A GUIDE FOR ONTOLOGY DEVELOPMENT WITH DISCIPLE

Gheorghe Tecuci and Mihai Boicu

This paper is a gentle introduction to ontologies, their design, and their development. It introduces basic notions, representational aspects, ontology development tools, and knowledge engineering guidelines. It also contains specific instructions for using the ontology tools of the Disciple agent development environment, hands-on experience sections, and exercises. The paper is addressed to anyone who wants to learn about ontologies through a hands-on guided practice with Disciple.

GEORGE MASON UNIVERSITY

RESEARCH REPORT 3

VOLUME 2008

Learning Agents Center MASON

A GUIDE FOR ONTOLOGY DEVELOPMENT WITH DISCIPLE

Gheorghe Tecuci and Mihai Boicu

Learning Agents Center, George Mason University, Fairfax, VA, 22030, USA

http://lac.gmu.edu

Keywords: ontology, concepts, generalization, ontology development tools, knowledge engineering

***Disclaimer**: The views and opinions expressed in this article are those of the authors and do not necessarily reflect the official policy or position of the National Science Foundation, the Air Force Office of Scientific Research, or any other agency of the U.S. government.*

22 August, 2008

A Guide for Ontology Development with Disciple

1. What is an Ontology

*An **ontology** is an explicit formal specification of the terms that are used to represent an agent's world* (Gruber 1993).

In an ontology, definitions associate names of entities in the agent's world (e.g., classes, individual objects, relations, problems) with human-readable text and formal axioms. The text describes what a name means. The axioms constrain the interpretation and use of a term. Examples of terms from the ontology of the PhD Advisor Assessment Agent include: student, PhD student, professor, course, and publication. The PhD Advisor Assessment Agent is a Disciple agent that helps a PhD student in selecting a PhD Advisor based on a detailed analysis of several factors, including: professional reputation, personality and compatibility with the student, learning experience of advisor's students, responsiveness to students, support offered to students, and quality of the results of previous students. This agent will be used to illustrate the various ontology issues discussed in this paper.

We define the **object ontology** as a hierarchical representation of the objects from the application domain. It includes both descriptions of the different types of objects (called concepts, such as professor or course), and descriptions of individual objects (called instances, such as CS580), together with the properties of each object and the relationships between objects.

The underlying idea of the ontological representation is to represent knowledge in the form of a graph (similar to a concept map), in which the nodes represent objects, situations, or events, and the arcs represent the relationships between them, as illustrated in Figure 1.

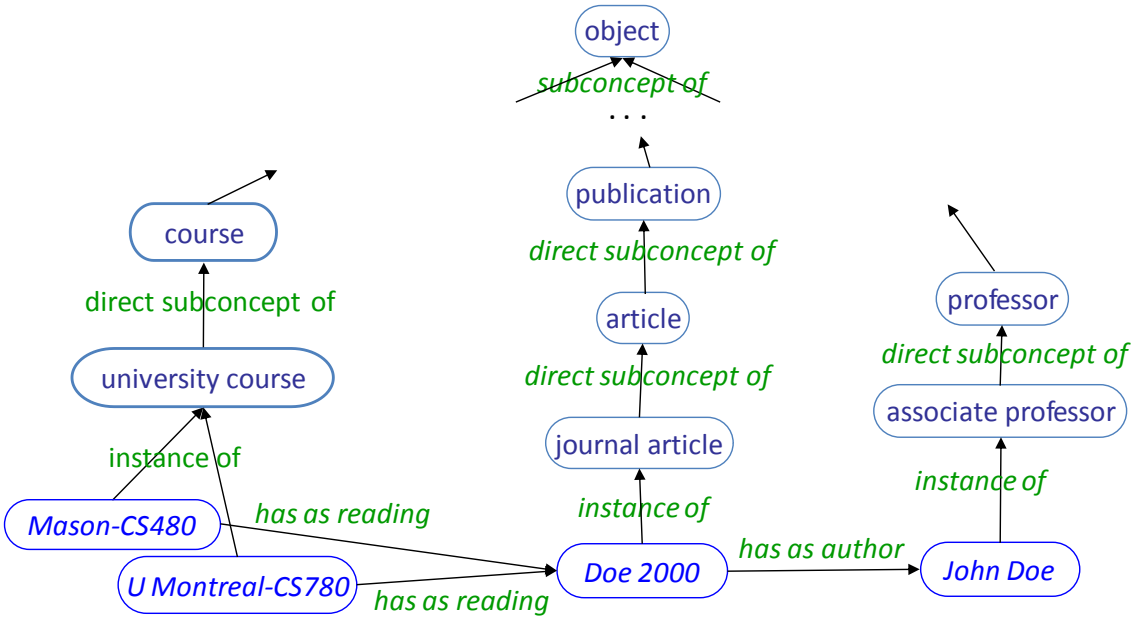

Figure 1. Fragment of an object ontology.

The object ontology plays a crucial role in Disciple and in cognitive assistants, in general, being at the basis of knowledge representation, user-agent communication, problem solving, knowledge acquisition and learning . First, the object ontology provides the basic representational constituents for all the elements of the knowledge base, such as the problems, the problem reduction rules, and the solution synthesis rules. It also allows the representation of partially learned knowledge, based on the plausible version space concept (Tecuci, 1998). Second, the agent's ontology enables the agent to communicate with the user and with other agents by declaring the terms that the agent understands. Consequently, the ontology enables knowledge sharing and reuse among agents that share a common vocabulary which they understand. Third, the problem solving methods of the agent are applied by matching them against the current state of the agent's world which is represented in the ontology. The use of partially learned knowledge (with plausible version spaces) in reasoning allows solving of problems with different degrees of confidence. And fourth, the object ontology represents the generalization hierarchy for learning, in which specific problem solving episodes are generalized into rules by replacing instances with concepts from the ontology.

2. Concepts and Instances

*A **concept** (or **class**) is a general representation of what is common to a set of **instances** (or **individuals**).* Therefore, it may be regarded as a *representation of that set of instances.* For example, professor in Figure 2 represents the set of all professors which include Amanda Rice and Dan Smith.

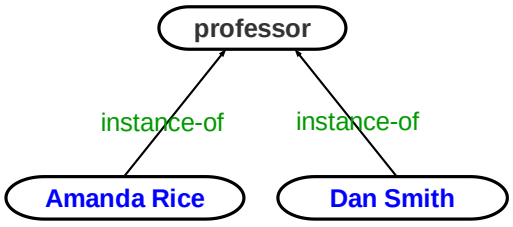

Figure 2. A concept and two of its instances.

*An **instance** (**individual**) is a representation of a particular entity in the application domain*, such as Amanda Rice.

We indicate that an instance belongs to a concept by using the relation **instance of**:

Amanda Rice instance of professor

3. Generalization Hierarchies

Generalization is a fundamental relation between concepts. *Intuitively, a concept P is said to be **more general than** (or **a generalization of**) another concept Q if the meaning of the representation of P is included into that of Q. Mathematically, P is said to be **more general than** Q if, and only if, the set of instances represented by P includes the set of instances represented by Q.*

Figure 3 shows several concepts with different degrees of generality. For example person is more general than student because a student must be a person or, in other words, the set of all persons includes the set of all students.

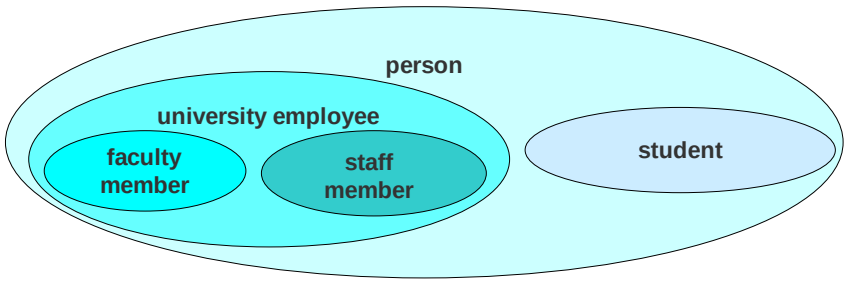

Figure 3. Concepts of different generality.

Let us notice that the above definition of generalization is extensional, based upon the instance sets of concepts. In order to show that P is more general than Q, this definition would require the computation of the (possibly infinite) sets of the instances of P and Q. Therefore, it is useful in practice only for showing that P is not more general than Q. Indeed, according to this definition, it is enough to find an instance of Q that is not an instance of P because this shows that the set represented by Q is not a subset of the set represented by P. Tecuci and Boicu (2008) provide an intentional definition of generalization that will allow one to compare the generality of the concepts.

One may express the generality relation between two concepts by using the relation **subconcept of**:

student subconcept of person

Other names used for expressing this type of relation are **subclass of** and **isa**.

A concept Q is a direct subconcept of *a concept P if an only if Q is a subconcept of P and there is no other concept R such that Q is a subconcept of R and R is a subconcept of P.*

One may represent the generality relations between the concepts in the form of a partially ordered graph that is usually called a generalization hierarchy (see Figure 4). The leaves of the hierarchy in Figure 4 are instances of the concepts which are represented by the upper level nodes. Notice that the instance John Doe is both an associate professor and a PhD advisor. Similarly, a concept may be a direct subconcept of different concepts.

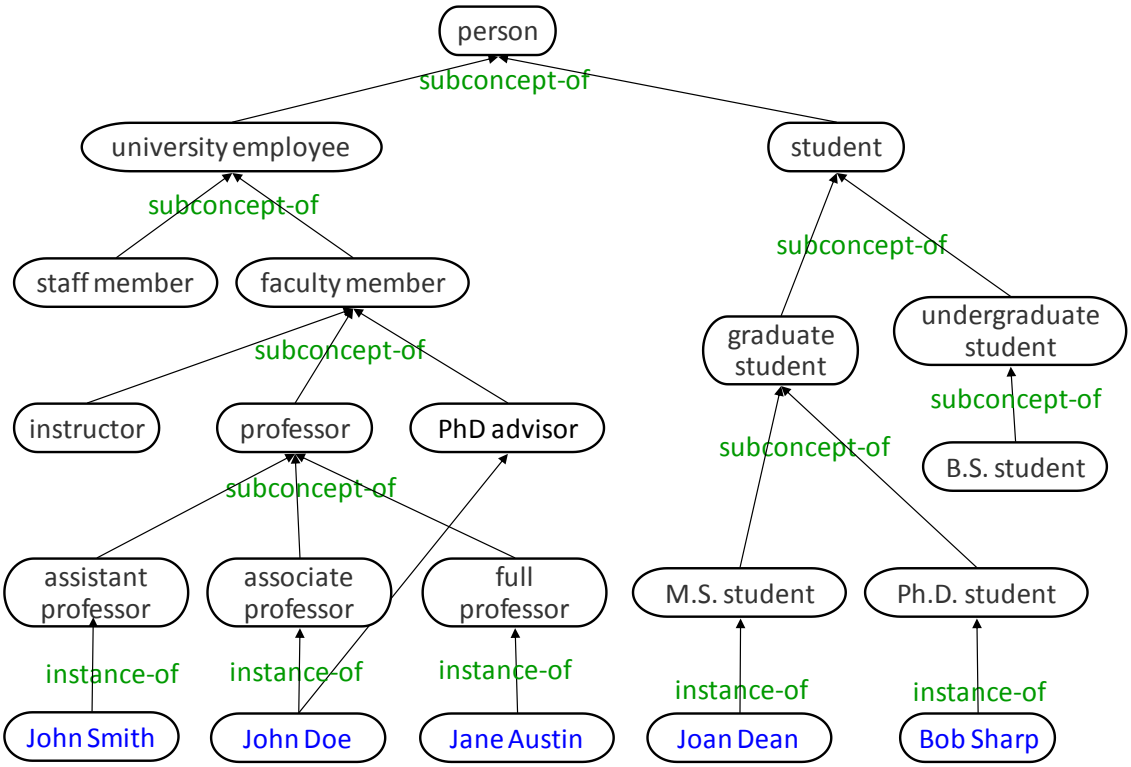

Figure 4.A generalization hierarchy.

4. Browsing a Generalization Hierarchy

Figure 5 shows the interface and the main functions of the Hierarchical Browser, an ontology tool that can be used to browse a hierarchy of concepts and instances.

The hierarchy in Figure 5 is rotated, with the most general concept (object) on the left hand side and its subconcepts on its right side. The hierarchy can be rotated by clicking on the "Rotate View" button. Clicking on the "Expand View" button leads to showing additional levels of the hierarchy, while clicking on the "Reduce View" button leads to showing fewer levels.

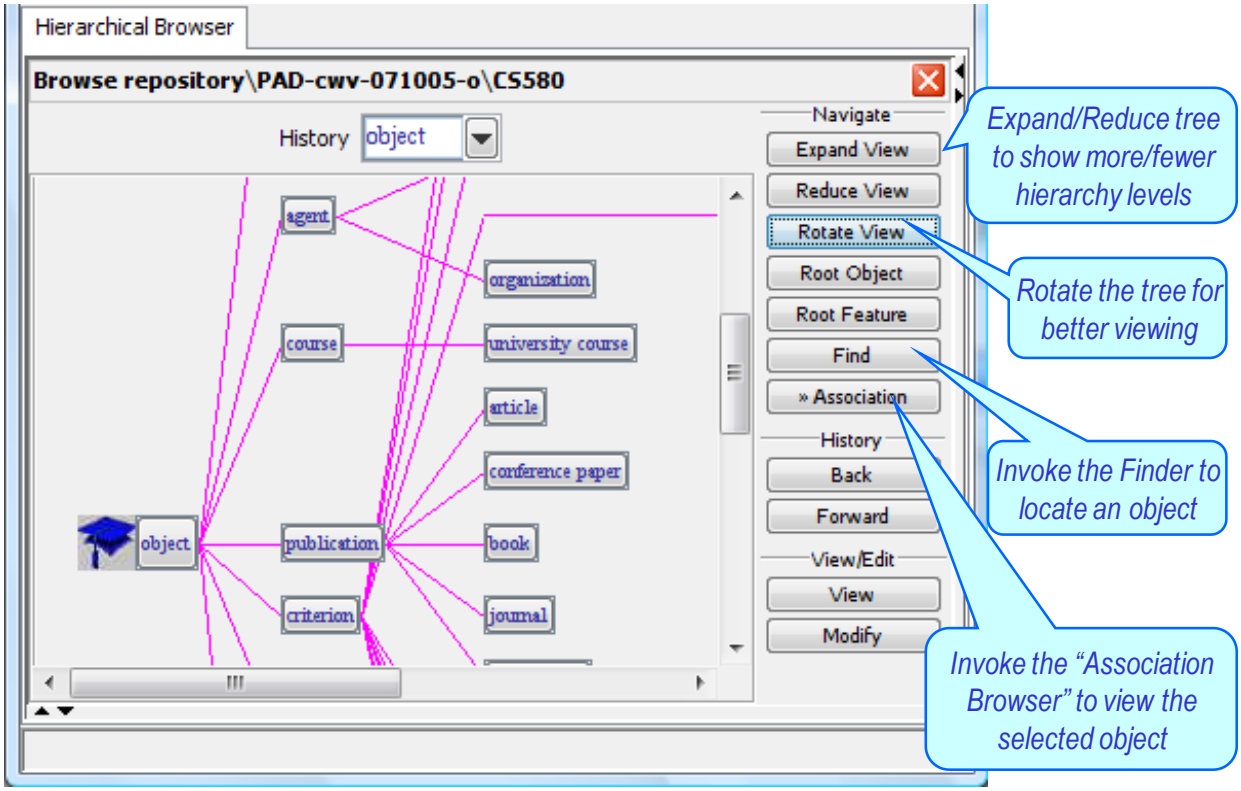

Figure 5.The interface and the main functions of the Hierarchical Browser.

Hands-on

Use the Hierarchical Browser to browse a generalization hierarchy.

5. Developing a Hierarchy of Concepts and Instances

Another tool for browsing a concept hierarchy is the Object Browser. Its interface and main functions are shown in Figure 6. This tool shows the hierarchy in a tree structure which could be expanded or collapsed by selecting a node (e.g. educational institution) and then by clicking on "Expand All" and "Collapse All", respectively. Similar effects may be obtained by clicking on the - and + nodes. Selecting a node, and then clicking on the "Hierarchical" button will open the Hierarchical Browser with that node as the top of the displayed hierarchy.

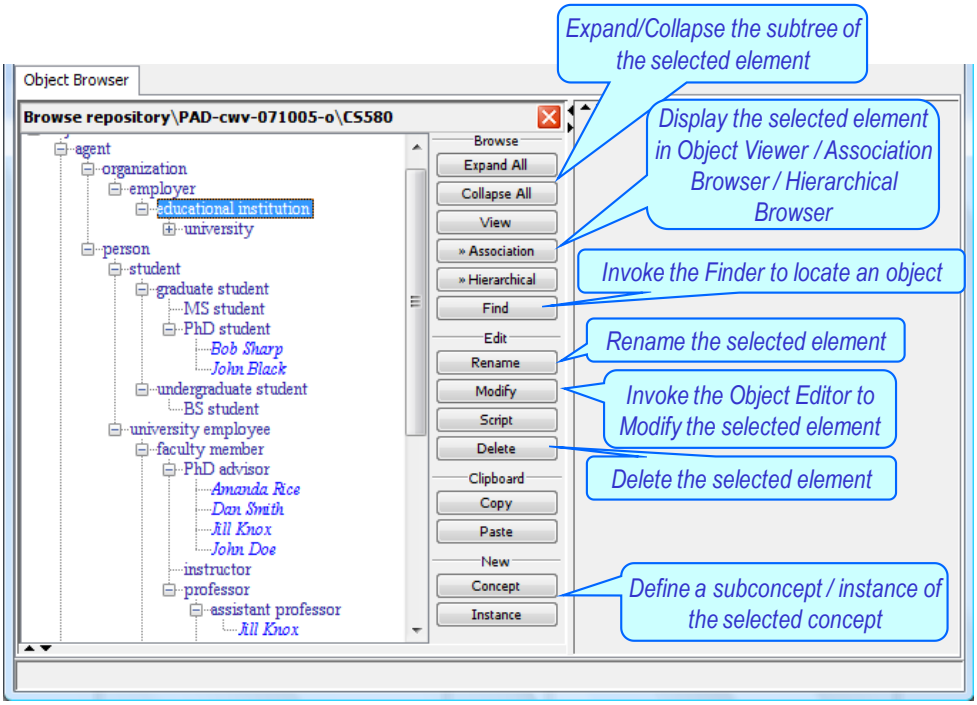

Figure 6.The Object Browser and its main functions.

The Object Browser can be used to develop a generalization hierarchy by defining concepts and instances, as described in Operation 1 and Operation 2.

Operation 1. Define a subconcept of a concept

- Open the Object Browser
- Click on the concept name to select it.
- Click on the "Concept" button.
- Write the name of the subconcept.
- Press the Enter key.

Operation 2. Define an instance of a concept

- Open the Object Browser
- Click on the instance name to select it.
- Click on the "Instance" button.
- Write the name of the instance.
- Press the Enter key.

In addition to defining concepts and instances, one should also be able to rename or delete them. These operations are performed as explain in Operation 3 and Operation 4, respectively. Deletion is a particularly complex operation. Disciple prevents it if it would lead to an inconsistent knowledge base because some of its elements refer to the element to be deleted.

Operation 3. Rename a concept or an instance

- Open the Object Browser

- Click on the concept or instance to select it.
- Click on the "Rename" button.
- Write the new name.
- Press the Enter key.

Operation 4. Delete a concept or an instance

- Open the Object Browser
- Click on the concept or instance to select it.
- Click on the "Delete" button.

Hands-on

Use the Object Browser to build a generalization hierarchy containing the following information:

course subconcept of object
operating systems course subconcept of course
artificial intelligence course subconcept of course
CS571 instance of operating systems course
CS580 instance of artificial intelligence course

6. Guidelines for Developing Generalization Hierarchies

Well-structured Hierarchies

Siblings in a generalization hierarchy are the concepts (or instances) that are direct subconcepts (instances) of the same concept. For example, assistant professor, associate professor and full professor in Figure 4 are siblings because they are all direct concepts of professor.

In a well-structured generalization hierarchy, all the siblings should have a comparable level of generality. In particular, they should be either all concepts or all instances. If some of them are instances, insert one or several concepts that include them.

The siblings should reflect concepts from the real world. However, if there are too many siblings, consider whether some of them may be grouped under another (natural) concept.

A case of a single sibling may be an indication of either an incomplete ontology or a modeling error.

Instance or Concept?

In general, a set of individuals is represented as a concept, while a single individual is represented as an instance. However, in certain applications, one may not wish to refine the representation of certain concepts, and represent them as instances. Consider the hierarchy from Figure 7. In that hierarchy, Artificial Intelligence Magazine is represented as an instance of a research journal. But one could have also represented it as a concept, the instances of which would have been specific issues of the Artificial Intelligence Magazine. Whether something is represented as an instance or as a concept influences how

Disciple learns. Therefore, *in many cases, learning considerations determine whether an entity is represented as an instance or as a concept.*

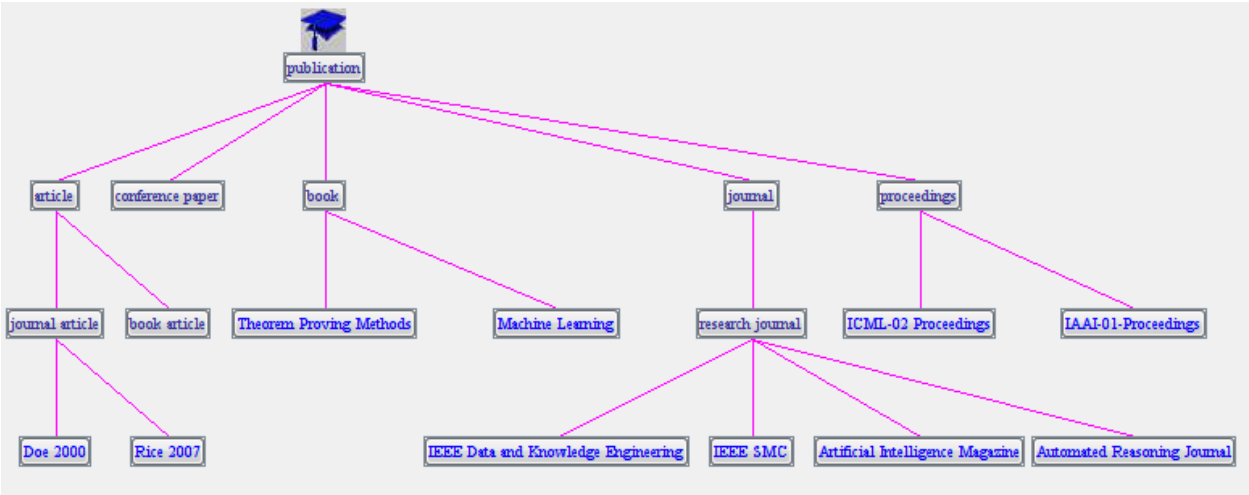

Figure 7.Representation as concepts or instances.

Naming Conventions

It is recommended to adopt a naming convention and to strictly adhere to it. This will both facilitate the understanding of the ontology and will help avoid modeling mistakes. In Disciple, it is recommended to use lower case letters for concept names. It is also recommended to consistently use either singular or plural, but not both, in naming the concepts.

Often, the names of the subconcepts of a concept include the name of the concept, as shown in several places in Figure 4 (e.g. student, graduate student, undergraduate student).

Automatic Support

Disciple will not accept ontology operations that will make the ontology inconsistent, such as introducing a circularity, where a concept is both a superconcept and a subconcept of another concept. Disciple will not accept definitions of concepts or instances that have the same names as previously defined ones.

7. Object Features

The objects in an application domain may be described in terms of their properties and their relationships with each other. For example, Figure 8 represents Mark White as a 46 years old associate professor employed by George Mason University. In general, the value of a feature may be a number, a string, an instance, a symbolic probability or interval (see Section 13), or a concept (see Section 16).

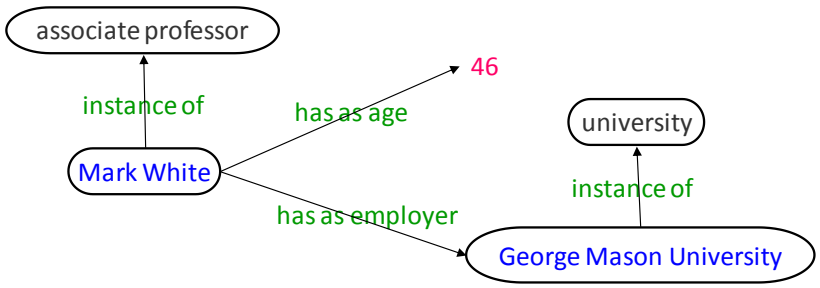

Figure 8.Sample object description.

8. Browsing the Objects and Their Features

Figure 9 shows the interface and the main functions of the Association Browser, which can be used to browse the objects and their features. This browser is centered on a given object (e.g. John Doe), showing its features (e.g. has as employer George Mason University), the features for which it is a value (e.g. Doe 2000 has as author John Doe), its direct concepts (e.g. PhD advisor and associate professor) and, in the case of a concept, its direct subconcepts or its direct instances.

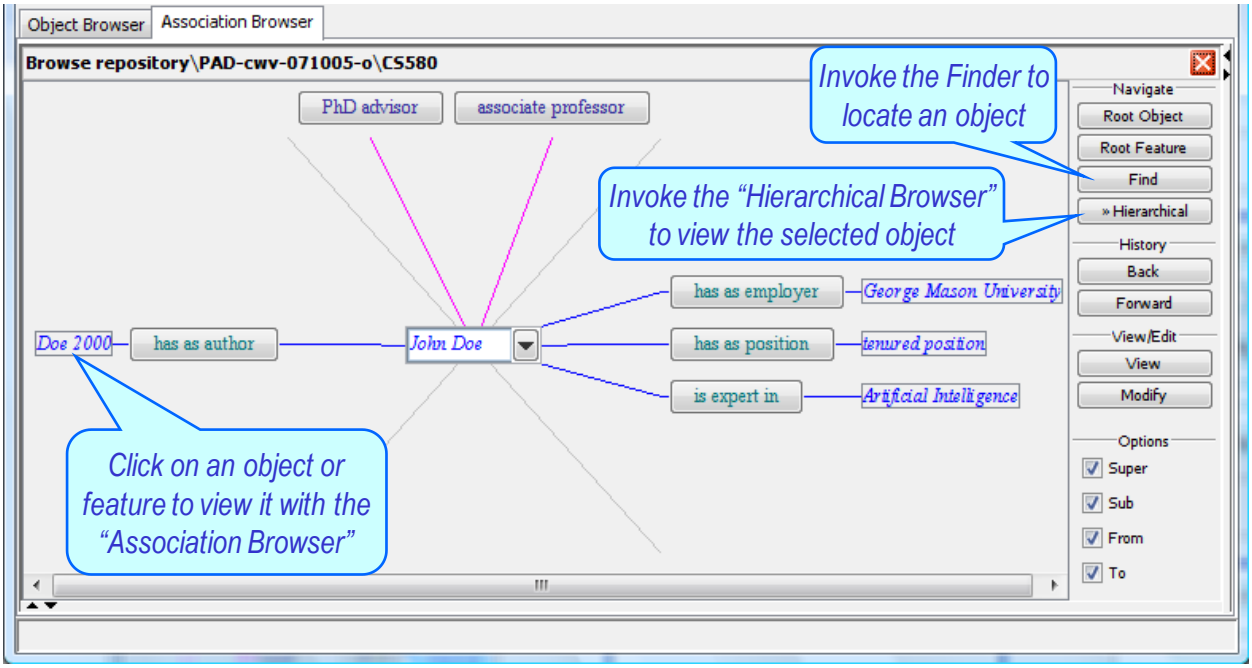

Figure 9.The Association Browser and its functions.

Clicking on any entity in the interface will center the Association Browser on that entity.

One may also browse the objects and their features by using the Object Browser and the Object Viewer, as illustrated in Figure 10 and described in Operation 5.

Operation 5. View an object

- Select the object in the Object Browser
- Click on the "View" button to view the object's features with the Object Viewer
- Click on the "Association" button to view the object's features with the Association Browser

Hands-on

Use the Association Browser, the Object Browser and the Object Viewer to browse the objects from the ontology and their features.

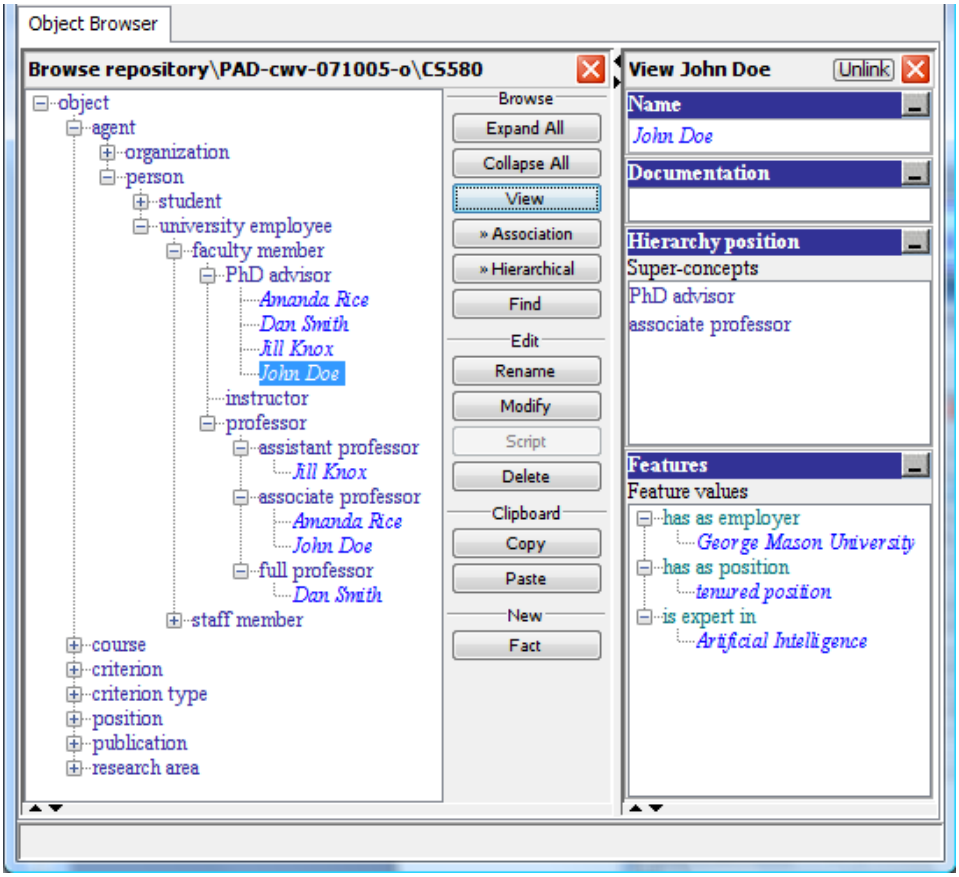

Figure 10. The Object Browser and the Object Viewer.

9. Defining Features

An object feature is itself characterized by several features which have to be specified when defining a new feature. They include its domain, range, superfeatures, subfeatures, and documentation.

*The **domain** of a feature is the concept that represents the set of objects that could have that feature.*

*The **range** is the set of possible values of the feature.*

For example, Figure 11 shows the representation of the has as employer feature. Its domain is person, which means that only individuals who are persons may have an employer. Its range is employer, meaning that any value of such a feature should be an employer.

There are several types of ranges that could be defined with Disciple: Concept, Number, Symbolic interval, Text and Any element.

We have already illustrated a range of type Concept (see Figure 11). A range of type "Number" could be either a set or an interval, and each of them could be either integer or real numbers. A range of type "Symbolic interval" is an ordered set of intervals, a data type discussed in Section 13. A range of type "Text" could be any string, a set of strings, or a natural language text. Finally, a range of type "Any element" could be any of the above entities.

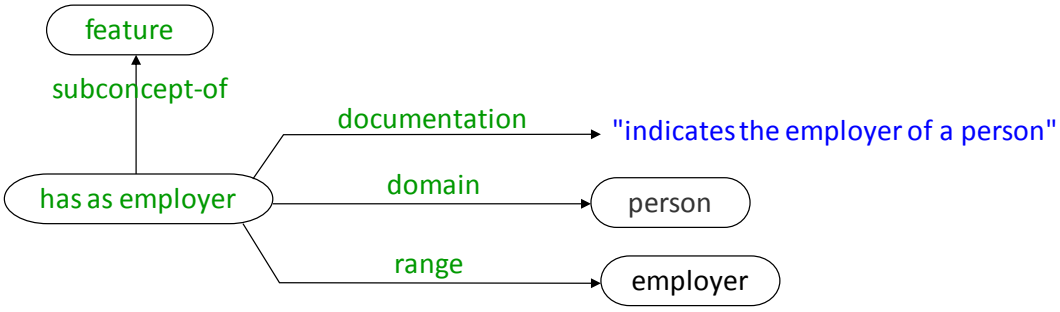

Figure 11. The representation of a feature.

Features are also organized in a generalization hierarchy, as illustrated in Figure 12.

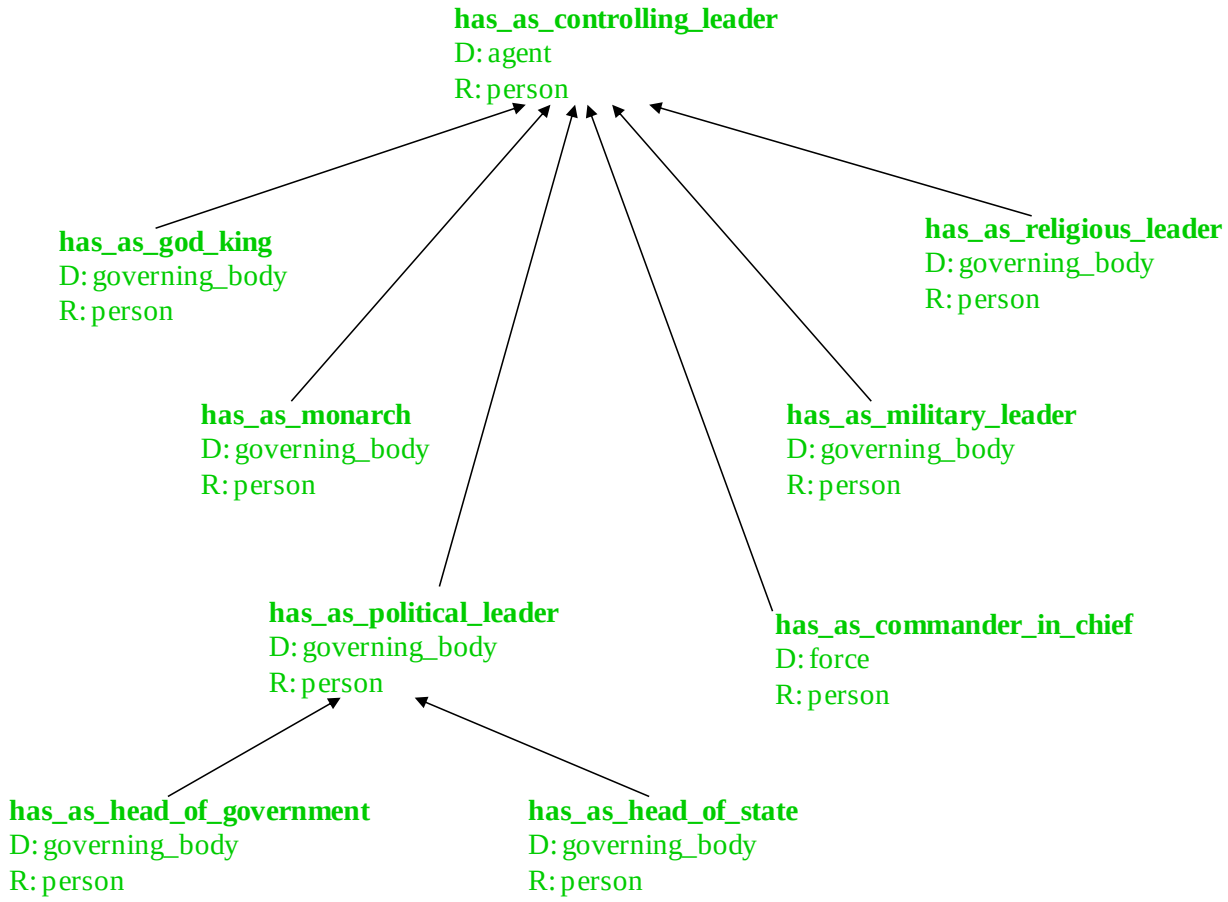

Figure 12. A generalization hierarchy of features.

One can browse the feature generalization hierarchy by using the Feature Browser, which is illustrated in the left-hand side of Figure 13. Its functions are similar to those of the Object Browser discussed in Section 5. To view the definition of a specific feature one may follow the steps from Operation 6.

Operation 6. View a feature definition

- Select the feature in the Feature Browser (e.g. has as author in the left hand side of Figure 13)
- Click on the "View" button to view the feature with the Feature Viewer (see the right hand side of Figure 13)
- Click on the "Association" button to view the feature with the Association Browser

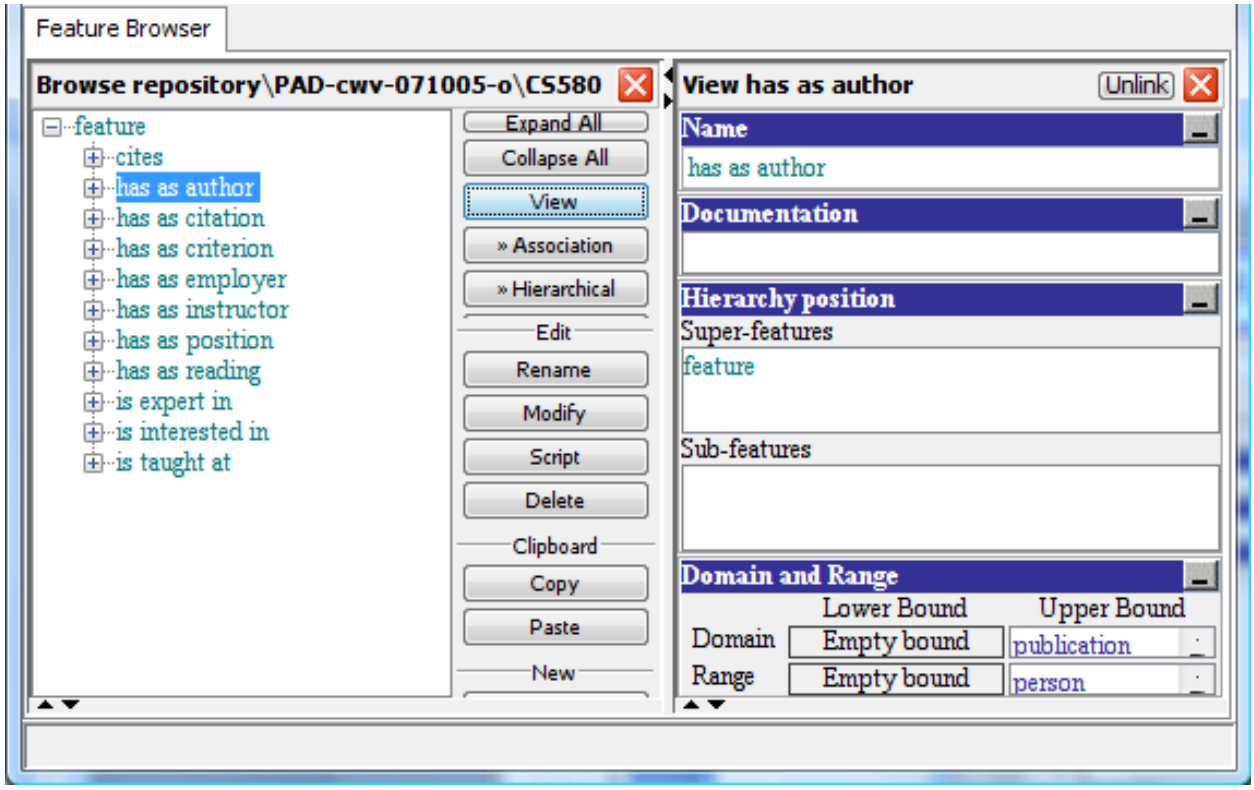

Figure 13. The Feature Browser and the Feature Viewer.

Hands-on

Browse the feature hierarchy by using the Feature Browser, the Feature Viewer, the Association Browser, and the Hierarchical Browser.

10. Developing a Hierarchy of Features

The features are defined and organized in a hierarchy by using the Feature Browser, similarly to how the Object Browser is used to develop a concept hierarchy. The steps needed to define a new feature (e.g. has as PhD advisor) are those from Operation 7.

Operation 7. Define a feature

- Open the Feature Browser
- Select the superfeature (e.g. feature) of the feature to be defined (i.e. has as PhD advisor)
- Click on the "Feature" button
- Specify the name of the feature to be defined (e.g. has as PhD advisor)
- Press "Enter"

When one defines a subfeature of a given feature, the domain and the range of the subfeature are set to be the ones of the superfeature. One can change them by clicking on the "Modify" button of the Feature Browser. This will invoke the Feature Editor, which is illustrated in Figure 14. Using the Feature

Editor, one can add or delete superfeatures or subfeatures of the selected feature, in addition to modifying its domain and range.

Figure 15 illustrates the process of changing the domain of the has as PhD advisor feature, from object to person. This process consists of the steps from Operation 8.

Operation 8. Change the domain of a feature

- Open the Feature Browser
- Select the feature with the domain to be changed (e.g. has as PhD advisor)
- Click on the "Modify" button to invoke the Feature Editor
- Select the "<P1>[domain]" tab in the Feature Editor
- Click on "Add" to open an Object Browser pane
- Browse and Select a concept for the new domain (e.g. person)
- Click on "Add as domain"
- Click on "Apply domain" to commit the addition in the ontology.

Operation 9. Change the range of a feature

- A range of type "Concept" is modified in the same way as a domain (see Operation 8)
- For range types other than "Concept", a type-specific editor is invoked after clicking on the "Add" button

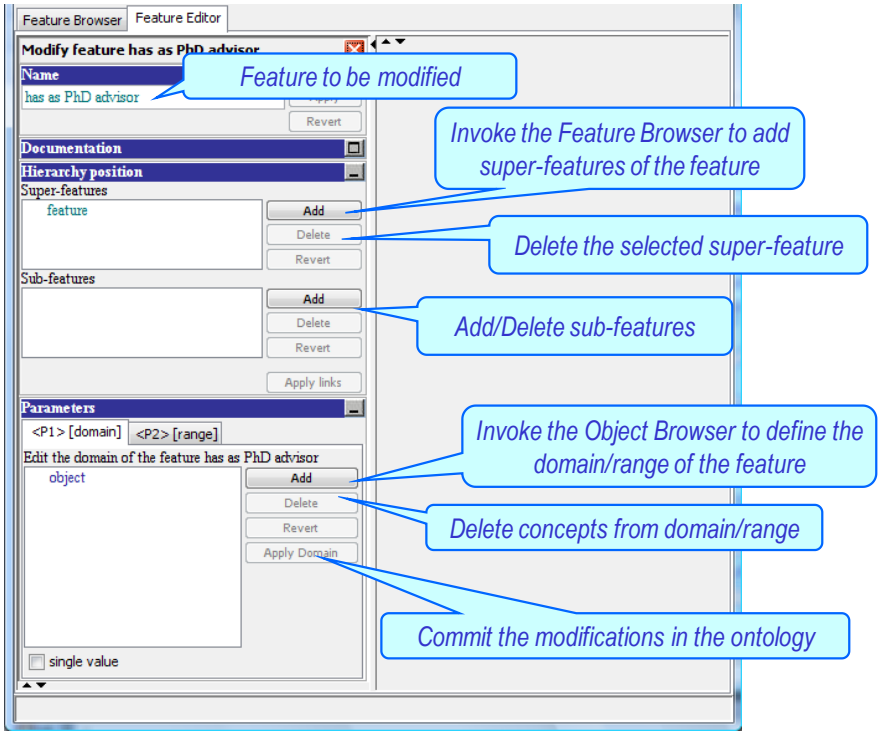

Figure 14. The Feature Editor and its main functions.

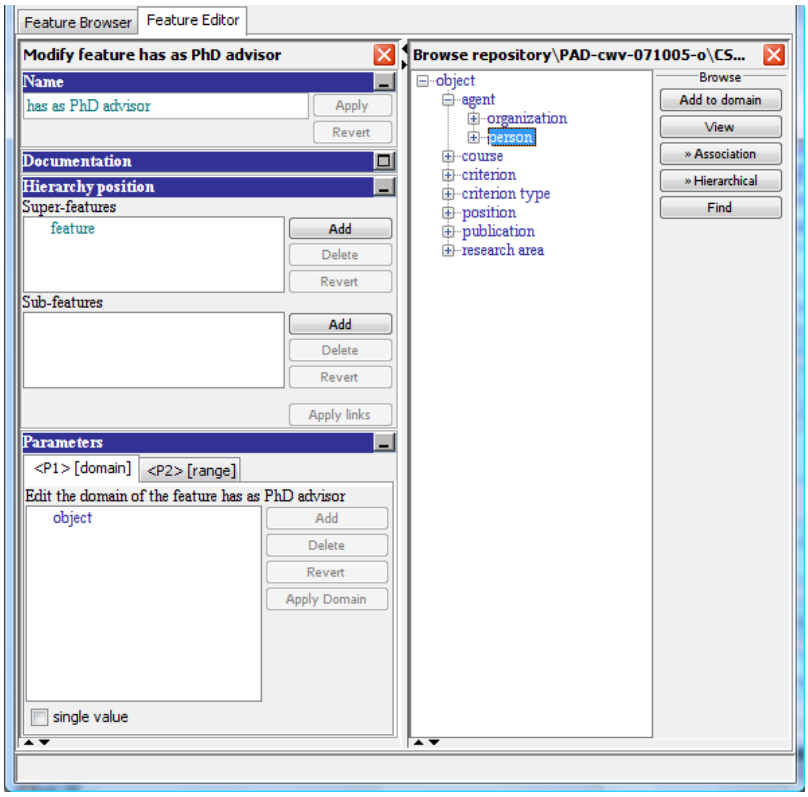

Figure 15. Changing the domain of a feature.

Hands-on

Use the Feature Browser and the Feature Editor to represent the feature hierarchy from Figure 12.

11. Defining Instances and Their Features

With some concepts and features defined, one may use the Object Editor to define instances of these concepts (as discussed in Section 5), and to associate features with them.

Figure 16 illustrates the process of associating the is interested in feature with John Doe. The steps of this process are those from Operation 10.

Operation 10. Define the feature of an object

- Open the Object Browser and select the object (e.g. John Doe)
- Click on the "Modify" button to open the Object Editor
- Click on "Add feature" in the Object Editor to open a Feature Browser pane
- Browse and Select the feature to be added to the object (e.g. is interested in)
- Click on the "Define for object" button
- Click on the "Add feature" button to add the feature in the Feature pane

- The Finder is automatically opened to locate the value of the feature in the knowledge base (see Figure 17)
- Write a part of the name of the value in the "Find" pane
- Click on the "Find" button
- Click on the correct value from the list returned by the Finder
- Click on the "Select" button
- If the range of the feature is a symbolic interval or a set of values, a selection pane is opened to choose the right value of the feature. For a number, one has to simply type it.

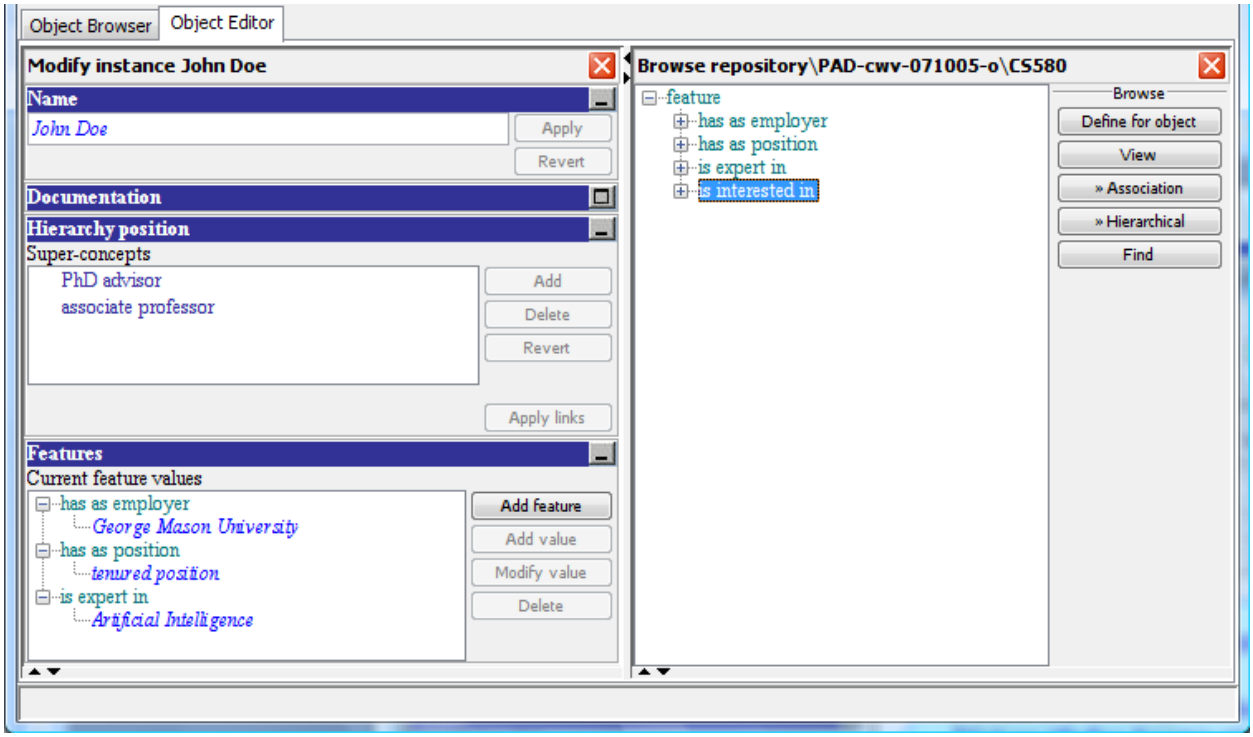

Figure 16. Defining the feature of an instance.

The Object Editor can also be used to update the list of the direct superconcepts for an object (instance or concept) and the list of the direct subconcepts or instances of a concept. The actual steps to perform are presented in Operation 11 and Operation 12. As with all the ontology operations, Disciple will not perform them if they would lead to inconsistent ontology (for example a cycle along the subconcept of relation).

Operation 11. Add a direct superconcept to an object

- Locate the object (e.g. John Doe) with the Object Browser and the Finder and click on the "Modify" button to open the Object Editor (see Figure 17)
- Click on "Add" in the "Super-concepts" pane to open an Object Browser pane
- Browse and Select the super-concept
- Click on "Add as parent"
- Click on "Apply links" to commit the addition in the ontology

Operation 12. Add a direct subconcept or an instance to a concept

- Locate the concept with the Object Browser and the Finder and click on the "Modify" button to open the Object Editor
- Click on "Add" in the "Sub-concepts and instances" pane to open an Object Browser pane
- Browse and Select the sub-concept or the instance to be added
- Click on "Add as child"
- Click on "Apply links" to commit the addition in the ontology

Hands-on

Use the Ontology tools of Disciple to develop an object ontology that represents the following information:

The color of Apple1 is red.
The color of Apple2 is green.
Apple1 is an apple.
Apple2 is an apple.
Apples are fruits.

Hint: Define object concepts, object features and instances.

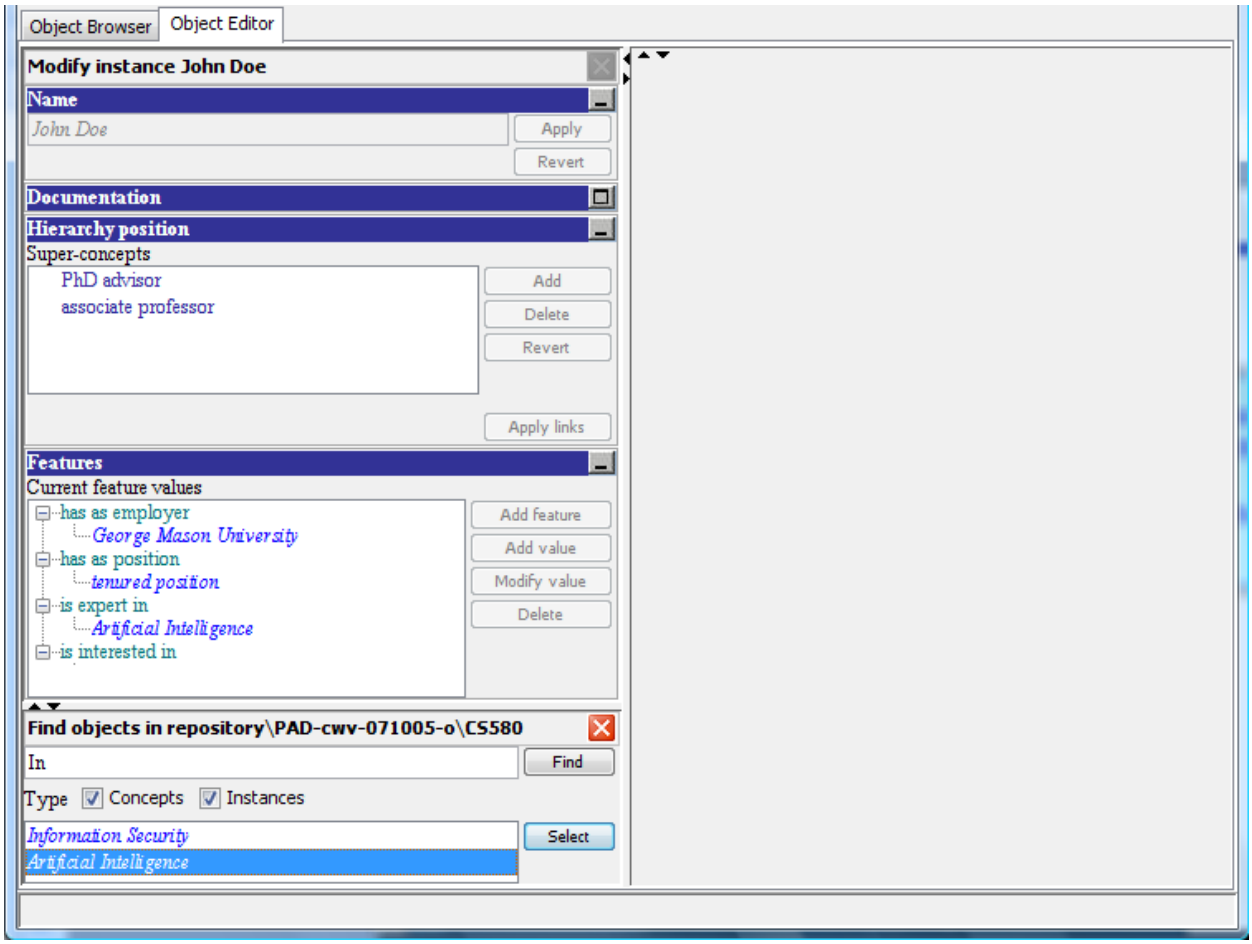

Figure 17. Defining the value of a feature with the Finder.

12. Guidelines for Defining Features and Values

Concept or Feature?

Almost any distinction from the real world may be represented, either as a concept or as a feature. While there are no absolute rules for this difficult modeling decision, several guidelines are useful to follow.

Represent well-established categories from the real-world as concepts.

For example, two alternative ways of representing the fact that Bob Evens is a PhD student are:

Bob Evens instance of PhD student
Bob Evens has as student level PhD

The first one is preferable because PhD student is a well-established category.

A feature in an ontology can naturally represent a binary relation. However, if the relation is not binary, one needs to define a concept or an instance with which one can associate any number of features. Let us consider the fact that John Doe has written "Windows of Opportunities". This can be represented as:

John Doe has as writing "Windows of Opportunities"

But what if we want to represent the additional knowledge that he has written it from 2005 until 2007? We can no longer associate this information with the has as writing feature. The solution is to define the concept writing and its instance Writing 1, instead of the feature has as writing, as illustrated in Figure 18.

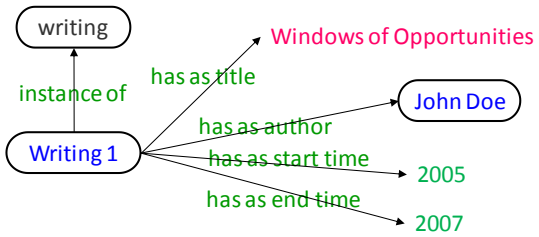

Figure 18. Ontology fragment representing "writing" as a concept.

Concept, Instance or Property Value?

Another modeling issue is to decide whether to represent the value of a feature as a constant (e.g. a string), an instance or a concept.

If one needs to associate additional properties with a value, then it cannot be represented as a constant.

Let us consider again the representation from Figure 18. If we want to represent any additional features of Windows of Opportunities (for instance, that it has 258 pages), then we have to represent it as an instance (which will have 258 pages as a property value).

If one anticipates learning concepts that will be characterized by various subsets of values, then one may wish to define a hierarchy of instances and concepts with the possible values.

For example, the colors could be defined as strings, such as white, yellow, orange. However, if concepts such as warm color or cold color are important in the application domain, then one may wish to define a hierarchy of colors.

The same considerations apply to numbers. While generally they are represented as values, one may wish to define an ordered set of intervals, as discussed in Section 13. In this case each interval will have a specific name (such as toddler, child, or adult) which can be used as property value.

Naming of Features

It is useful to define names that allow an easy distinction between a concept and a feature. Two common practices are to add the "has-as" prefix or the "of" suffix to feature names, such as has as author or author of.

Automatic Support

The ontology tools have several features that facilitate their operations. For example, when modifying the domain or the range of a feature (see the right hand side of Figure 15), Disciple will only display the concepts that are acceptable values.

When defining a new feature for a given concept or an instance (see the right hand side of Figure 16), Disciple will only display the features the domain of which include the given concept or instance.

When defining the value of a given feature (see Figure 17), the Finder will only display those values that are in the range of the feature.

13. Ordered Sets of Intervals

In many domains it is useful to define ordered sets of intervals and to associate symbolic names with them. For example, one may consider the interval of human ages as [0.0, 150.0] and partition it into successive intervals corresponding to specific ages, such as infant [0.0, 1.0), toddler [1.0, 4.5), etc., as shown in the right hand side of Figure 19. Then one can use these symbolic names as feature values:

Alice Brown has as age teen

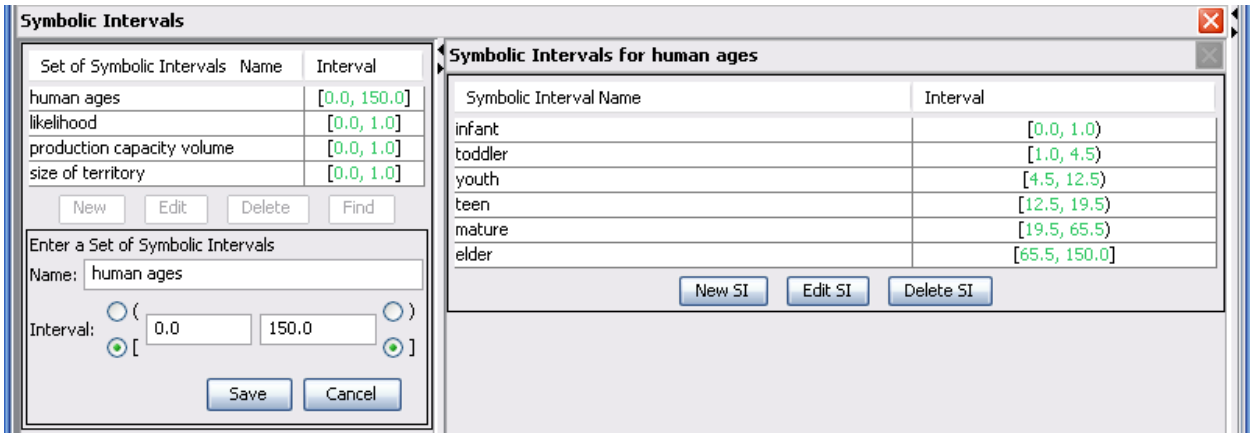

Figure 19. Sample sets of symbolic intervals.

As will be discussed in Chapter x, this representation will facilitate learning through the generalization of a set of symbolic values into a larger interval.

The ordered set of intervals may also be used to represent symbolic probabilities. For example, one may define likelihood as corresponding to the probability interval [0.0, 1.0]. This probability interval may be partitioned into subintervals, such as, no evidence [0.0, 0.0], a remote possibility (0.0, 0.2), unlikely [0.2, 0.4), an even chance [0.4, 0.6], likely (0.6, 0.8], almost certain (0.8, 1.0), and certain [1.0, 1.0]. Then one can express the fact that the likelihood that John Smith will get tenure is between 0.8 and 1 by:

John Smith likelihood of getting tenure almost certain

One may use the Ordered Set of Intervals Browser (illustrated in Figure 19) to define such entities, as shown in Operation 13.

Operation 13. Define an ordered set of intervals

- Open the Ordered Set of Interval Browser
- Click on the "New" button to create a new set of symbolic intervals (see Figure 19)
- Provide a name for the newly created set (e.g. human ages)
- Provide a base interval (e.g. [0.0, 150.0]) that will be partitioned by its component symbolic intervals
- Click on "Save" to save the information into the knowledge base
- A single default symbolic interval is created for the entire base interval (i.e. human ages subinterval 1 for [0.0, 150.0])
- Click on "Edit SI" to modify the currently selected symbolic interval
- Change the generated generic name (e.g. to infant) and the corresponding symbolic interval (e.g. to [0.0, 1.0))
- Click on "Save" to save the symbolic interval in the set
- Click on the "New SI" button to create the next symbolic interval
- Write the name (e.g. toddler) and the numeric range (e.g. [1, 4.5)) for the current interval
- Click on "Save"
- Repeat these operations to define all the intervals in the set
- Click on the "Save" button (in the bottom left side of Figure 19) to save the newly defined set of symbolic intervals into the knowledge base

Disciple will only save a set of symbolic intervals that is a correct partition of the total interval (i.e. the symbolic intervals are disjoint and their union is the same with the total interval).

Hands-on

Define a set of symbolic intervals, a new feature having as range that set, and an instance having that feature.

14. Transitivity

The **instance of** and **subconcept of** relations are transitive:

$\forall X, \forall Y, \forall Z$, [(X subconcept of Y) $\wedge$ (Y subconcept of Z) $\rightarrow$ (X subconcept of Z)]

$\forall X, \forall Y, \forall Z$, [(X instance of Y) $\wedge$ (Y subconcept of Z) $\rightarrow$ (X instance of Z)]

Let us consider the hierarchy fragment from the middle of Figure 20. By applying the transitivity of instance of and subconcept of one may infer that:

Joan Dean instance of person
M.S. student subconcept of person

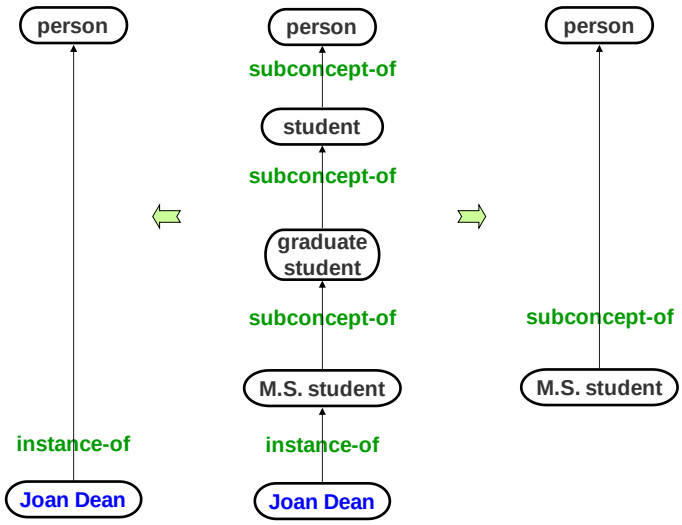

Figure 20. Use of the transitivity of instance of and subconcept of.

15. Inheritance

A theorem which is implicitly represented in an ontology is the inheritance of features from a more general concept to a less general concept or an instance.

An instance inherits the properties of the concepts to which it belongs:

$$\forall X,\forall Y,\forall Z,\ [(X \text{ instance of } Y) \wedge (Y \text{ feature } Z) \rightarrow (X \text{ feature } Z)]$$

Similarly, a concept inherits the properties of its superconcepts:

$$\forall X,\forall Y,\forall Z,\ [(X \text{ subconcept of } Y) \wedge (Y \text{ feature } Z) \rightarrow (X \text{ feature } Z)]$$

For example, in the case of the ontology in Figure 21 one can infer:

professor retirement age 66
assistant professor retirement age 66
John Smith retirement age 66

by inheriting the retirement age property from

faculty member retirement age 66

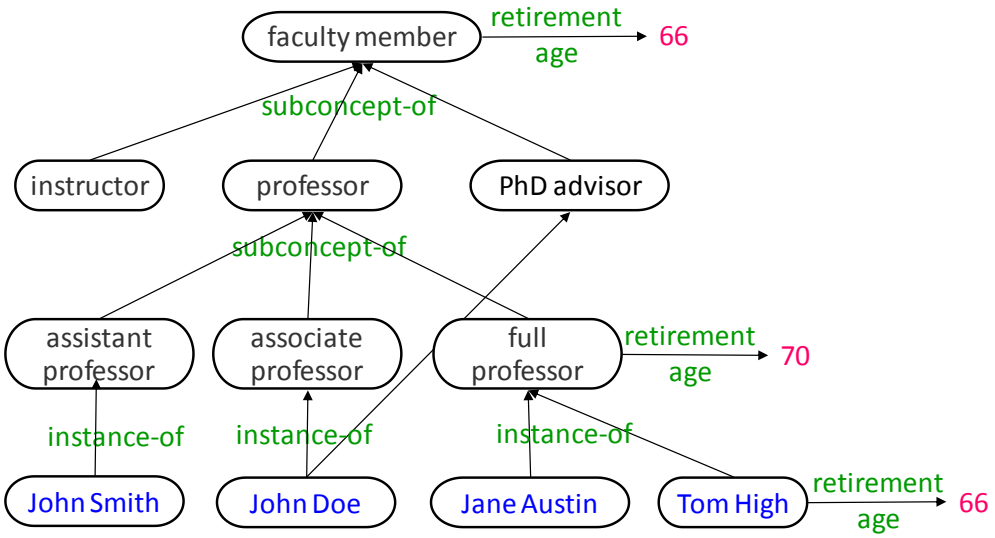

Figure 21. An illustration of default inheritance.

The inheritance of properties is one of the most important strengths of an ontology, allowing a compact and economical representation of knowledge. Indeed, if all the instances of a concept have the same property P, with the same value, then it is enough to associate the property with the concept because it will be inherited by each of the concept's instances. There are, however, two special cases of inheritance that one should pay special attention to. They are discussed below.

Default inheritance

There are some domains of knowledge in which exceptions to general rules exist. For example, it is generally useful to assume that all birds can fly. Certain birds, such as the ostrich and the kiwi, however, cannot fly. In such a case, it is reasonable to use a representation scheme in which properties associated with concepts in a hierarchy are assumed to be true for all subconcepts and instances, unless specifically overridden by a denial or modification associated with the subconcept or the instance.

Let us consider again the example in Figure 21. The fact that the retirement age of an assistant professor is 66 is inherited from faculty member. On the other hand, a full professor does not inherit this property from faculty member because it is explicitly represented that the retirement age of a full professor is 70. This overrides the default which is further up in the tree. Therefore, to find a feature of some object, the agent will first check whether the feature is explicitly associated with the object and take the corresponding value. Only if the feature is not explicitly associated with the object will the agent try to inherit it from the superconcepts of the object, by climbing the generalization hierarchy.

What is the retirement age of Jane Austin? Because this property is not explicitly associated with Jane Austin, the agent will attempt to inherit it from one of her ancestors in the generalization hierarchy. It will check these ancestors from bottom-up and will use the first value found, which is 70. Therefore, the retirement age will be inherited from full professor and not from faculty member.

Tom High, however, decided to retire at 66. Therefore this value is explicitly associated with him and will not be inherited.

Multiple inheritances

It is possible for a concept or an instance to have more than one direct superconcept. For example, in the ontology from Figure 21, John Doe is both an associate professor and a PhD advisor. Therefore, John Doe will inherit features from both of them and there is a potential for inheriting conflicting values. In such a case, the agent should use some strategy in selecting one of the values. A better solution, however, is to detect such conflicts when the ontology is built or updated, and to directly associate the correct feature value with each element that would otherwise inherit conflicting values.

16. Concepts as Feature Values

The previous section has discussed how the features of the concepts are inherited. However, in all the examples given, the value of the feature was a number. The same procedure will work if the value is an instance or a string. But what happens if the value is a concept, which has itself a set of instances, as shown in Figure 22?

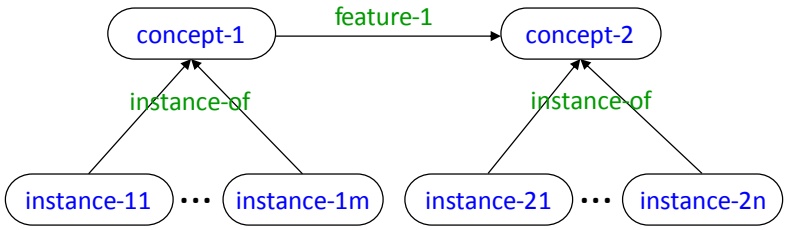

Figure 22. Relationship between two concepts with instances.

In this case, each instance of concept-1 has feature-1, the value of which is the set of all the instances of concept-2:

instance-11 feature-1 instance-21
...
instance-11 feature-1 instance-2n
...
instance-1m feature-1 instance-2n

One has to exercise care when defining features between concepts. For example, the correct way to express the fact that a parent has a child is to define the following feature:

has-as-child
domain: parent
range: child

On the contrary, the expression, "parent has-as-child child" means that each parent is the parent of each child.

17. Ontology Maintenance

Maintaining the consistency of the object ontology is a complex knowledge engineering activity because the object and the feature definitions interact in complex ways. For example, deleting an object concept requires the updating of all the knowledge base elements that refer to it, such as any feature that contains it in its range or domain, or any concept that inherits its features.

Figure 23 illustrates one potential consequence of deleting a subconcept of relation. Let us consider that in the initial state of the object ontology, the domain of the feature f is the concept A. Let us further consider that C has the feature f with value 7. This is consistent with the definition of f because C is a subconcept of B which is a subconcept of A. Therefore C is a subconcept of A and is in the domain of f.

Let us now delete the relation "B subconcept of A", that is, B is no longer a subconcept of A. In the new object ontology C can no longer have the feature f because it is no longer in the domain of f. The modification done to B generated an error in another part of the knowledge base (at C).

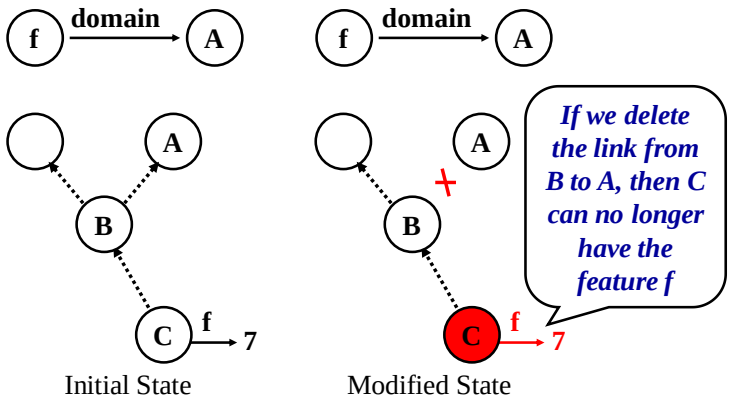

Figure 23. Complexity of ontology maintenance.

18. Steps in Ontology Development

Table 1 defines the ontology development steps which are also illustrated in Figure 24.

Table 1. Ontology development steps

1. Define basic concepts (types of objects) and their organization into a hierarchical structure (the generalization hierarchy).
2. Define generic object features by using the previously defined concepts to specify their domains and ranges.
3. Define instances (specific objects) by using the previously defined concepts and features.
4. Extend the object ontology with new concepts, features, and instances.
5. Repeat the above steps until the ontology is judged to be complete enough.

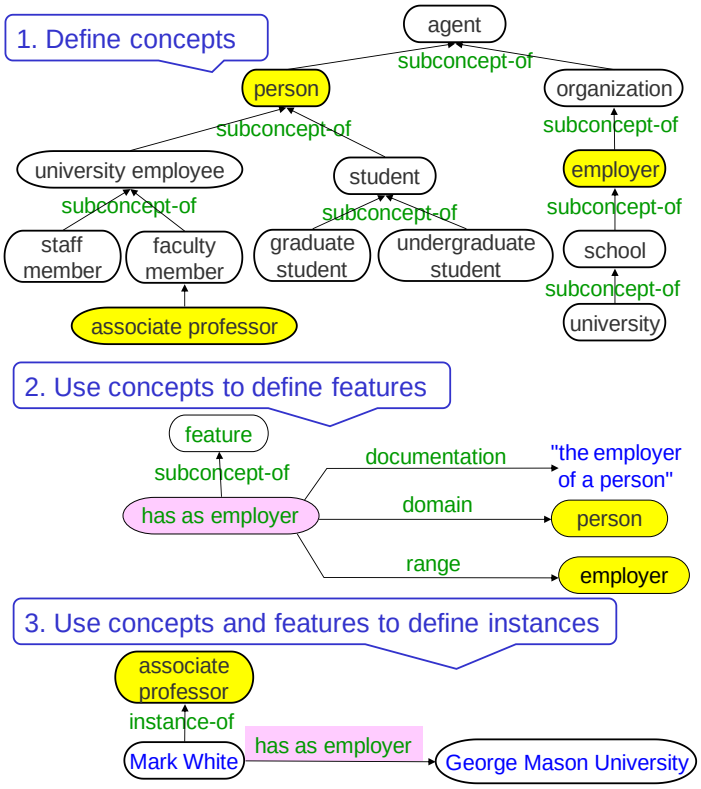

Figure 24. Steps in ontology development.

First one needs to define basic concepts and to organize them into a hierarchical structure. This may be performed by using the Object Browser of Disciple, as discussed in Section 5. At this step one may also define some of the instances.

Once a set of basic concepts have been defined, one can define features that use these concepts as their domains and ranges. For example, one may define the feature has as employer with the domain person and range employer which are previously defined concepts. The features are defined and organized in a hierarchy by using the Feature Browser and the Feature Editor, as discussed in Section 10.

With some of concepts and features defined, one may define instances of these concepts and associate features with them, as discussed in Section 11. For example, one can define Mark White as an instance of associate professor, and specify its feature has as employer with the value George Mason University.

Ontology development is an iterative process, as indicated by the last step in Table 1.

19. Ontology Development Methodology

Ontology design is a creative process and there is no single correct ontology. However, one may follow the guidelines described in Sections 6 and 12 that will help design a good ontology. This development goes through several stages which are described in the following.

Ontology Specification

The first step is to determine the scope of the ontology by specifying its main concepts, features and instances. One general approach to ontology specification is to formulate a set of competency questions that the envisioned knowledge-based agent should be able to answer (Gruninger and Fox 1995). Based on these questions, one identifies some of the concepts, features and instances that should be part of the ontology.

However, Disciple provides a much more precise guidance to ontology specification. As will be presented in detail in Section x, the subject matter expert and the knowledge engineer define a set of typical problems that the envisioned agent has to be able to solve. Then they actually solve these problems the way they would like Disciple to solve them. This process identifies very clearly what concepts and features should be present in the ontology to enable the agent to solve those types of problems.

Ontology Reuse

Significant reusable ontologies have been developed as part of the well-known projects Cyc (Cyc, 2008), *WordNet* (Fellbaum, 1998), SUMO (Niles and Pease, 2001), UMLS (Humphreys and Lindberg, 1993), Protégé (Protégé 2000; Noy and McGuinness, 2001) and Ontolingua (Farquhar, 1997). Additionally, are many libraries of reusable ontologies, including “Ontologies by Keyword” (http://www.daml.org/ontologies/keyword.html), the Ontolingua ontology library (http://www.ksl.stanford.edu/software/ontolingua/), the DAML ontology library (http://www.daml.org/ontologies/), UNSPSC (www.unspsc.org), RosettaNet (www.rosettanet.org), DMOZ (www.dmoz.org)), the SNOMED structured vocabulary for Medicine (Price and Spackman 2000), the Unified Medical Language System (Humphreys and Lindberg 1993).

Because ontology design and development is a complex process, it make sense to import relevant concepts and feature from such ontologies (guided by the previously developed specification), rather than defining them from scratch. In particular, one may wish to look for general purpose ontologies (such as an ontology of time, of space, or of units of measures), if they are necessary to the agent under development.

Ontology Development

With the most relevant concepts identified, one may actually develop the ontology by using the tools presented in this chapter. However, ontology development is an iterative process. Therefore additional concepts, features and instances will be added while teaching the agent to solve problems.

An important aspect to emphasize is that the ontology will always be incomplete. Moreover, one should not attempt to represent everything in the ontology. On the contrary, the ontology is intended to represent only the terms of the representation language which are used in the definitions of problems and rules. Much of the most complex knowledge will be represented into the rules.

20. Exercises

1. What are the possible relationships between two concepts A and B, from a generalization point of view? Provide examples of concepts A and B in each of the possible relationships.

2. How could one prove that A is more general than B? Is the proposed procedure practical?

3. How can one prove that A is not more general than B? Is the proposed procedure practical?

4. Consider the feature hierarchy from Figure 25. Indicate the necessary relationship between: a) BD1 and 1D1; b) BR1 and 1R1; c) A2D1 and 1D1; d) AD1 and BD1; e) 1D1 and 1R1.

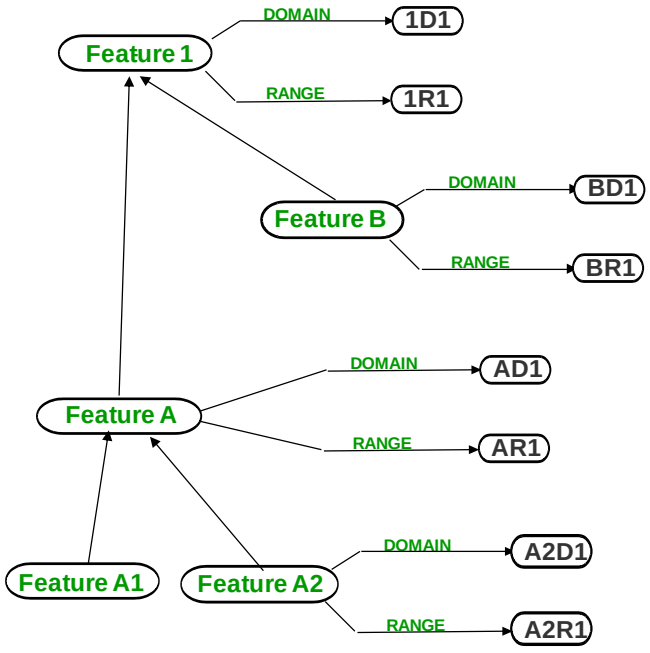

Figure 25. Sample feature hierarchy.

5. Develop an object ontology that represents the following information:

Puss is a calico.
Herb is a tuna.
Charlie is a tuna.
All tunas are fishes.
All calicos are cats.
All cats like to eat all kinds of fish.
Cats and fishes are animals.

Hint: You should define object concepts, object features and instances.

6. Develop an object ontology that represents the following information:

Basketball players are tall.
Muresan is a basketball player.
Muresan is tall.

Hint: Define object concepts, object features and instances.

7. Develop an object ontology that represents the following information:

Birds are animals.
Birds have feathers, fly and lay eggs.
Albatros is a bird.
Donald is a bird.
Tracy is an albatros.

Hint: Define object concepts, object features and instances.

8. Insert the additional knowledge that platypus lays eggs into the object ontology from Figure 26. Explain the result.

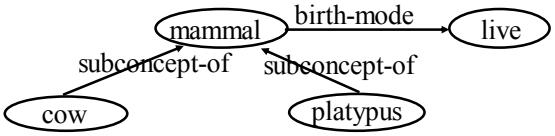

Figure 26. Ontology fragment.

9. Explain in English what information is represented in Figure 27.

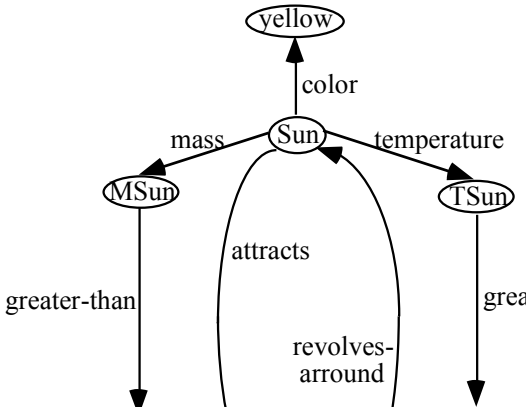

Figure 27. Knowledge representation.

10. Explain in English what information is represented in Figure 28. How could one encode the additional information that Clyde owned nest-1 from Spring-90 to Fall-90?

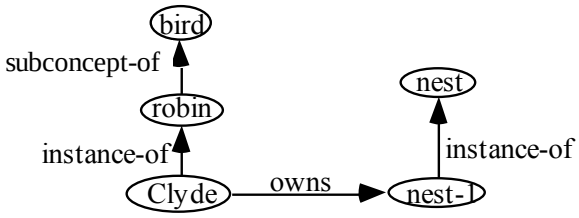

Figure 28. Representation with binary predicates.

21. Solutions

1. There are three possible cases: a) A is more general than B; b) B is more general than A; c) There is no generalization relationship between A and B.

2. To prove that A is more general than B it is enough to prove that all the instances of B are also instances of A. This is not a practical procedure if A and B have an infinite number of instances.

3. To prove that A is not more general than B it is enough to prove that B contains an instance which is not an instance of A. This is a practical procedure even for concepts with an infinite number of instances.

4. If the feature A is more general than the feature B, then the domain of A should be more general than the domain of B. Similarly, the range of A should be more general than the range of B.

10. A binary relation (feature) cannot encode the additional information that Clyde owned nest-1 from Spring-90 to Fall-90. To represent this additional information one would need to use a predicate with four arguments, such as ownership(owner, ownee, start-time, end-time). We need to represent ownership as a node (concept) rather than a binary relation (feature), as indicated in Figure 29.

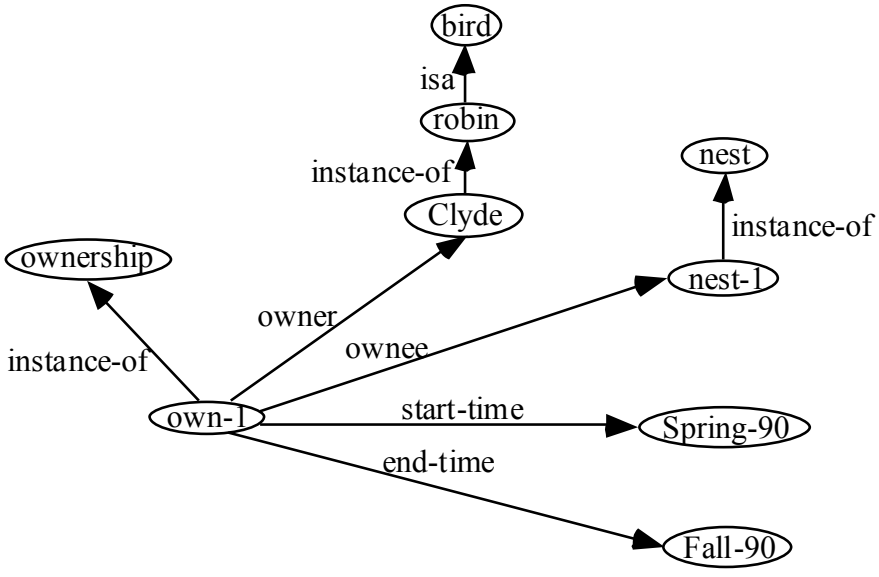

Figure 29. An ontology fragment representing ownership as concept (node).

22. References and Bibliography

Barbulescu M., Balan G., Boicu M., Tecuci G., "Rapid Development of Large Knowledge Bases," in *Proceedings of the 2003 IEEE International Conference on Systems*, *Man & Cybernetics*, Volume: 3, pp. 2169 - 2174, Washington D.C., October 5-8, 2003.

Boicu C., Tecuci G., "Mixed-Initiative Ontology Learning," in *Proceedings of the 2004 International conference on Artificial Intelligence*, IC-AI'2004, June 21 - 24, 2004, Monte Carlo Resort, Las Vegas, Nevada, USA. IEEE Computer Society, Los Alamitos, California, 2004.

Boicu M., Tecuci G., Stanescu B., Balan G.C. and Popovici E. (2001). Ontologies and the Knowledge Acquisition Bottleneck, in *Proceedings of IJCAI-2001 Workshop on Ontologies and Information Sharing,* Seattle, Washington, August 2001. http://lac.gmu.edu/publications/data/2001/ontbottleneck.pdf

Booch, G., Rumbaugh, J. and Jacobson, I. (1997). *The Unified Modeling Language User Guide:* Addison-Wesley.

Chimaera (2000). *Chimaera Ontology Environment.* www.ksl.stanford.edu/software/chimaera

Cyc (2008). OpenCyc just got better --- much better!, http://www.opencyc.org/, accessed on 22 Augusat 2008.

Duineveld, A.J., Stoter, R., Weiden, M.R., Kenepa, B. and Benjamins, V.R. (2000). WonderTools? A comparative study of ontological engineering tools. *International Journal of Human-Computer Studies* 52(6): 1111-1133.

Farquhar, A. (1997). *Ontolingua tutorial.* http://ksl-web.stanford.edu/people/axf/tutorial.pdf

Fellbaum, C. (ed.) (1998). *WordNet—An Electronic Lexical Database*. MIT Press, Cambridge.

Gómez-Pérez, A. (1998). Knowledge sharing and reuse. *Handbook of Applied Expert Systems.* Liebowitz, editor, CRC Press.

Gruber, T.R. (1993). A Translation Approach to Portable Ontology Specification. *Knowledge Acquisition* 5: 199-220. See also Gruber T. What is an ontology, http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

Gruninger, M. and Fox, M.S. (1995). Methodology for the Design and Evaluation of Ontologies. In: *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing*, IJCAI-95, Montreal.

Hendler, J. and McGuinness, D.L. (2000). The DARPA Agent Markup Language. *IEEE Intelligent Systems* 16(6): 67-73.

Humphreys, B.L. and Lindberg, D.A.B. (1993). The UMLS project: making the conceptual connection between users and the information they need. *Bulletin of the Medical Library Association* 81(2): 170.

Niles, I. and Pease, A. (2001). Towards a standard upper ontology. In Welty, C. and Smith, B. (eds) *Int. Conf. Formal Ontology In Information Systems* (*FOIS-2001*), Ogunquit, Maine, October 17–19, pp. 2–9. ACM Press, New York.

Noy N.F. and McGuinness D.L. (2001). Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.

Ontolingua (1997). *Ontolingua System Reference Manual*. http://www-ksl-svc.stanford.edu:5915/doc/frame-editor/index.html

Price, C. and Spackman, K. (2000). SNOMED clinical terms. *BJHC&IM-British Journal of Healthcare Computing & Information Management* 17(3): 27-31.

Protege (2000). *The Protege Project.* http://protege.stanford.edu

Rosch, E. (1978). Principles of Categorization. *Cognition and Categorization.* R. E. and B. B. Lloyd, editors. Hillside, NJ, Lawrence Erlbaum Publishers: 27-48.

Rothenfluh, T.R., Gennari, J.H., Eriksson, H., Puerta, A.R., Tu, S.W. and Musen, M.A. (1996). Reusable ontologies, knowledge-acquisition tools, and performance systems: PROTÉGÉ-II solutions to Sisyphus-2. *International Journal of Human-Computer Studies* 44: 303-332.

Tecuci G., *Building Intelligent Agents,* Academic Press, 1998, pp. 33-78.

Tecuci G. and Boicu M. (2008). Learning-Based Knowledge Representation, *Research Report 4,* Learning Agents Center (to appear).

Tecuci G., Boicu M., Boicu C., Marcu D., Stanescu B., Barbulescu M., The Disciple-RKF Learning and Reasoning Agent, *Computational Intelligence, Vol.21, No.4*, 2005, pp. 462-479.

Tecuci G., Boicu M., Marcu D., Barbulescu M., Boicu C., Le V., Hajduk T., Teaching Virtual Experts for Multi-Domain Collaborative Planning, *Journal of Software*, Volume 3, Number 3, pp. 38-59, March 2008.

Uschold, M. and Gruninger, M. (1996). Ontologies: Principles, Methods and Applications. Knowledge Engineering Review 11(2).Colomb, R.M., Formal versus Material Ontologies for Information Systems Interoperation in the Semantic Web, *The Computer Journal* Vol. 49 No. 1, 2006.

23. Acknowledgements

This material is based on research partially sponsored by the Air Force Office of Scientific Research (FA9550-07-1-0268), the Air Force Research Laboratory (FA8750-04-1-0257), and the National Science Foundation (0750461).

The following persons have contributed significantly to various versions of the ontology tools of Disciple: Marcel Barbulescu, Dorin Marcu, Cristina Boicu, Bogdan Stanescu, and Xianjun Hao.

24. Appendix: Operations Index