed and in what can be validly concluded.
- Folowing are the details of each of these languages.

An OWL Ontology: Basic Elements

- Classes
- Properties
- Instances of classes
- Relationships among instances

Data Aggregation and Privacy

- OWL's ability to express ontological info about instances appearing in multiple documents supports linking of data from diverse sources in a principled way.
- The ability to express equivalences using
  - owl:sameAs
    - can be used to state that seemingly different individuals are actually the same.
  - Owl:InverseFunctionalProperty (see detials later)
    - can also be used to link individuals together.
    - e.g., if a property such as
    - "SocialSecurityNumber" is an
    - owl:InverseFunctionalProperty,
    - then two separate individuals could be inferred to be identical based on having the same value of that property.
    - When individuals are determined to be the same by such means, info about them from different sources can be merged.
    - This *aggregation* can be used to determine facts that are not *directly* represented in any one source.
- The ability of the SW to link info from multiple sources is a desirable and powerful feature that can be used in many applications.
- However, this does have potential for abuse -- pottential privacy implications

Simple Named Classes

- Each user-defined class is implicitly a subclass of owl:Thing.
- OWL also defines the empty class, owl:Nothing.
- Domain specific root classes are defined by simply declaring a named class,
  - For our sample wines domain, we create three root classes:
  - Winery, Region, and ConsumableThing.

- <owl:Class rdf:ID="Winery"/>
- <owl:Class rdf:ID="Region"/>
- <owl:Class rdf:ID="ConsumableThing"/>
- Note that we have only said that there exist classes that have been given these names, indicated by the 'rdf:ID=' syntax.

Ontology Definition: Distributed and Incremental

- Ontology definitions may be
- incremental and distributed.
- The syntax
  - rdf:ID="Region"
    - is used to introduce a name, as part of its definition. This is the rdf:ID attribute similar to the familiar ID attribute defined by XML.
- Within this document,
  - the Region class can now be referred to using
    - #Region, e.g. rdf:resource="#Region".
- Other ontologies may reference this name using its complete form,
  - "http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine#Region".

- Another form of reference uses the syntax
  - rdf:about="#Region" to *extend* the definition of a resource.
  - This use of the rdf:about="&ont;#x" syntax
    - is a critical element in the creation of a distributed ontology.
    - It permits the extension of the imported definition of x without modifying the original document and supports the incremental construction of a larger ontology.

Class Constructor

- The fundamental taxonomic constructor for classes is
  - rdfs:subClassOf
    - If X is a subclass of Y, then every instance of X is also an instance of Y.
    - The rdfs:subClassOf relation is transitive.
    - If X is a subclass of Y and Y a subclass of Z then X is a subclass of Z.

```
<owl:Class rdf:ID="PotableLiquid">
  <rdfs:subClassOf rdf:resource="#ConsumableThing" />
...
</owl:Class>
```

- Defines PotableLiquid (liquids suitable for drinking) to be a subclass of ConsumableThing.
- Both of these classes can be defined in a separate ontology that
  - would provide the basic building blocks for a wide variety of food and drink ontologies
    - The food ontology includes a number of classes, e.g., Food, EdibleThing, MealCourse, and Shellfish, etc.

A Class Definition

- A class definition has two parts:
  - 1) a name introduction or reference; and 2) a list of restrictions.
  - Each of the immediate contained expressions in the class definition further restricts the instances of the defined class. Instances of the class belong to the intersection of the restrictions.
- Below is a simple definition for the class Wine. Wine is a PotableLiquid.

```
<owl:Class rdf:ID="Wine">
   <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
   <rdfs:label xml:lang="en">wine</rdfs:label> Lable is similar to a comment
   <rdfs:label xml:lang="fr">vin</rdfs:label>
...
</owl:Class>
<owl:Class rdf:ID="Pasta">  the definition of Pasta as an EdibleThing
   <rdfs:subClassOf rdf:resource="#EdibleThing" />
...
</owl:Class>
```

The rdfs:label entry provides an optional human readable name for this class. Presentation tools can make use of it. The "lang" attribute provides support for multiple languages.

Individuals

- Indiviudlas – members of classses
- An individual is minimally introduced by declaring it to be a member of a class.

<Region rdf:ID="CentralCoastRegion" /> *Note the following is identical in meaning to the example*.

<owl:Thing rdf:ID="CentralCoastRegion" />

<owl:Thing rdf:about="#CentralCoastRegion">

  <rdf:type rdf:resource="#Region"/>

</owl:Thing>

- Here, *rdf:type* is an RDF property that ties an individual to a class of which it is a member.
- Note:
  - We have decided that CentralCoastRegion (a specific area) is member of Region, the class containing all geographical regions.
- Note: "Region" and "CentralCoastRegion" need NOT to be in the same file once the names are exttended with a URI.
  - They can be imported and augmented, creating derived ontologies.
  - Thus being able to design Web ontologies to be distributed.
- Another example of an individual in the Grape ontology, we can define an individual: the Cabernet Sauvignon grape varietal. It is an individual since it denotes a single grape varietal

Class vs. an Individual

- A class is simply a name and collection of properties that describe a set of individuals.
- Individuals are the members of a set
- Thus classes should correspond to naturally occurring sets of things in a domain of discourse, and
  - individuals should correspond to actual entities that can be grouped into these classes.
- Distinction between classes and invdividuals
  - *Levels of representation:* In certain contexts something that is obviously a class can itself be considered an instance of something else,
    - e.g., in the wine ontology we have the notion of a Grape, which is intended to denote the set of all *grape varietals.* CabernetSauvingonGrape is an example of an instance of this class, as it denotes the actual grape varietal called Cabernet Sauvignon. However, CabernetSauvignonGrape could itself be considered a class, the set of all actual Cabernet Sauvignon grapes.

Properties

- A property is a binary relation.
- Two types of properties are distinguished:
  - *datatype properties*,
  - *object properties*.
- Restrictions on properties, can be accomplished through
  - The domain and range can be specified. The property can be defined to be a specialization (subproperty) of an existing property.

```
<owl:ObjectProperty rdf:ID="madeFromGrape">
  <rdfs:domain rdf:resource="#Wine"/>
  <rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
```

- Here, madeFromGrape has a domain of Wine *and* a range of WineGrape,
  - i.e., it relates instances of the class Wine to instances of the class WineGrape.
  - Multiple domains means that the domain of the property is the intersection of the identified classes (and similarly for range).
- Note: In OWL, a sequence of elements without an explicit operator represents an implicit conjunction

```
<owl:ObjectProperty rdf:ID="course">
    <rdfs:domain rdf:resource="#Meal" />
<rdfs:range rdf:resource="#MealCourse" />
</owl:ObjectProperty>.
```
the property course ties a Meal to a MealCourse.

Properties (Cont.)

- In OWL, a range may be used to infer a type.
  - For example, given:

```
<owl:Thing rdf:ID="LindemansBin65Chardonnay">
  <madeFromGrape rdf:resource="#ChardonnayGrape"/>
</owl:Thing>
```

  - we can infer that LindemansBin65Chardonnay is a wine because the domain of madeFromGrape is Wine

Properties Hierarchy

- Properties, like classes, can be arranged in a hierarchy.
  - For example, given:

```
<owl:Class rdf:ID="WineDescriptor" />
<owl:Class rdf:ID="WineColor">
   <rdfs:subClassOf rdf:resource="#WineDescriptor" />
    ...
</owl:Class>
```

We can have:

```
<owl:ObjectProperty rdf:ID="hasWineDescriptor">
     <rdfs:domain rdf:resource="#Wine" />
     <rdfs:range  rdf:resource="#WineDescriptor"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasColor">
   <rdfs:subPropertyOf rdf:resource="#hasWineDescriptor" />
   <rdfs:range rdf:resource="#WineColor" />
     ...
</owl:ObjectProperty>
```

Relates wines to their color& componenets of their taste, including sweetness, body and flavor

hasColor is a subproperty of hasWinDescriptor property,

with a range further restricted to WineColor.

Properties (Cont.)

- It is now possible to expand the definition of Wine to include the notion that a wine is
  - made from at least one WineGrape.
- As with property definitions, class definitions have multiple subparts that are implicitly conjoined.

```
<owl:Class rdf:ID="Wine">
<rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
   <rdfs:subClassOf>
      <owl:Restriction>
         <owl:onProperty rdf:resource="#madeFromGrape"/>
            <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
                  1
            </owl:minCardinality>
     </owl:Restriction>
  </rdfs:subClassOf>
...
</owl:Class>
```

Properties (Cont.)

- We can now describe the class of Vintages, discussed previously

```
<owl:Class rdf:ID="Vintage">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#vintageOf"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
        1
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

The property *vintageOf* ties a Vintage to a Wine.

```
<owl:ObjectProperty rdf:ID="vintageOf">
<rdfs:domain rdf:resource="#Vintage" />
<rdfs:range  rdf:resource="#Wine"/>
</owl:ObjectProperty>
```

Properties and Data Types

- Datatype properties may range over RDF literals or simple types defined in accordance with XML Schema datatypes.
- OWL uses most of the built-in XML Schema datatypes.
- References to these datatypes are by means of the URI reference for the datatype, http://www.w3.org/2001/XMLSchema.

Examples include:

- xsd:string; xsd:normalizedString; xsd:boolean; xsd:decimal; xsd:floatxsd:double; xsd:integer; xsd:nonNegativeInteger; xsd:positiveInteger; xsd:nonPositiveInteger;
- See OWL Web Ontology Lang Guide for more details
- Example,

```
<owl:Class rdf:ID="VintageYear" />


<owl:DatatypeProperty rdf:ID="yearValue">
  <rdfs:domain rdf:resource="#VintageYear" />
  <rdfs:range  rdf:resource="&xsd;positiveInteger"/>
</owl:DatatypeProperty>
```

The *yearValue* property relates VintageYears to positive integer values. We introduce the hasVintageYear property, which relates a Vintage to a VintageYear

Property Characteristics

- Following are property characteristics which provide a powerful mechanism for enhancing reasoning about a property:
- Transitivity Property
- Symmetric Property
- Functional Property
- Inverse of Property
- Inverse of Functional Property

Property Characteristics - Transitivity

- If a property, P, is specified as transitive then
  - for any x, y, and z: P(x,y) and P(y,z) implies P(x,z)
- The property locatedIn is transitive.

```
<owl:ObjectProperty rdf:ID="locatedIn">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  <rdfs:domain rdf:resource="&owl;Thing" />
  <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>

<Region rdf:ID="SantaCruzMountainsRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
</Region>

<Region rdf:ID="CaliforniaRegion">
  <locatedIn rdf:resource="#USRegion" />
</Region>
```

- Because the *SantaCruzMountainsRegion* is locatedIn the *CaliforniaRegion*, then it must also be locatedIn the *USRegion*, since locatedIn is transitive.

Property Characteristics - Symmetry

- If a property, P, is tagged as symmetric
  - then for any x and y: P(x,y) iff P(y,x)
- The property adjacentRegion is symmetric, while locatedIn is not.
-

```
<owl:ObjectProperty rdf:ID="adjacentRegion">
  <rdf:type rdf:resource="&owl;SymmetricProperty" />
  <rdfs:domain rdf:resource="#Region" />
  <rdfs:range rdf:resource="#Region" />
</owl:ObjectProperty>

<Region rdf:ID="MendocinoRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
  <adjacentRegion rdf:resource="#SonomaRegion" />
</Region>
```

- The MendocinoRegion is adjacent to the SonomaRegion and vice-versa. The MendocinoRegion is located in the CaliforniaRegion but not vice versa.

Property Characteristics - Functional

- If a property, P, is tagged as functional
  - then for all x, y, and z: P(x,y) and P(x,z) implies y = z
- E.g., in our wine ontology,
  - hasVintageYear is functional.
    - A wine has a unique vintage year, i.e., a given individual Vintage can only be associated with a single year using the hasVintageYear property.

```
<owl:Class rdf:ID="VintageYear" />

<owl:ObjectProperty rdf:ID="hasVintageYear">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Vintage" />
  <rdfs:range  rdf:resource="#VintageYear" />
</owl:ObjectProperty>
```

Property Characteristics – Inverse of

- If a property, P1, is tagged as the *owl:inverseOf* P2,
  - then for all x and y: P1(x,y) iff P2(y,x)
- Note that the syntax for *owl:inverseOf* takes a property name as an argument. A iff B means (A implies B) and (B implies A).

```
<owl:ObjectProperty rdf:ID="hasMaker">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
</owl:ObjectProperty>


<owl:ObjectProperty rdf:ID="producesWine">
  <owl:inverseOf rdf:resource="#hasMaker" />
</owl:ObjectProperty>
```

1. Wines have makers, which in the definition of Wine are restricted to Winerys. Then each Winery produces the set of wines that identify it as maker.

Property Restrictions – AllValuesFrom

- *allValuesFrom* and *someValuesFrom*, are local to their containing class definition.
- The owl:allValuesFrom restriction requires that
  - for every instance of the class that has instances of the specified property, the values of the property are all members of the class indicated by the owl:allValuesFrom clause.

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:allValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

The maker of a Wine must be a Winery. The allValuesFrom restriction is on the hasMaker property of this Wine class only. Makers of Cheese are not constrained by this local restriction.

Property Restrictions – SomeValuesFrom

- *owl:someValuesFrom* is similar to owl:allValuesFrom.
  - If we replaced owl:allValuesFrom with owl:someValuesFrom in the example above, it would mean that
    - at least one of the hasMaker properties of a Wine must point to an individual that is a Winery.

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food;PotableLiquid" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:someValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

- The difference between the two formulations is the difference between a universal and existential quantification.

allValuesFrom and someValuesFrom

- Note:
- The difference between the two formulations is the difference between a universal and existential quantification.

Relation	Implications
allValuesFrom	- For all wines, if they have makers, all the makers are wineries.
someValuesFrom	For all wines, they have at least one maker that is a winery.

- The first does not require a wine to have a maker. If it does have one or more, they must all be wineries.
- The second requires that there be at least one maker that is a winery, but there may be makers that are not wineries.

Property Restrictions - Cardinality

- In addition to specifying the minimum cardinality (as shown earlier), we can also specify the exact # of elements in a relation.

Example, below specifies Vintage to be a class with exactly 1 VintageYear.

```
<owl:Class rdf:ID="Vintage">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasVintageYear"/>
      <owl:cardinality
     rdf:datatype="&xsd;nonNegativeInteger">1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

- Earlier, we specified hasVintageYear to be a functional property, which is the same as saying that
  - every Vintage has at most one VintageYear.
  - This application of that property to Vintage using the cardinality restriction asserts something stronger, that every Vintage has exactly one VintageYear.
- owl:maxCardinality can be used to specify an upper bound. owl:minCardinality can be used to specify a lower bound.
- In combination, the two can be used to limit the property's cardinality to a numeric interval

Property Restrictions - hasValue

- *hasValue* allows us to specify classes based on the existence of particular property values.
  - Hence, an individual will be a member of such a class whenever at least one of its property values is equal to the hasValue resource.

```
<owl:Class rdf:ID="Burgundy">
 ...
 <rdfs:subClassOf>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#hasSugar" />
   <owl:hasValue rdf:resource="#Dry" />
  </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

- Here we declare that all Burgundy wines are dry,
  - i.e., their hasSugar property must have at least one value that is equal to Dry.
- As for allValuesFrom and someValuesFrom, this is a local restriction. It holds for hasSugar as applied to Burgundy.

Reusing & Sharing Ontologies: Ontology Mapping

- To achieve maximum impact,
  - Ontologies need to be widely shared.
- Tominimize the intellectual effort in developing an ontology
  - Ontologies need to be re-used.
- If you can find an existing ontology that has already undergone extensive use and refinement, it makes sense to adopt it.
  - For example, we might adopt a date ontology from one source and a physical location ontology from another and then extend the notion of location to include the time period during which it holds.
- Much of the effort of developing an ontology is devoted to
  - hooking together classes and properties in ways that maximize implications.

Ontology Mapping

- It will be challenging to merge a collection of ontologies – while maintaing consistency.
- Tool support will almost certainly be required to maintain consistency.
  - Equivalence among Classes and Properties
  - Identity among individuals
  - Different Individuals
- See OWL web Ontology Language Guide for more details

Complex Classes – Owl DL

- Provides constructors with which to form classes
  - Set Operators – intersectionOf; unionOf; ComplementOf
  - Enumerated Classes – oneOf
  - Disjoint Classes – disjointWith

Ontology Development

ONTOLOGY EDITING ENVIRONMENTS

- Protege
- Ontolingua
- Chimera

Example

STEPS OF DEVELOPING ONTOLOGY

1. Determining the domain and scope of the ontology
2. Reusing existing ontologies
3. Enumerating important terms in the ontology
4. Defining the classes and the class hierarchy
5. Defining the slots (properties) of classes
6. Defining the facets of the slots
7. Creating instances

1-DETERMINING THE DOMAIN AND THE SCOPE

- What is the domain that the ontology will cover?
  - Newspaper
- For what type of questions the information in the ontology should provide answers?
  - What will be the content of newspaper?
  - What are the phone numbers of columnists?
- Who will use the application of ontology?
  - Newspaper Organization

2-REUSING EXISTING ONTOLOGIES

- Refining and extending existing resources
- Becomes requirement if our system needs to interact with existing applications using ontologies
- Libraries of reusable ontologies
  - Ontolingua ontology library
    - http://www.ksl.stanford.edu/software/ontolingua/
  - DAML ontology library
    - http://www.daml.org/ontologies/

3-ENUMERATING IMPORTANT TERMS

- Making a list of important terms and their properties that need to be explained to users or we would like to talk about
- Newspaper
  - Content
  - Number of Pages
  - Page

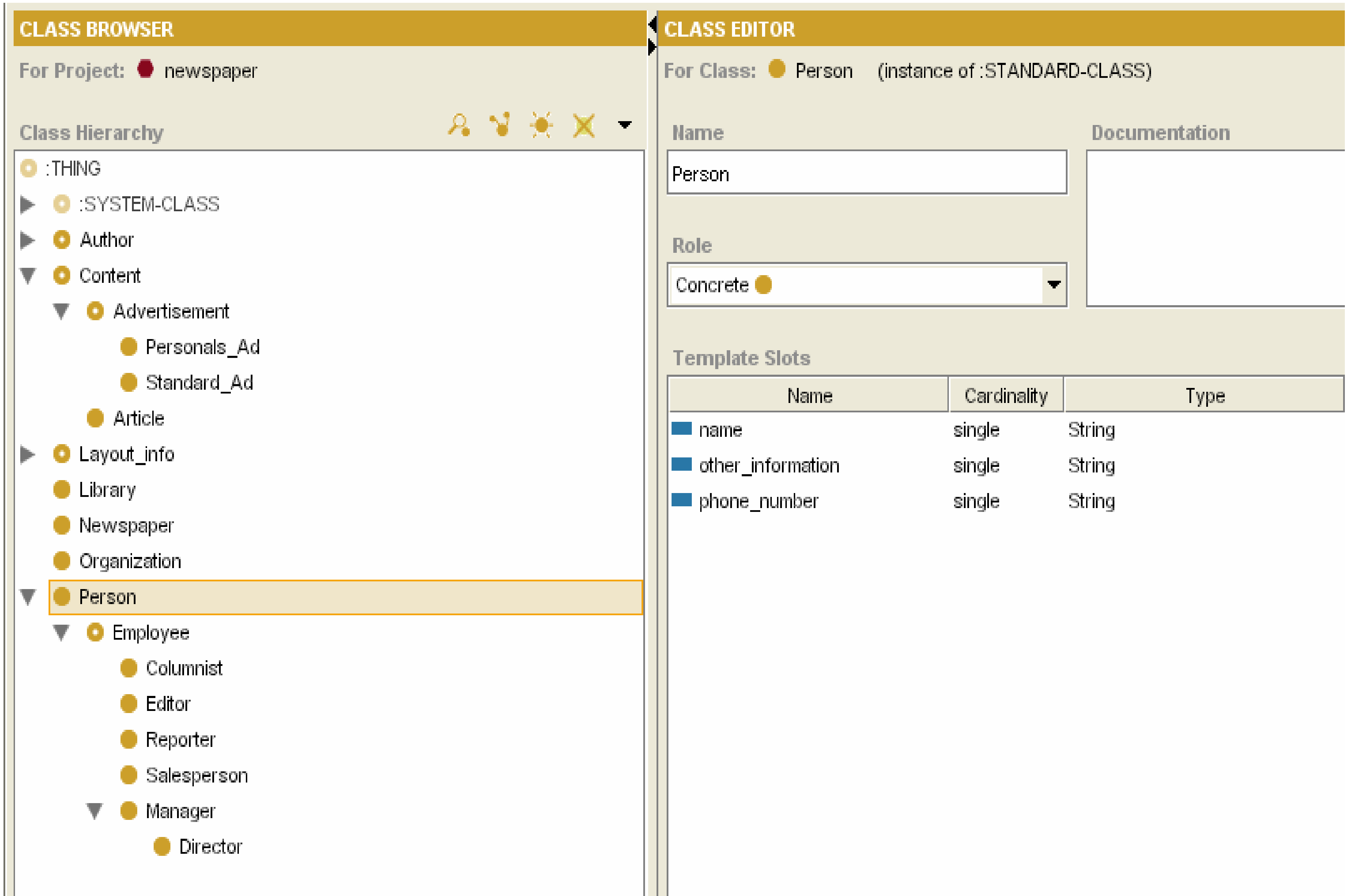
- Author
- Content
- Layout information

4-DEFINING CLASSES AND THE HIERARCHY

- Possible approaches to develop a class hierarchy
  - ***Top-down Approach***
    - superclass →subclass
  - ***Bottom-up Approach***
    - subclass→superclass
  - ***Combination Approach***
    - defining more important concepts, then generalizing and specializing them by using both approaches
- None of the approaches is better than the others
- Approach choice depends on the personal preference

Example

5-DEFINING SLOTS (PROPERTIES) OF CLASSES



CLASS BROWSER

For Project: newspaper

Class Hierarchy

- :THING
  - :SYSTEM-CLASS
  - Author
  - Content
    - Advertisement
      - Personals_Ad
      - Standard_Ad
    - Article
  - Layout_info
  - Library
  - Newspaper
  - Organization
  - Person
    - Employee
      - Columnist
      - Editor
      - Reporter
      - Salesperson
      - Manager
        - Director

CLASS EDITOR

For Class: Person (instance of :STANDARD-CLASS)

Name: Person

Documentation

Role: Concrete

Template Slots

Name	Cardinality	Type
name	single	String
other_information	single	String
phone_number	single	String

6-DEFINING BORDERS OF SLOTS (PROPERTIES)

- ***<u>Slot Cardinality:</u>***
  - defines number of values a slot can have
  - single cardinality (one value)
  - multiple cardinality (more than one value)

- ***<u>Slot value type:</u>***
  - String
  - Number (Integer, Floating)
  - Boolean (True, False)
  - Enumerated (flavor: strong, moderate, delicate)
  - Instance : defines allowed classes from which instances can come

Example

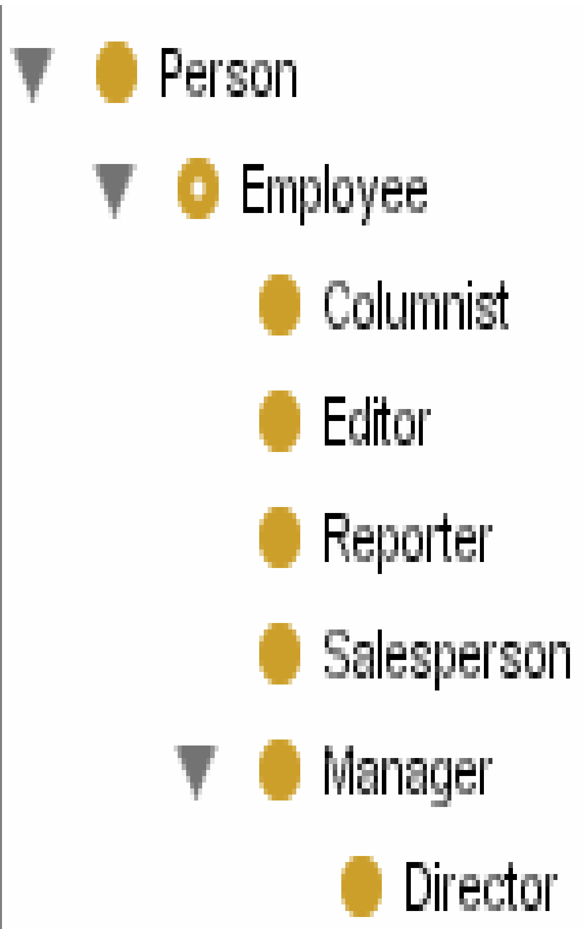
7-Creating Instances

1. Choosing a class
2. Creating an individual instance
3. Filling in the slot values

Example

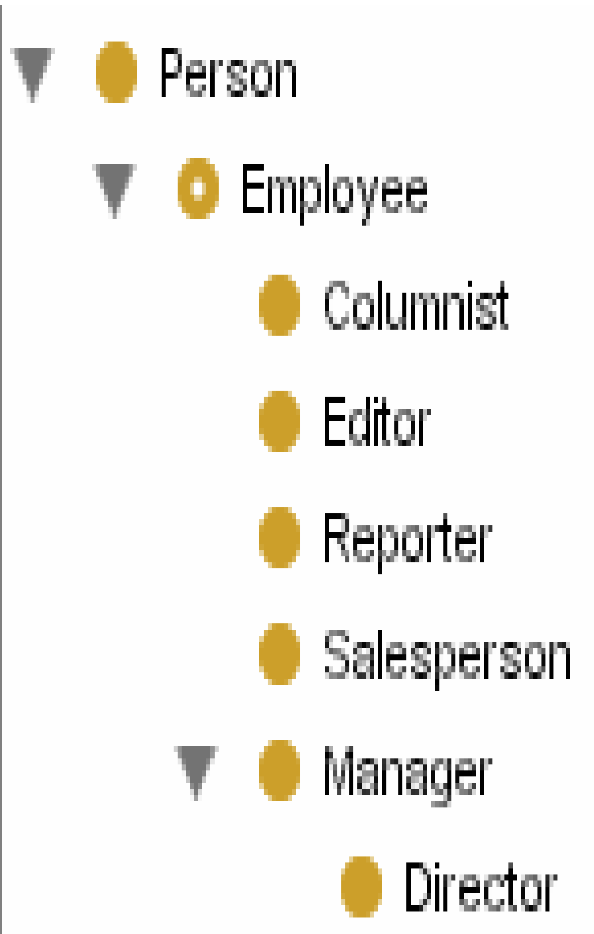
ENSURING CORRECT CLASS HIERARCHY

- ***Is a relation***
  - an employee is a person
  - a person may not be an employee
- Synonyms for the same concept do not represent different classes
  - worker== employee
- ***Transitivity of the hierarchical relations***
  - Columnist →Employee, Employee → Person, Columnist → Person
- Columnist is the ***direct*** subclass of Employee
- Employee is the ***direct*** subclass of Person
- Columnist is the ***indirect*** subclass of Person



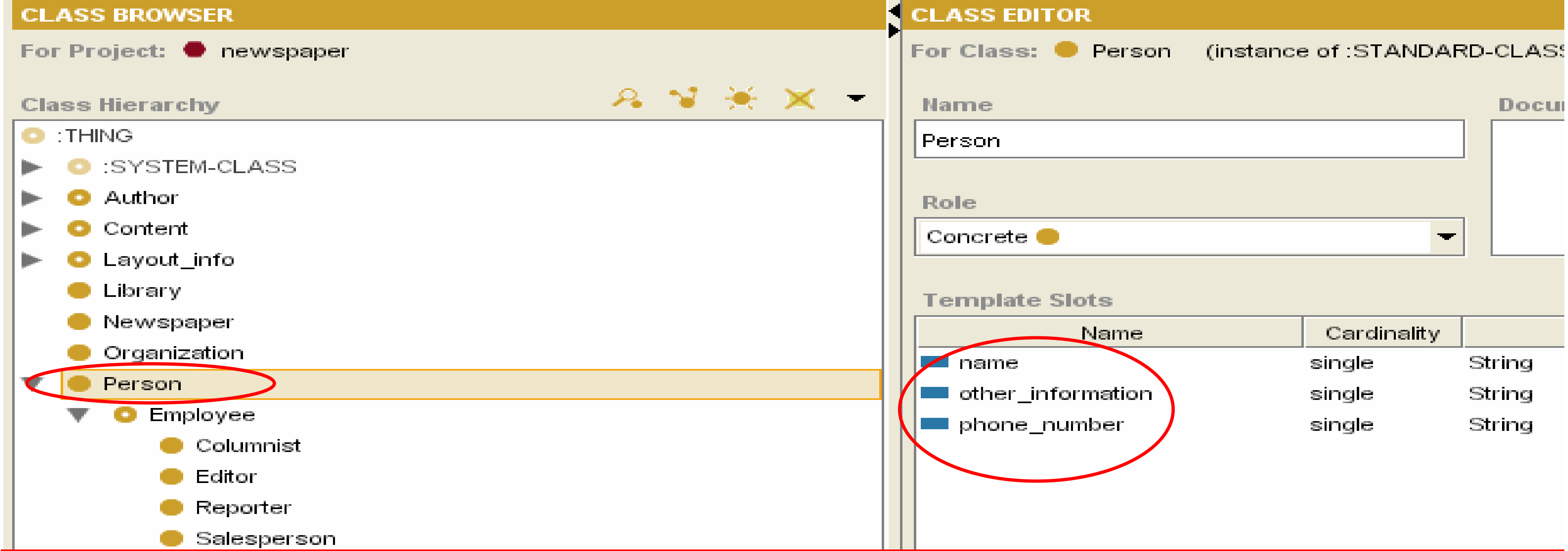
SIBLINGS IN A CLASS HIERARCHY

- Direct subclasses of the same superclass
- ***Number of siblings***
  - 1< number of siblings < 1000
  - one sibling: modeling problem, incomplete ontology, unnecessary class
  - more than thousand: need for intermediate categories



WHEN TO INTRODUCE NEW CLASS

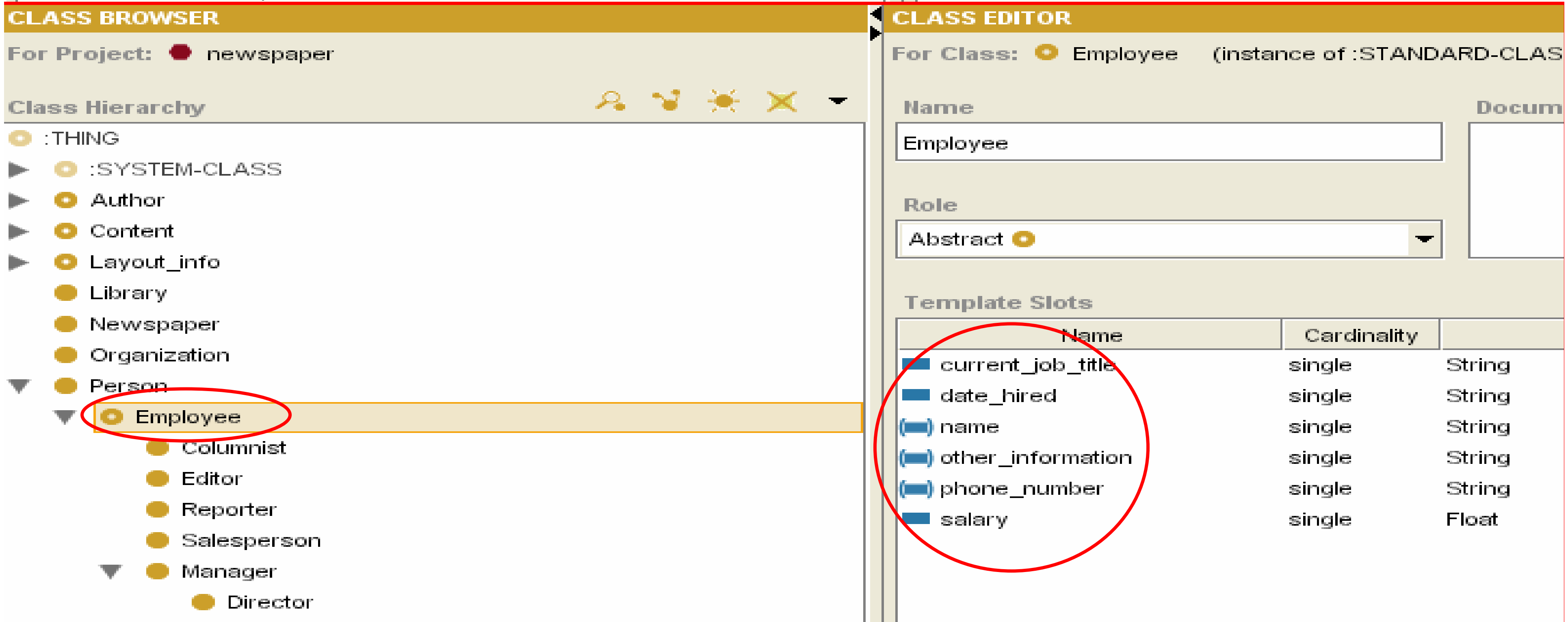
- ***Introduce new class if subclass :***
  - has additional properties superclass does not have
  - has restrictions different from those of superclass
    - overriding
- Classes in terminological hierarchies do not have to introduce new properties
  - classification of diseases
- Previous class definitions by the domain experts
- ***Property or a new class?***
  - Red Wine, White Wine (important)
  - Red House, Blue House



Name: Person

Role: Concrete

Name	Cardinality	
name	single	String
other_information	single	String
phone_number	single	String



Name: Employee

Role: Abstract

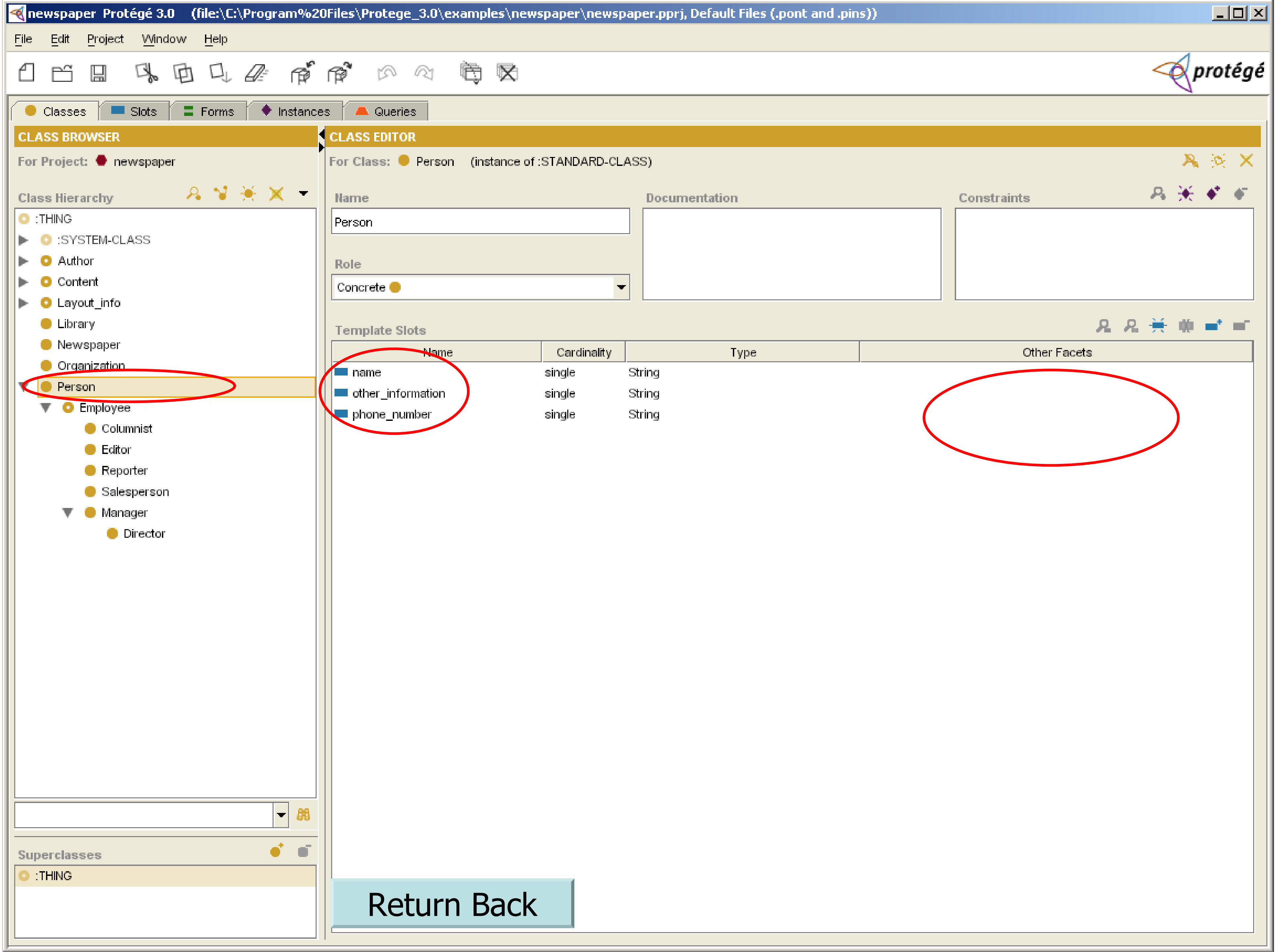
Name	Cardinality	
current_job_title	single	String
date_hired	single	String
name	single	String
other_information	single	String
phone_number	single	String
salary	single	Float

STANDARDS FOR NAMING

- Defining standards for naming makes ontology much more understandable and avoids modeling mistakes
- Using capital letter for class name and lowercase letter for slot
  - Employee. name
- Deciding whether to use singular or plural names
  - Author or Authors

- Avoiding attaching strings like class, slot or property
  - Employee Class, name property
- Avoiding abbreviations
- All subclasses should either include or not include the name of superclass

EXAMPLES



newspaper Protégé 3.0 (file:\C:\Program%20Files\Protege_3.0\examples\newspaper\newspaper.pprj, Default Files (.pont and .pins))

File Edit Project Window Help

Classes | Slots | Forms | Instances | Queries

CLASS BROWSER

For Project: newspaper

Class Hierarchy

- :THING
  - :SYSTEM-CLASS
  - Author
  - Content
  - Layout_info
  - Library
  - Newspaper
  - Organization
  - Person
    - Employee
      - Columnist
      - Editor
      - Reporter
      - Salesperson
      - Manager
        - Director

Superclasses

:THING

CLASS EDITOR

For Class: Person (instance of :STANDARD-CLASS)

Name: Person

Documentation

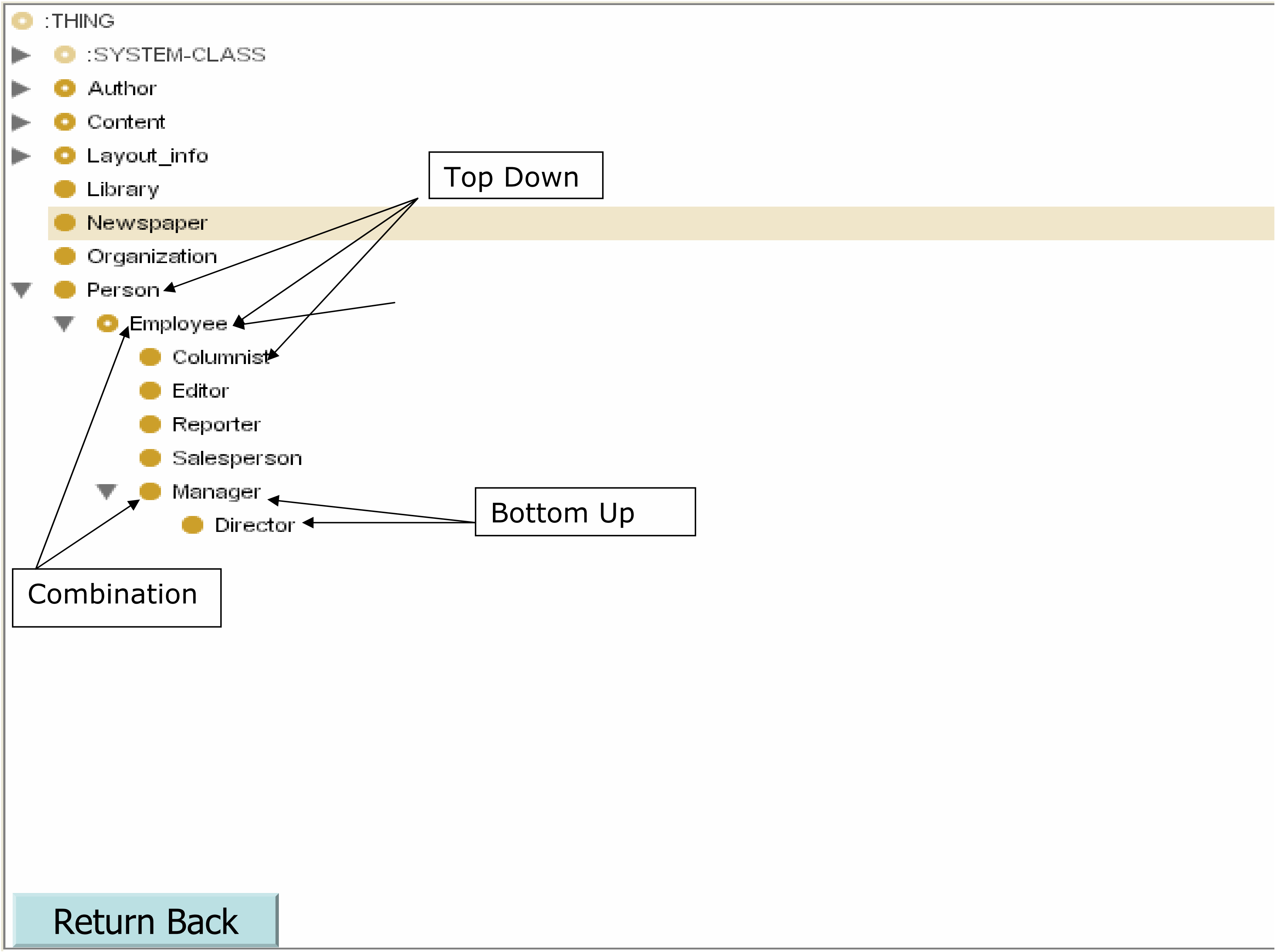
Constraints

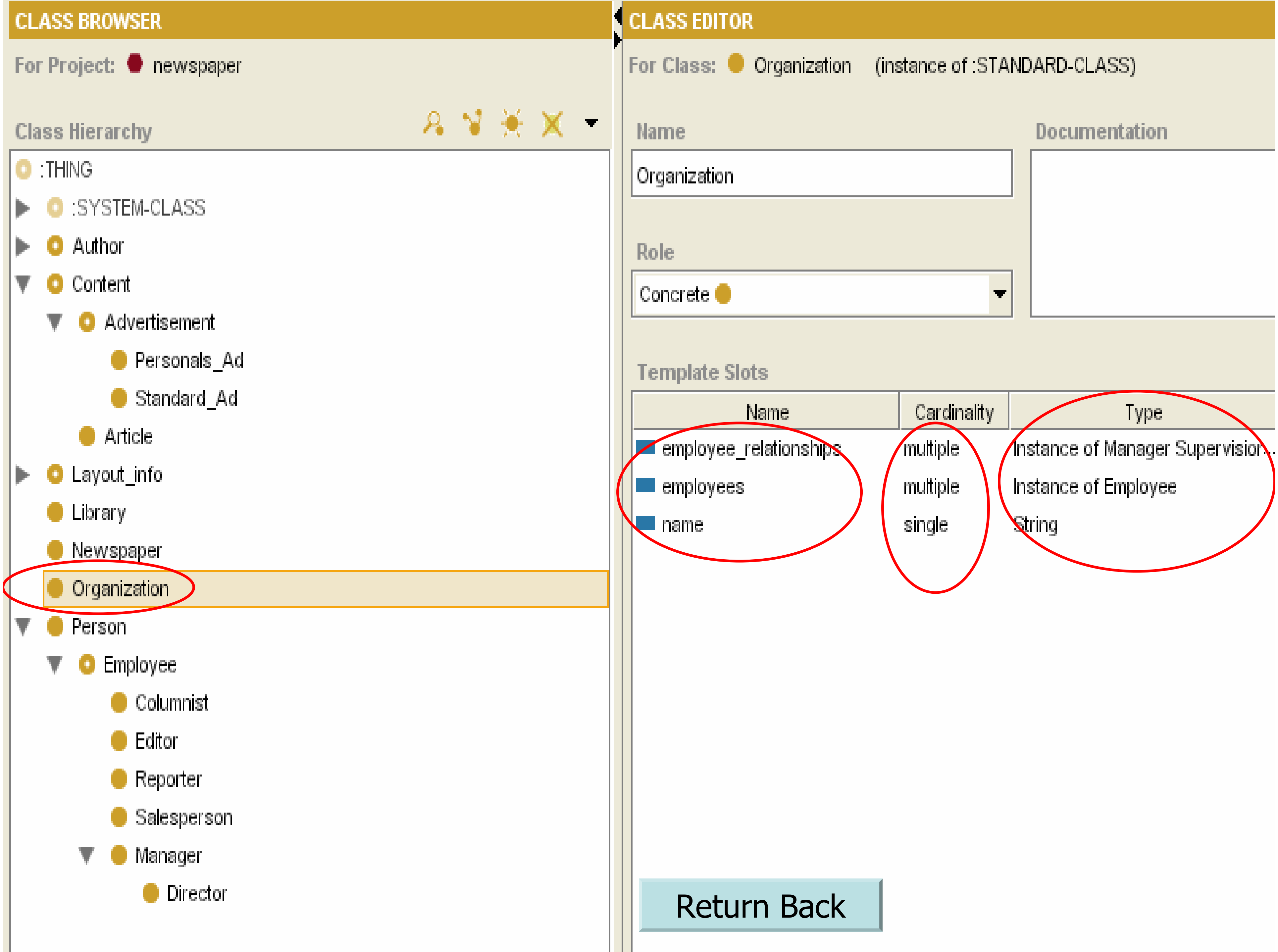
Role: Concrete

Template Slots

Name	Cardinality	Type	Other Facets
name	single	String	
other_information	single	String	
phone_number	single	String	

Return Back


:THING
:SYSTEM-CLASS
Author
Content
Layout_info
Library
Newspaper
Organization
Person
Employee
Columnist
Editor
Reporter
Salesperson
Manager
Director
Top Down
Bottom Up
Combination
Return Back



:THING
- :SYSTEM-CLASS
- Author
- Content
  - Advertisement
    - Personals_Ad
    - Standard_Ad
  - Article
- Layout_info
- Library
- Newspaper
- Organization
- Person
  - Employee
    - Columnist
    - Editor
    - Reporter
    - Salesperson
    - Manager
      - Director

For Class: Organization (instance of :STANDARD-CLASS)

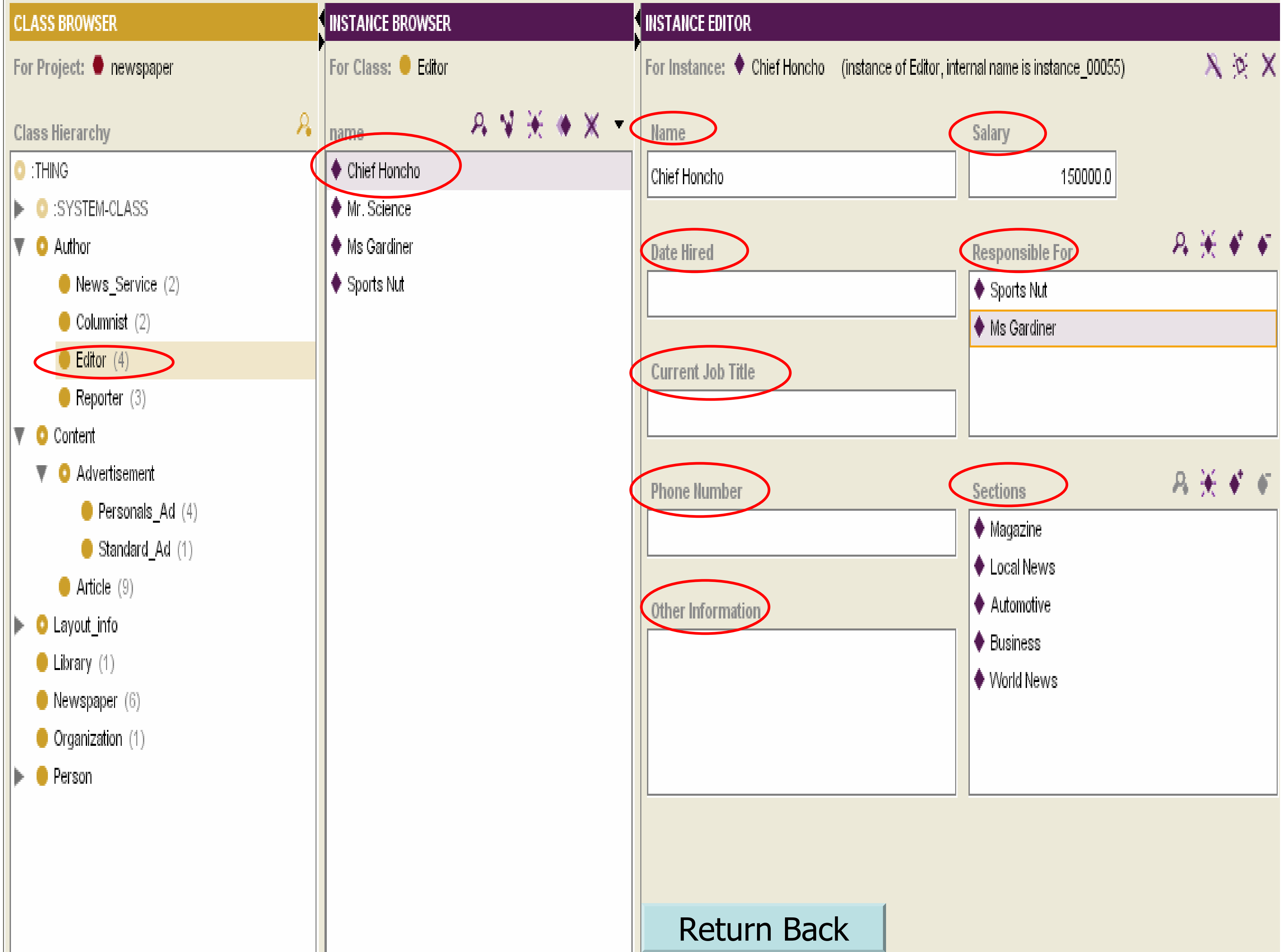
Name: Organization

Role: Concrete

Documentation:

Template Slots

Name	Cardinality	Type
employee_relationships	multiple	Instance of Manager Supervisior...
employees	multiple	Instance of Employee
name	single	String



CLASS BROWSER

For Project: newspaper

Class Hierarchy

- :THING
- :SYSTEM-CLASS
- Author
  - News_Service (2)
  - Columnist (2)
  - Editor (4)
  - Reporter (3)
- Content
  - Advertisement
    - Personals_Ad (4)
    - Standard_Ad (1)
  - Article (9)
- Layout_info
- Library (1)
- Newspaper (6)
- Organization (1)
- Person

INSTANCE BROWSER

For Class: Editor

name

- Chief Honcho
- Mr. Science
- Ms Gardiner
- Sports Nut

INSTANCE EDITOR

For Instance: Chief Honcho (instance of Editor, internal name is instance_00055)

Name: Chief Honcho

Salary: 150000.0

Date Hired:

Responsible For:
- Sports Nut
- Ms Gardiner

Current Job Title:

Phone Number:

Sections:
- Magazine
- Local News
- Automotive
- Business
- World News

Other Information:

Return Back