BOTTOM-UP ONTOLOGY CREATION

WITH

A DIRECT INSTANCE INPUT INTERFACE

A Thesis
Presented to
The Academic Faculty

by

Charles Cheng-hsi Wei

In Partial Fulfillment
of the Requirements for the Degree
Master of Computer Science in the
Department of Computer Science

California Polytechnic State University, San Luis Obispo
January, 2009

Approved by:

Dr. Franz Kurfess , Advisor
Department of Computer Science
California Polytechnic State University,
San Luis Obispo

Dr. [Committee Member2]
Department of Computer Science
California Polytechnic State University,
San Luis Obispo

Dr. [Committee Member3]
Department of Computer Science
California Polytechnic State University,
San Luis Obispo

Date Approved: February 2, 2009

ACKNOWLEDGEMENT

First of all, it is not possible to give enough thanks to my wife, Clare, without whose unfailingly loving, practical and emotional support this work would certainly have faltered. Indeed it may never have been undertaken at all. I am grateful to her. My daughter, Jamie has also of course been affected by my study. She has been, sometimes totally unexpectedly, impressed me with her surprising ideas. They are definitely my main support both on the study and my work.

My advisor, Dr. Franz Kurfess, is truly a great teacher and helped me a lot. I learned a lot in the knowledge management area. Without his advice I wouldn't be able to finish the thesis. I greatly appreciate his advice and help. Also, I have to thank Dr. K.C. Lo in Civil and Environmental Engineering Department, CalPoly. His advice and encouragement helped me to overcome all the difficulties I have encountered.

A special appreciation is for Aunt Y.L. Tsai. She helped not only me but also my family in the early stage of our life in the United States. Her advice of living in America has been really helpful.

Many thanks to my parents, who gave me full financial support when I was studying in CalPoly. Without their support I wouldn't have been able to finish my studies. Furthermore, without their encouragement in the very beginning, I would not have dared to study overseas and achieve my dream.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

In general an ontology is created by following a top-down, or so called genus-species approach, where the species are differentiated from the genus and from each other by means of differentiae [8]. The superconcept is the genus, every subconcept is a species, and the differentiae correspond to roles. To complete it a user organizes data into a proper structure, accompanied with the instances in that domain in order to complete the construction of an ontology. For example, it is a concept learning procedure in a school. Students first learn the general knowledge and apply them on their exercise and homework for practice. After they are more familiar with the knowledge, they can use what they have learned to solve the problems in their daily life. The deductive learning approach is based on the fundamental knowledge that a student has acquired already.

By contrast, a more intuitive way of learning is the bottom-up approach, which is based on atomism. That is also a frequently used way for humans to acquire knowledge. From sensing the world by vision, hearing, and touching, people learn information about actual objects, i.e., instances, in the world. After an instance has been collected, a relationship between it and existing knowledge will be created and an ontology will be formed automatically.

The primary goal of this thesis is to make a better instance input interface on the ontology development tool ProtÈgÈ to simplify the procedure of ontology construction. Without setting up the organization of class and properties (slots) first, simply input all the information of an instance and the program will form an ontology automatically. It means after an instance has been input, the system will find a proper location inside of

the ontology to store it. The second goal is to show the feasibility of a bottom-up approach for the building of an ontology.

CHAPTER 0

BACKGROUND

An ontology describes basic concepts in a domain and defines relations among them [1]. It provides the basic blocks in its structure: classes or concepts, properties or slots, and restrictions on slots. As a result, an ontology provides a common vocabulary for researchers who need to share information in a specific domain [1]. The goals to create an ontology are listed below:

- To share common understanding of the structure of information among people or software agents
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from operational knowledge
- To analyze domain knowledge

To create an ontology there are three main approaches. The first approach is generating an ontology from text-based documents. The second approach to create an ontology is extracting the concepts and relationships from large quantities of data. The last approach to make an ontology is model-based, which extracts the concepts and relationships from specifications, formalizations and computer-generated artifacts [23].

Ontology learning from text aims at generating domain ontologies from a given collection of textual resources by applying natural language processing and machine learning techniques [6]. However, it requires significant computational effort on natural language processing and it is still difficult to working on the knowledge which resides in

different languages. To simplify and narrow the scope of a research, I create an interface in ProtÈgÈ for instance input instead of retrieving the data from text-based documents in this thesis.

For data-driven approaches, for example, data mining with an FCA (Formal Concept Analysis), are tried to retrieve or analyze the relationship between data. Originally the concepts only exist in human's mind. When it comes to computer processing, these concepts must be transformed to formal ones which will be stored in knowledge bases [19]. It encounters the same problem as the first approach I mentioned above. As a result I decided not to pursue this approach in this thesis.

To make the questions of automatic ontology creation simpler, I roughly divided it into two parts. The first portion is retrieving the information from the existing documents. For the reasons I mentioned above, this is not easy to achieve with the existing knowledge. However, the second part of building up an ontology in a more efficient way is a reachable goal with the existing techniques. For the goal of improving the ontology creation, it brings up two questions of this thesis: how to improve the efficiency of information input and how to form an ontology automatically.

The solution for the former question in this thesis is creating an instance input interface which combines the many steps and interfaces of information input into one. The solution of the later question is the bottom-up creation of ontology. Both achievements help improve the automation of ontology creation. The details will be illustrated in the following chapters.

CHAPTER 1

INTRODUCTION

What is an Ontology?

In both computer science and information science, an ontology is a formal explicit description of concepts in a domain of discourse (classes (sometimes called concepts)), properties of each concept describing various features and attributes of the concept (slots (sometimes called roles or properties)), and restrictions on slots (facets (sometimes called role restrictions)). An ontology together with a set of individual instances of classes constitutes a knowledge base. [1]

One of the main purposes to use an ontology is to easily share a common understanding of the structure of information among people or software agents since it provides a common vocabulary for researchers who need to share information in the domain. It provides a format which is highly organized and clarified in a category for easy understanding and implementation.

It can also be used to analyze and reuse domain knowledge. A mature ontology is a highly organized structure and it is easy to clarify or retrieve the relationship between classes. The slots in classes are also good for users to find out properties affiliated with concepts, or the respective instances. As a result it helps to identify the functionality and characteristic of each class and instance.

Furthermore, a well-built ontology makes domain assumptions explicit and separates domain knowledge from operational knowledge. This aspect of ontologies is an advantage for knowledge sharing and reusing for different topics and fields. It makes an

ontology easy to apply to the main idea without redundant information. This characteristic also offers high flexibility of knowledge usage.

What is the basic approaches to build an ontology?

Knowledge-base construction often begins at the "top," with higher-level class definitions, and then proceeds downwards to the process of creating instances of those classes [4]. This is also the way of learning in school. We apply the theory to practical examples after we have learned the general knowledge in a particular domain. It follows the top-down process.

In the contrast, a more intuitive way of learning is the bottom-up approach. Try to imagine the way a baby learns. With touching, listening, and observing individual objects, a baby memorizes the items with their names, which are defined by their parents or instructor, and with their special meaning and properties. Each individual object is an actual instance to a baby. After gathering enough instances in his/her mind, an individual organization of information is formed. This is how we learn in the very beginning of our life.

In this thesis, I tried to use the bottom-up approach to build up or expand an ontology. Similar to the way a baby learns, the Instance Input interface I developed under the ProtÈgÈ ontology development environment which allows users to input the instances with their own specific slots (properties) and the values along with those slots directly. In other words, it is an input-then-organize (bottom-up) approach instead of the traditional, organize-then-input (top-down) approach.

In addition, in this research I use the bottom-up approach not only to simulate the learning process but also to try to increase the automation of ontology creation. In my

view, I divide the ontology creation into two main steps: the first one is instance recognition and identification, which is used to recognize and define the new information. The subsequent process is ontology formation, which organizes the new information into an existing ontology or a new one. The former is beyond the scope of this thesis. I will merely focus on how to use the bottom-up approach to do the automation of ontology creation with a simple filtering process based on slots here. The result indicates that it is simple but powerful.

The bottom-up approach with slot filtering process

Different from the top-down approach, the bottom-up approach works by

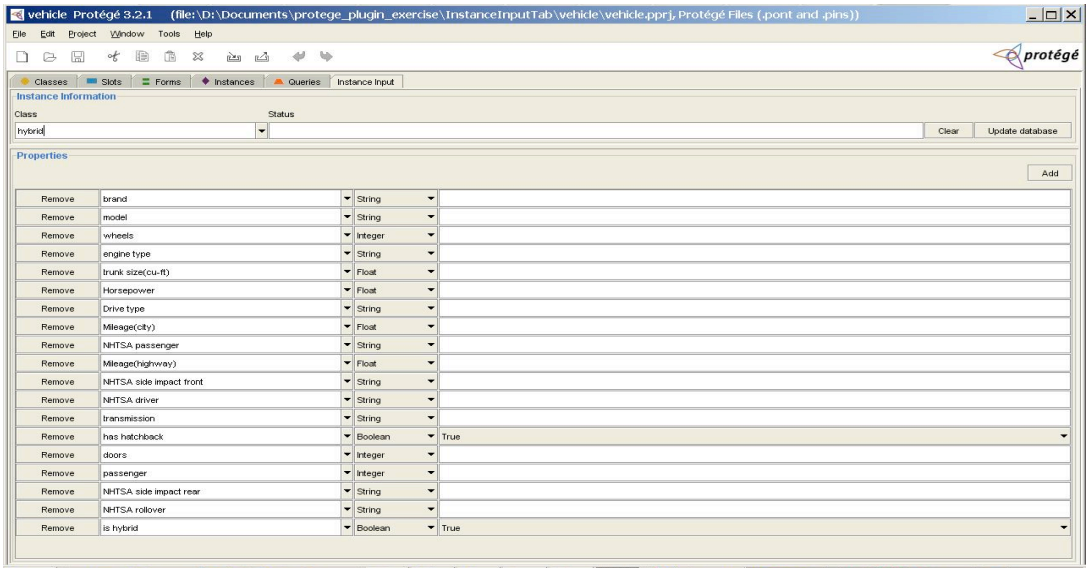

Figure 1-1

inputting instances under its own class before the structure of classes has been built. The first step is not organizing all the information in a specific domain but analyzing the properties of an instance that will be added to an ontology. This is the instance recognition and identification procedure which is done manually by users in this thesis. As a result, the slots belonging to an instance should be clarified and the values of each

slot should be identified. The properties are the key features to identify an instance and make it unique in an ontology.

For example, as shown in Figure 1-1 above, a hybrid car has several properties for people to identify it. Those properties appear in the format of slots with values in an instance. Based on the slot filtering process within the interface in this thesis the 'hybrid' instance can be locate at a most proper location.

The slot filtering process uses a simple rule: the subclasses of the specific class in which the new instances reside not only contain all the slots of that class but more. On the other hand, the specific class not only contains all the slots belonging to its super-classes but more. With proper slots defined in a class within an ontology, the interface can insert the class or instance to a proper position automatically.

Thesis Goal 1: The improvement of efficiency on information input

ProtÈgÈ is an integration environment which provides a complete integration environment for ontology creation and implementation. Its goals are (1) achieving interoperability with other knowledge-representation systems, and (2) being an easy-to-use and configurable knowledge-acquisition tool [3]. An expert can create an ontology or modify the organization with the initial interfaces.

However, developing an ontology is usually an iterative process [1], not to speak of creating an ontology in ProtÈgÈ. A user needs to define the classes, slots, and instances in it, categorize them and switch back and forth between several interfaces to input them. The procedure of building an ontology for a huge amount of data is a difficult and time consuming task [22].

To improve and simplify the steps, I developed an interface to facilitate instance input with slots. With this interface users can create an instance directly instead of

creating the classes where it resides and the slots which it possesses in advance. It provides a flexible way to input instances with all the slot information. The details will be discussed in Chapter Three.

Thesis Goal 2: Automation of ontology creation

Developing an ontology is usually an iterative process. You start with a rough first pass at the ontology. You then revise and refine the evolving ontology and fill in the details [1]. Manually arranging the hierarchy inside of an ontology is very difficult and also a time consuming issue since it is similar to a task of organizing your files in your computer. You need to identify them, categorize them, and put them into proper folders. Especially in situations where an ontology is to be shared among multiple individuals, its structure also to some degree reflects the "mental model" of its creators. This can lead to problems such as incorrect knowledge, inconsistencies across the knowledge base, inappropriate levels of detail, or a general mismatch of the model used by the creators with the model of the intended user.

In this thesis I use bottom-up approach to complete the creation of an ontology. Different from the traditional top-down approach I mentioned above, it collects the information from the most specific instances to form the whole ontology with the most general category on the top. Also to simplify the process, the hierarchy relationship is not generated but defined already in ProtÈgÈ. The subclasses contain all the properties which their parental classes have. The details will be discussed in the following chapter.

CHAPTER 2

APPROACH OVERVIEW

There are two approaches I used to build an ontology automatic in this thesis. The first is building the ontology from the specific to the general, which is a bottom-up approach. The other is categorizing the ontology by searching the "best match" location with the properties of an instance. The interface achieves the thesis goal with the approaches above.

Approach # 1: building an ontology from the specific to the general

For this approach I created an interface which allows users to input the instances individually. It completes an ontology creation based on atomism. By contrast to the Aristotelian genus-species approach, atomism proceeds bottom-up in that it builds objects out of smaller objects [8], which are instances in ontologies. In an ontology creation the genus-species uses the top-down process for the differentiation of classes and instances. This is not the solution I want to apply.

The knowledge model of Protege-2000 is frame-based: frames are the principal building blocks of a knowledge base. A Protege ontology consists of classes, slots, facets, and axioms [3]. With the definition described in previous sentence an ontology is formed into a hierarchy which contains the classes from the most general concept at the top to the most specific samples at the bottom.

The basic idea of the bottom-up approach is building an ontology from instances to form a hierarchy of classes from the most specific to the most general, that is, concepts. Since most of the basic information a human being learns is not from the concepts in the very beginning but all identical instances around in his/her life, the bottom-up approach would be the most intuitive way building a concept.

There was a very interesting example I noticed it from the growth of my daughter.

When Jamie, my daughter, was three years old, she knew that we had an old Mercury Sable station wagon which was golden. However, Jamie didn't know our car belonged to a category named 'vehicle', or an 'auto', to say nothing of 'mid-size' station wagon. All the general concepts didn't mean any thing to her. The ideas of that 'thing' in her mind were:

- It's "ours".
- It has five doors.
- It has four wheels.
- It is golden.
- It is big.

As a result, they are the properties of our old golden Mercury Sable. Based on those properties listed above, whenever Jamie saw the same model cars on the street, she would say "it's our big golden car", or "it's not our big golden car". She could not tell what "our" was since she didn't know the meaning of "our". As she learned from the specific instance, the golden Mercury Sable, she classified the cars into two categories: one is our big golden car and the other is "not our big golden car".

Later Jamie learned another car from her uncle. It was a golden Mazda 3. From that she noticed the brands and she knew our car was Mercury Sable. Therefore her description of a car had been changed. Instead she said "it is a Toyota", "it is a Ford", or "it is a Honda" when she saw cars.

Now she knows the different types, brands and models of cars on the street and they are the instances of different categories, such as SUV, sedan, minivan, hatchback, and so on. Furthermore, she understands those different categories are the subclasses of “autos” which is under a more general concept named “vehicle”.

The example above indicates a relationship between each class and the instances and the construction procedure of a concept: the fundamental information is gathered from specific things individually and then connected to form the knowledge.

The bottom-up approach of ontology creation is the same as the above process. It is based on the information collection by instance identification. In the very beginning users don’t touch the global view of the whole concept but only some specific instances or a small portion of the whole ontology. Gradually it will form an integration of a specific knowledge by expending it with repeating the procedure of instance

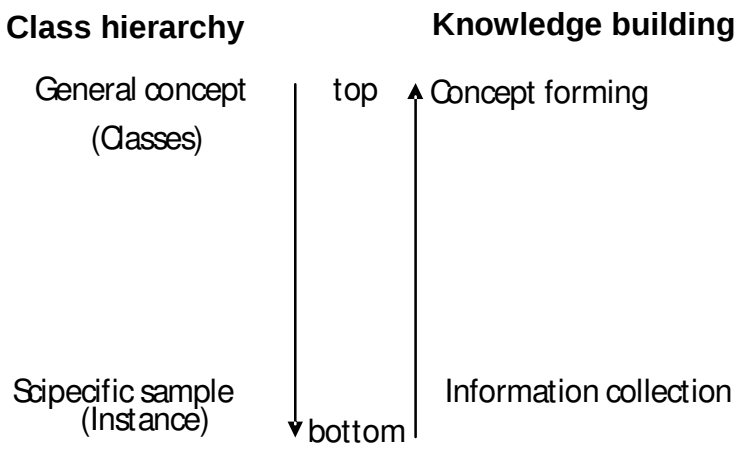

Figure 2-1

identification and location.

Another simple example is a puzzle game. From the beginning a player needs to identify the shape and picture of each piece, then look for the only location where it fits. The procedure is very similar to our bottom-up approach. Identification of a piece is the process of recognizing a specific instance. Searching the proper location is identical to

the steps of insert an instance along with the class it resides. There is also a close relationship to the methods used in constraint programming and constraint satisfaction; due to time constraints, however, this connection was not further explored in this thesis.

Based on the operation of the instance input interface, it meets the requirement defined in the beginning. The interface provides a proper and simple way to create or expend an ontology by easily inputting instances.

Approach # 2: searching the most appropriate location for a class with the slots

Once a new instance has been filled with slots and values, it is ready to be added to an ontology. Therefore, finding a suitable location for it becomes the second topic for the bottom-up approach of ontology creation. In this thesis,]the following two rules are applied to find a suitable location:

Rule #1 :

The new instance resides in a class which contains all the slots of its superclass.

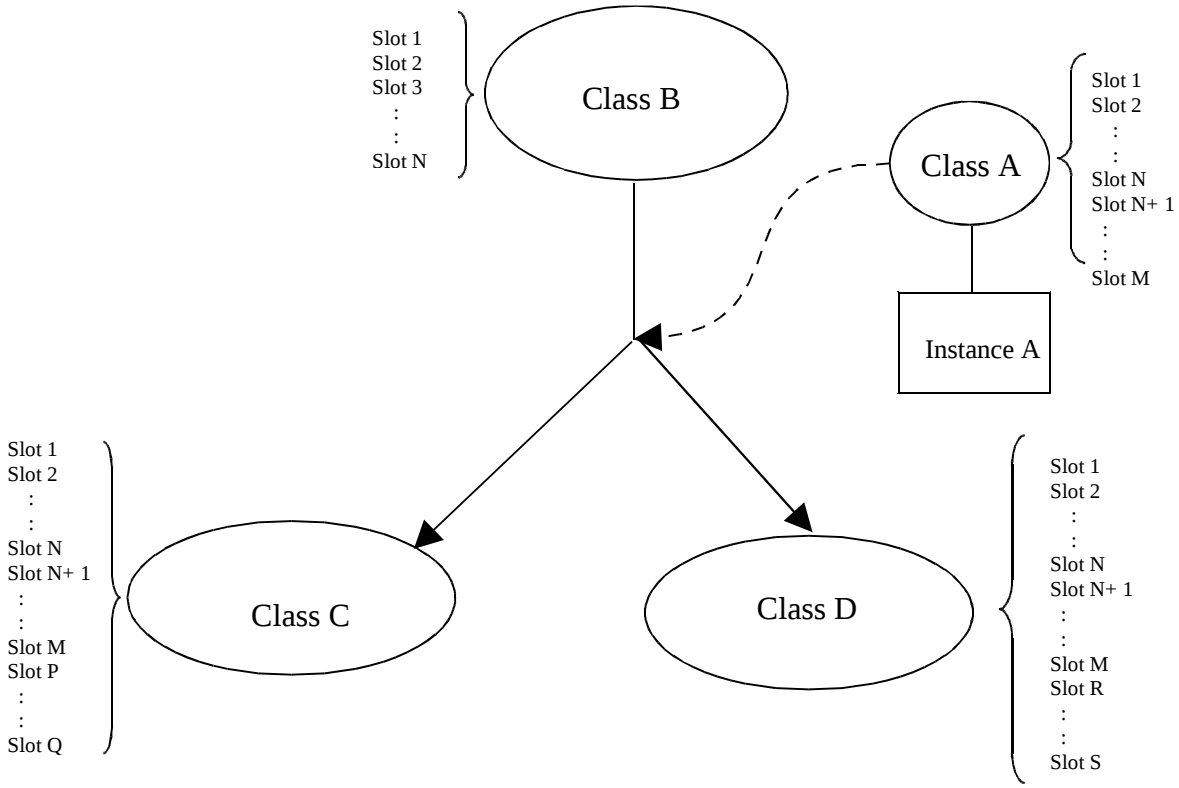

Figure 2-2

As well as its subclasses contain all the slots of it. Figure 2-2 shows how the Class A with Instance A inserts to an existing ontology:

Class B is the direct superclass of Class C and Class D. In Figure 2-2 it indicates that the relationship: the subclasses contain all the slots which belong to superclass. As a result, Class C has at least all the slots in Class B and so does Class D. Now a new Class A along with Instance A, which contains all the slots in Class B, therefore Class A is one of the subclasses of Class B. On the other hand, Class C and Class D contain all the slots in Class A, so Class A becomes the direct superclass of Class C and Class D.

In logic expression it looks as below:

{the collection of slots belong to Class A} $\subset$ ıe collection of slots belong to the subclass of Class A}

and

{the collection of slots belong to Class A} $\supset$ he collection of slots belong to the superclass of Class A}

As you may see, this rule is very simple but powerful. According to the samples I tested in my thesis, it can locate the proper position for the class which the new instance resides. Inside of the interface the program compares the slots belonging to the new instance with the slots of classes in the ontology. As soon as it finds the first matched class, it will put the instance there. Once it can not find a class which contains the exact same slots, the interface will create a new class based on the rule above. The result shows it is a satisfied solution.

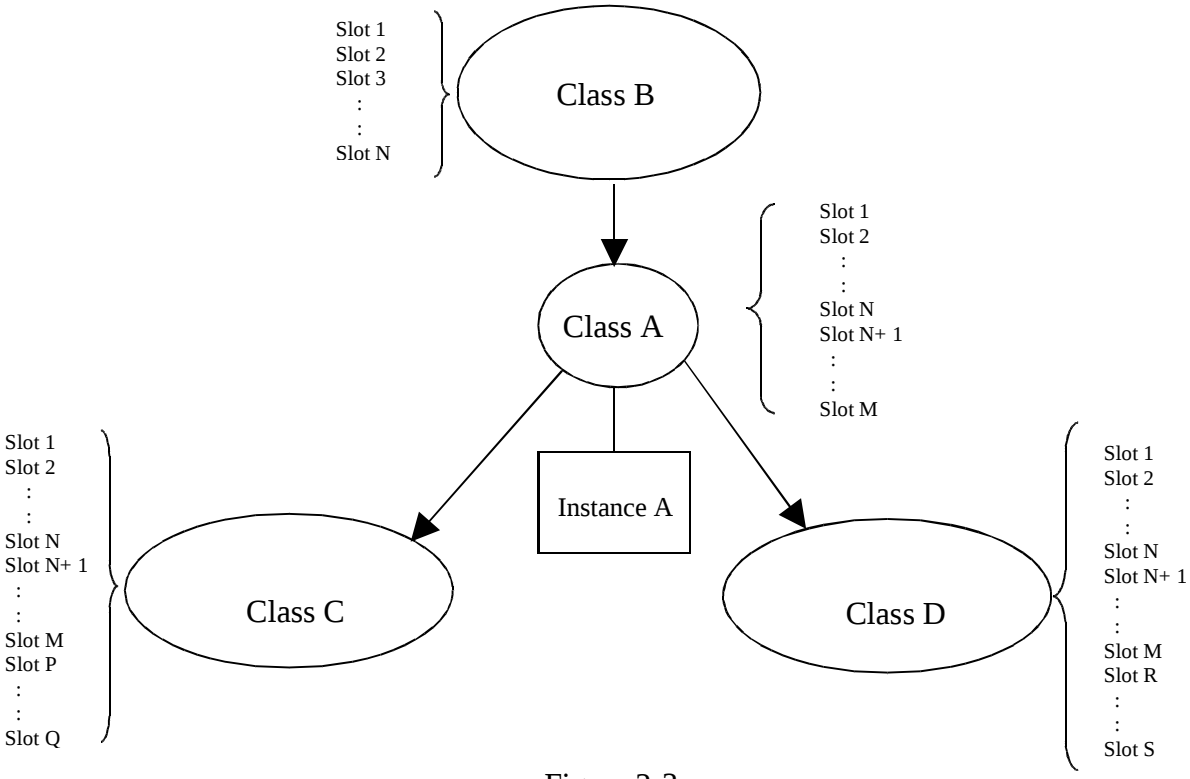

Figure 2-3

Rule #2 :

The second rule is about how to locate the class more accurately. To make the result more precisely, a good naming rule of slots is necessary. The rule is “naming a slot as specific as possible”. It defines an instance clearly and indicates the characteristics of the class which the instance resides.

The searching algorithm I used in the program to find the proper location of classes and instances is depth-first. As a result it will search on the same level of classes before deeper dive. Without using more specific name of slots would cause misplaced the instances. In order to increase the hit rate, which helps less modification after input, applying the specific names of slots is necessary.

For example, most of the classes have a property named “name”. It indicates how people call it. In most cases it is good enough for categorizing the classes precisely. However, it confuses the program sometimes and locates the classes not in the best place since the slot name “name” is not specific enough to describe which domain it resides.

With a more accurate name would help the program find a better location. As a result, a slot named “sedan name” under the “sedan” class or a “computer monitor name” is more specific than simply “name”. According to the samples it can achieve a very accuracy of locating the class within the best place.

Unfortunately, with lacking the semantic comparison ability, my program is not able to place all the instances input into locations expected. I will discuss this in the samples of Chapter 5.

CHAPTER 3

PROGRAM FEATURES

The Instance Input Interface is a tab widget developed and used on ProtÈgÈ 3.2.1. It allows a user to input instances and classes in an easier way than the traditional procedure of Ontology construction. For the detail of its features and requirements will be described in this chapter.

Environment requirement

The Instance Input Interface is written in Java 1.5.0 and implemented ProtÈgÈ 3.2.1 API. It requires to be operated on ProtÈgÈ 3.2.1 or higher edition installed on the computer.

Key features

One of the main purposes for this program is to provide users an easier instance input interface. As a result, the key features of the interface are:

1. Condensed procedure:

   With the interface, users can add an instance under a specific class by inputting proper values of selected slots which belong to that class. Or it can make a new instance within a new class simultaneously. Furthermore, it is used to create a new class alone and set its properties up in one operation. It simplifies the input procedure of an instance and a class, as well as saves the time of creating an Ontology.

2. Intuitive operation:

   The interface contains most of the functions for instance input and is arranged for intuitive operation. A blank interface shows below:

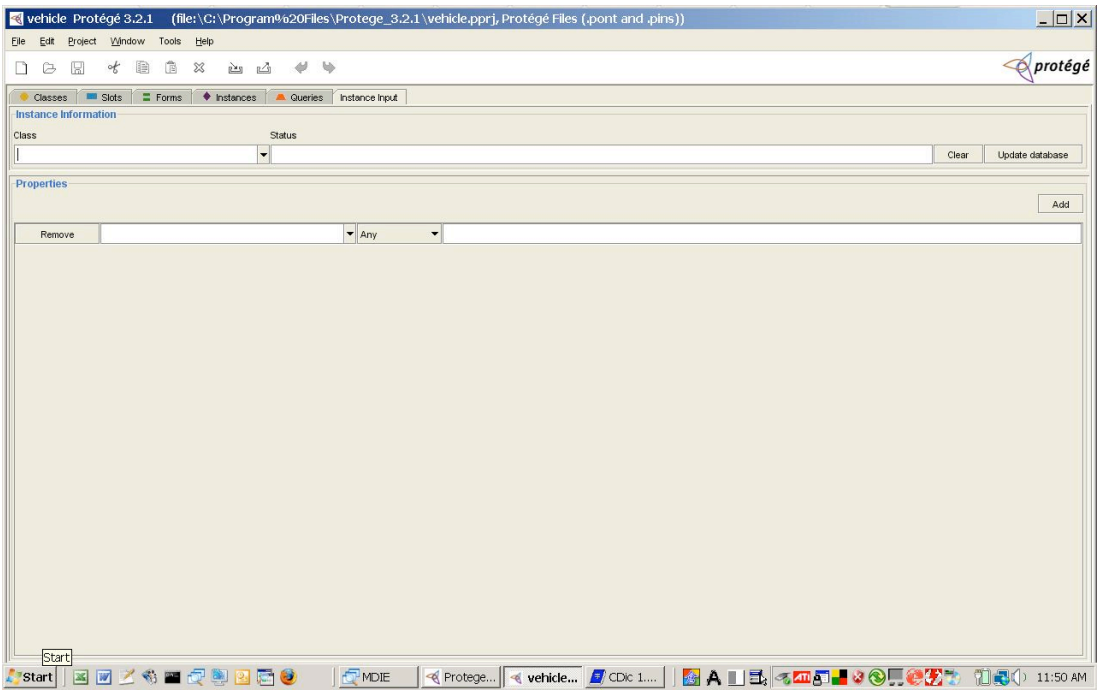

Figure 3-1

It is simply to press the “Add” button on the right-upper corner inside of “Properties” field to add a new slot with a single click. Deleting a slot is as easy as clicking the “Remove” button on the most left hand side of that rows. Each row contains the editable slot name list which allows users to select an existing slot from the database or add a new one. By choosing a slot type within a combo list a slot can be defined for 7 different types of data in ProtÈgÈ.

In addition, to input a slot with multiple values is easy within this interface. Users simply add multiple rows of slots with the same name and different values, and the interface will set the slot as a “multiple input” property.

3. Automatic categorization

   After an instance has been input, the interface will categorize it into the Ontology with the General-to-Specific rule which I described in Chapter 2. The program searches the best matched location based on the properties belonging to the instance.

   (1) There is a class with exactly the same properties as the instance: this instance will be put under that specific class.

   (2) The instance contains all the properties of a class: this instance will be put under a new created class which is under the specific class which contains fewer properties.

   (3) No class matches the instance: the program will create a new class under "Root" class and put the instance under it.

For more detail operation instruction, it will be described in next chapter.

Installation

There are 2 simple steps to install the Instance Input Interface onto ProtÈgÈ:

1. Create a folder named "InstanceInputTab" under "ProtÈgÈ_3.2.1\plugins\" and copy the InstanceInput.jar into it.
2. Start ProtÈgÈ with either an existing project or a new project. Click the "Project" on the tool bar and choose "Configure…" selection under it. Find the "InstanceInputTab" and check the "Visible" box in front of it.

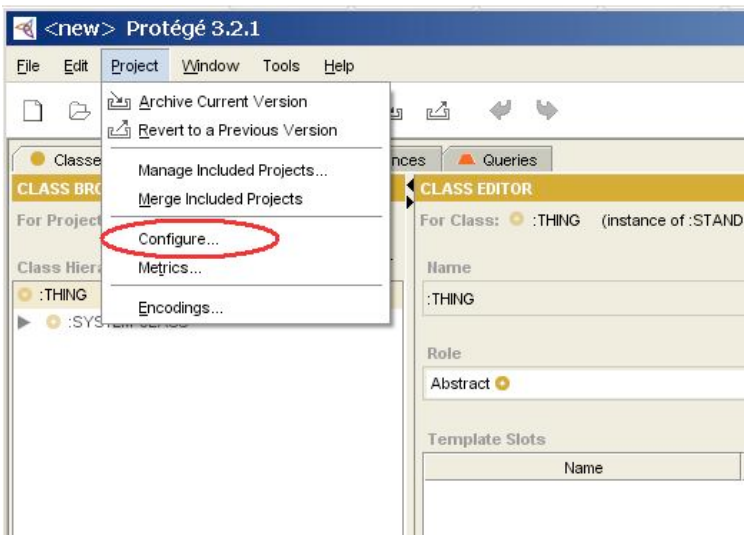

Figure 3-2

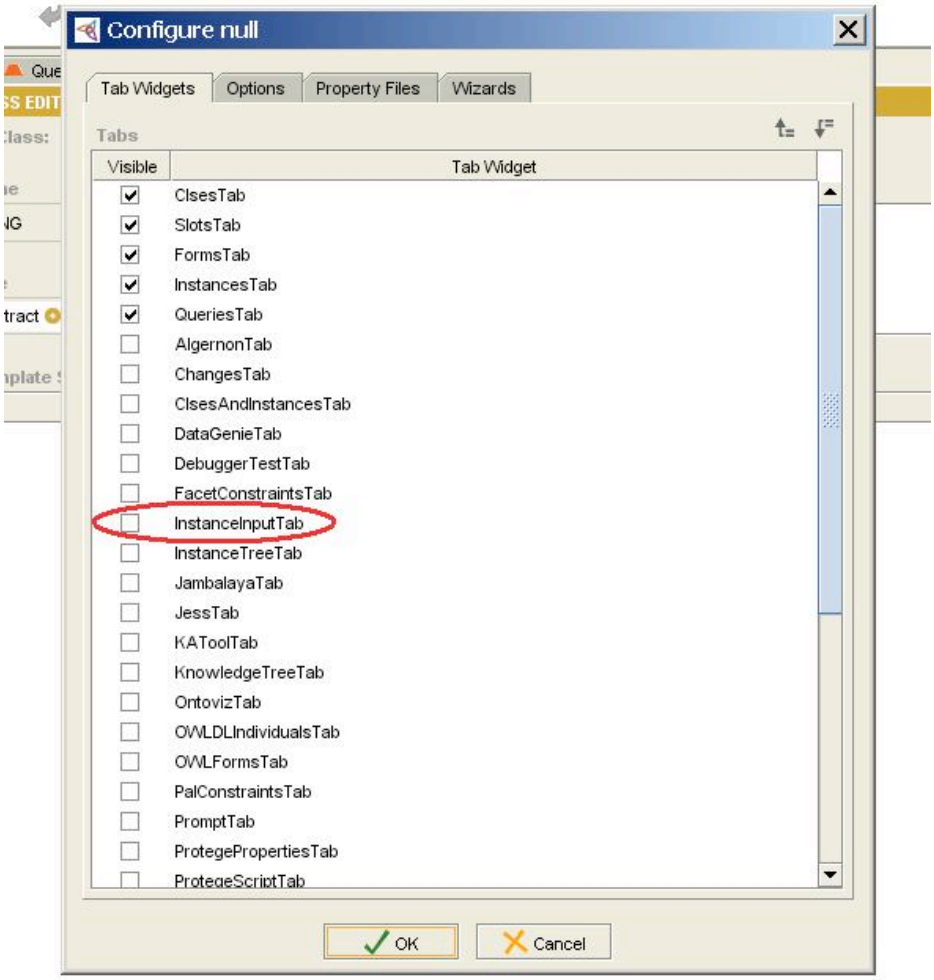

Figure 3-3

3. Now the Instance Input Interface is ready to use.

CHAPTER 4

INTERFACE OPERATION

The new tab widget provides a easier way to input instances. With simply single step, users are able to input an empty class, an instance within a specific new class, or a new instance under an existing class with this interface. It will dramatically simplify the construction of an ontology.

This chapter contains three main portions: instance information, instance properties, and operation rules. The first two describe the functionality of each button and input area or selection. The last one focuses on the operation of creating instances.

(1) Instance information

This is the upper area inside of the interface. It shows the selection of Class, the status of interface, and has two buttons for clearing the input or passing the input result to ProtÈgÈ.

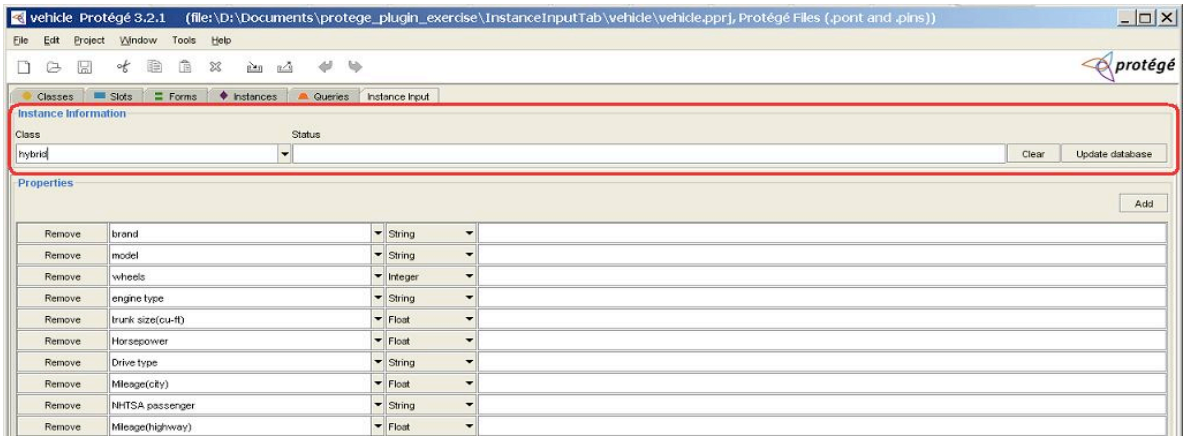

Figure 4-1

1. ‘Class’ combobox

This is an combobox which indicates the class holds the new instance. It provides two functions:

(i) List all the slots which belong to the selected class in the Instance properties area automatically. On the other hand, once the slots in the Instance properties area are exactly the same as a specific class, the name of that class will be pop out in the Class combobox.

(ii) Allow user to input the name of a new Class.

2. ‘Status’ text box

   The ‘Status’ text box is not an edible text field. It shows the certain information during the running time. This is for debug only.

3. ‘Clear’ button

   This button is used to discard the information has been input. It clears all the selection and values in ‘Properties’ area.

4. ‘Update Database’ button

   ‘Update Database’ button is used to pass the input data to ProtÈgÈ. At the same time the class and instance list will be synchronized with the ontology database. As well as the ‘Properties’ will be cleared for next input.

   Before updating database with pressing ‘Update Database’ button, the slots and values will be temporarily stored in the interface and can be modified without affecting the ontology database.

(2) Instance Properties

The lower area of the interface lists all the properties of an instance. All the slots and values are able to set up here. It combines the definition of slot and instance value input in one step.

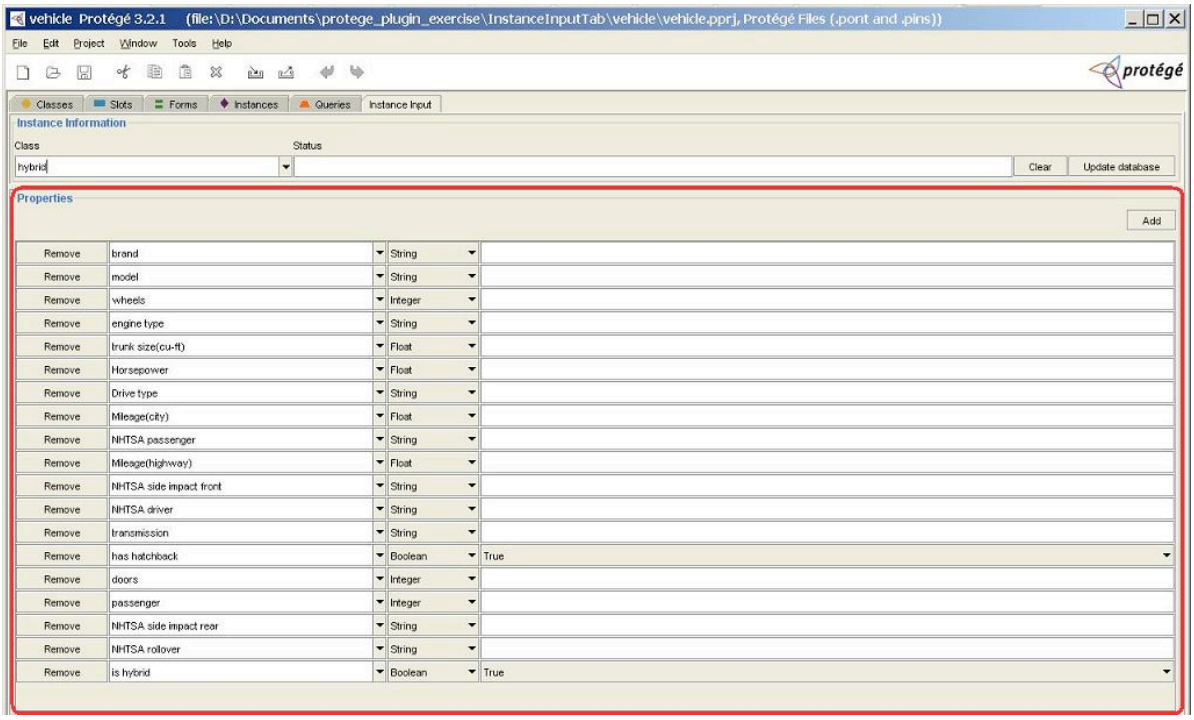

Figure 4-2

1. ‘Add’ button

   Every click of ‘Add’ button adds an empty row of slot for input. There is no default value and slot name for a new slot except slot type. The default value of slot type is ‘Any’.

2. ‘Remove’ button

   On the most left hand side of each row, it is a ‘Remove’ button. As its name shows this button is used to delete the row of slot which it resides. It designs for the purpose of intuitive usage. With one click a row will be removed

from the property input area.

3. ‘Slot name’ combobox

   The second edible area from the left in a row is the ‘Slot name’ combobox. It contains the list of existing slots in the ontology. By selecting the slot name in the list will change the value of slot type and the value input area accordingly.

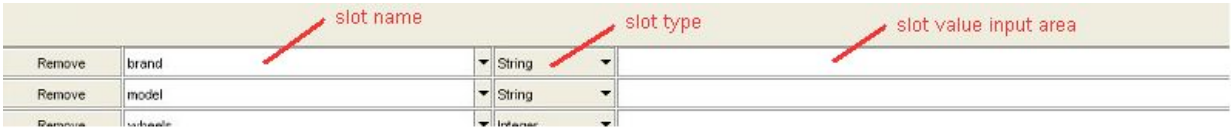

Figure 4-3

   The same as the ‘Class’ combobox in ‘Instance information’ area, users can input a new slot name in the ‘Slot name’ combobox to create a new slot in the ontology.

4. ‘Slot type’ list box

   Next to the ‘Slot name’ combobox on the right hand side is the ‘Slot type’ list box. It shows the type of a slot and also controls the type of value input area on the right. There are eight different selections in the list: Any, Boolean, Class, Float, Integer, Instance, String, and Symbol.

5. ‘Slot value’ input area

   On the left of each row is the ‘Slot value’ input area. It is an dynamic input area and will change to meet the slot type selected in ‘Slot type’ list box. For the type of Any, Float, Integer and String, an text field shows and allows users to type the value. Once the slot type is changed to Boolean, this area

will switch to a True or False selection list box. In the type of Class, Instance, or Symbol, it will be a list box which contains all the available classes, instances, or symbols respectively.

(3) Operation rules

There are five rules for different cases of instance input. Following the rules users can create an empty class, add properties to a class, insert a instance in a class, create an instance within a new class, or directly make a new instance with slots contains multiple values.

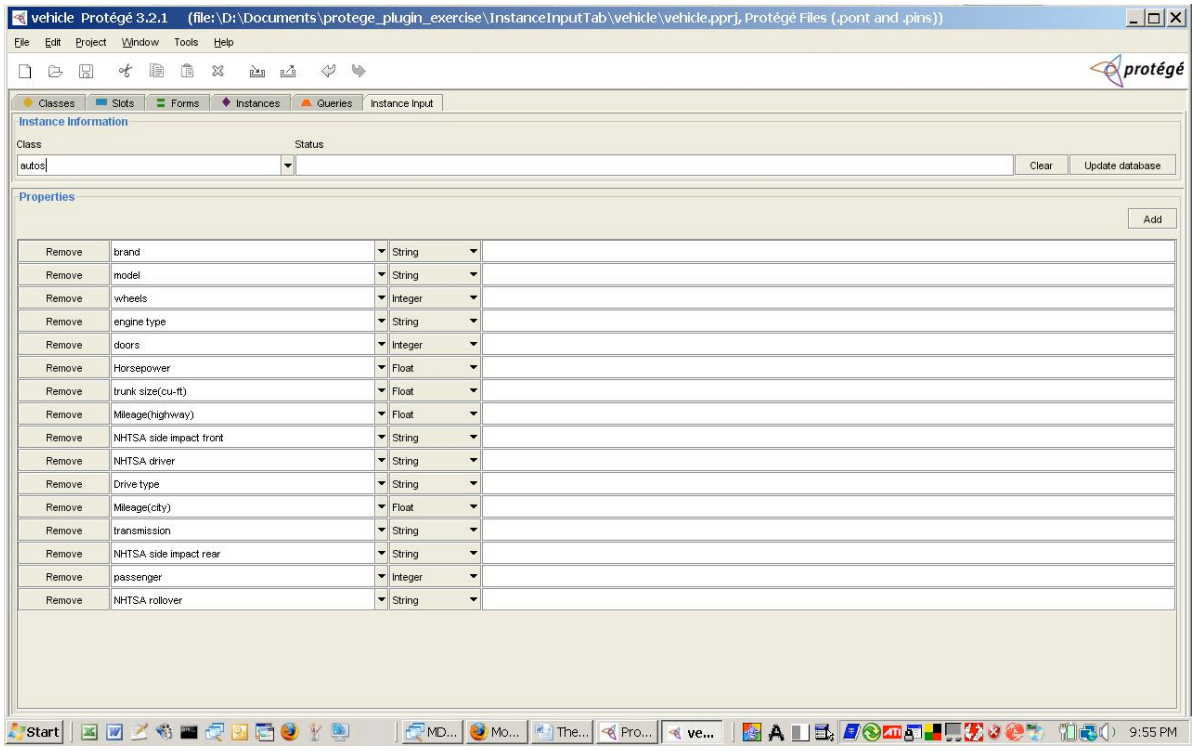

Figure 4-4

1. Create a new class with no instance

   This rule applies to create an empty class alone. It is useful for the construction of an ontology since not all the classes need instances under it.

To create an empty class, simply add all the slots in the properties area without input any values. After finishing the slot addition, click the “Update Database” button in instance information area will add the class to the ontology.

An example shows in Figure 4-4. A class named ‘autos’ is a sub-category of ‘vehicle’ and also a superior class of ‘sedan’, ‘hybrid’, ‘pickup’, and so on. Since all the individual instances of autos will be put in the more specific classes under it, there is no need to set any instance inside of it. As a result, ‘autos’ would be an empty class can be created by applying this rule.

2. Create a new instance within a new class

   Similar to the first rule above, the steps to create a new instance within a new class is the same as to create an empty class. The only difference between them is the former needs the user to fill in all the values. After filling in all values and pressing the ‘Update Database’ button, a new class with the instance under it will be added to the ontology.

3. Add properties to an existing class

   Sometimes users need to modify the definition of an existing class by adding a certain slots. The task can be achieved by using the Instance Input Interface with the following steps:

   (i) Users select the class from the ‘Class’ combobox in ‘Instance information’ area. The slots belong to that specific class will display in the ‘Properties’ area.

   (ii) Click on ‘Add’ button in ‘Properties’ area. In the new row, users may select a slot which exists in the ontology or input a new slot name and select a proper slot type. Then fill in the value at the new

row.

(iii) Users may notice that the selection in ‘Class’ combobox becomes blank since the slots in ‘Properties’ has been changed and the interface will treat them as belonging to a different class. At this time the users place the same class name back to the ‘Class’ combobox and press ‘Update Database’. The new slots will be added to that class.

4. Add a new instance to an existing class

To add a new instance to an existing class is the simplest operation by using this interface. Merely within two steps the creation of an instance can be done.

(i) Select the class from the ‘Class’ combobox in ‘Instance information’ area. The slots belong to that specific class will display in the ‘Properties’ area.

(ii) Fill in the values for each slot and press the ‘Update Database’ button at the end. The new instance will add under the specific class.

Furthermore, it is easy to add an instance with slots which contain multiple values. While a user fills in the values for each slot, he/she may click the ‘Add’ button to increase the slot rows. In the new rows the user selects the slot name which he/she wants to make with multiple values. After filling in all the values, the user types the same class name and clicks the ‘Update Database’ button. The new instance with slots contains multiple values will be added to the ontology.

CHAPTER 5

EVALUATION

There are two sections of evaluation: the first section is the improvement of new interface on ontology creation. The second section is how the bottom-up approach helps on ontology creation.

Evaluation 1: Improvement of new interface on ontology creation

The original idea of designing the interface was creating a more user-friendly operation environment for ontology users. It helps to reduce the input steps on ontology creation. The following figure shows the difference:

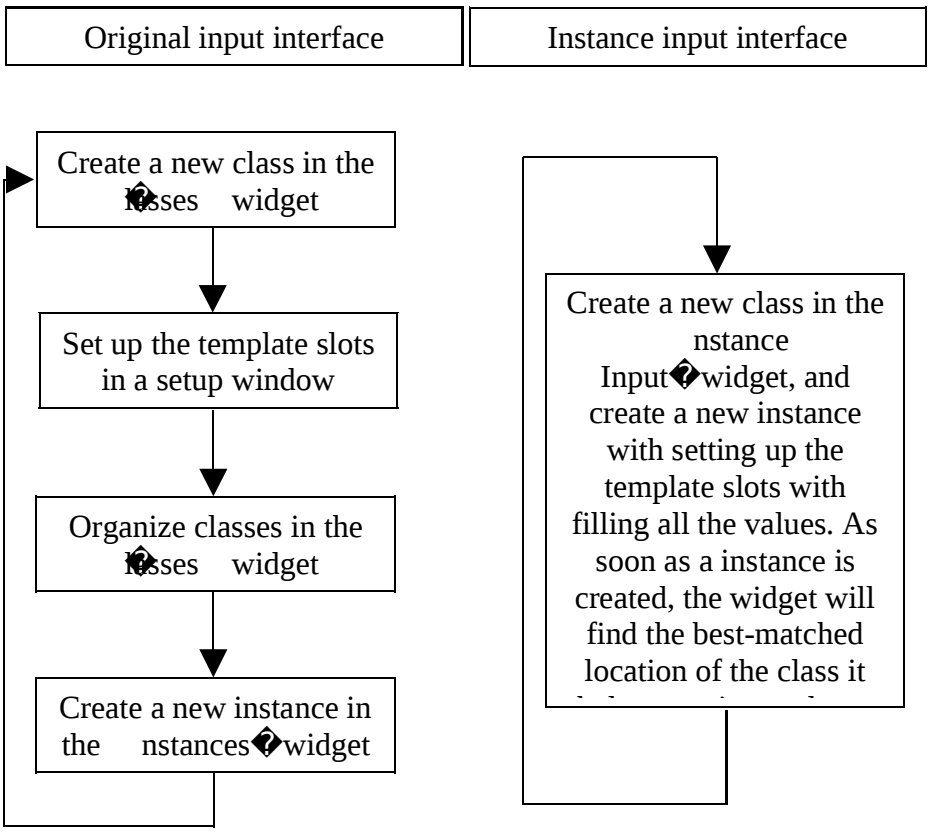

Figure 5-1 Comparison of original and improved steps with new interface

As it shows above, the original procedure of ontology creation is repeating four steps by switching between three widgets and windows:

- Creating a new classes in the “Classes” widget
- Set up the template slots in a setup window
- Organize classes in the “Classes” widget
- Create a new instance in the “Instances” widget

With the new Instance Input Interface, all four steps can be done in one widget. It not only reduces the annoying clicks and switches between windows, but also helps to organize classes with the bottom-up approach on ontology creation which I will evaluate in the following paragraphs. In those real examples of ontology, it did help me to save time and actions on the input tasks.

The reduction of clicks and switches varies. It depends on the number of template slots and instances. Since the input task with new interface completes within a widget, the more templates and instances an user inputs, the more time and actions it saves. The new interface provides a more convenient and efficient working environment on ontology creation for general users.

Evaluation 2: Performance of bottom-up approach on ontology creation

Totally I built ten ontologies for testing. In this chapter I will report three typical examples to show both the strength and weakness on modifying an existing ontology or creating brand new ontologies respectively. Within a different example it demonstrates the different grade of success caused by its limitation and the strategy I applied. The detail will be described and discussed in the case individually.

The evaluation of the approach is based on the “hit rate”. To get the appropriate hit rate, I use the following equation to evaluate the result for each example:

$$\text{Hit Rate (\%)} = \frac{\text{The total of the classes which reside in the proper locations}}{\text{The total of the classes which have been input}}$$

The higher "Hit Rate" will indicate the rules I used in the Instance Input Interface is useful. In other words, the approach I imply in this thesis is good enough to do the front end process. Otherwise the approach needs to be refined with the rules improvement.

The definition of "the total of the classes which reside in the proper location" is the number of the classes which either is at the same location as the comparison or at the location as the author expects. I don't count the instances since they all follow the classes which they belongs to. In other words, once the classes reside in the correct place, all the instances under them will locate at the correct categories. Therefore there is no need to count the instances to bluffing the Hit Rate.

Example 1: Newspaper

This example shows how to use the Instance Input Interface to insert a new instance into an existing ontology. It demonstrates the ability of expansion with the Instance Input Interface in an easier way. An user can input either instances or classes to expend the ontologies simply in one interface instead of switching back and forth between the Class Management Widget and Instance Management Widget. In this case I add the following classes and instances into Newspaper ontology:

Photographer

The slots of a Photographer are:

- Name
- Current_job_title
- Date_hired

- Other_information
- Phone_number
- Salafy
- Camera

Cartoonist

The slots of a Cartoonist are:

- Name
- Current_job_title
- Date_hired
- Other_information
- Phone_number
- Salafy
- Drawing tools

Office Technician

The slots of a Office Technician are:

- Name
- Current_job_title
- Date_hired
- Other_information
- Phone_number
- Salafy
- In office / Boolean

Publish Technician

The slots of a Office Technician are:

- Name

- Current_job_title
- Date_hired
- Other_information
- Phone_number
- Salafy
- In publish department / Boolean

Contractor

The slots of a Contractor are:

- Name
- Other_information
- Phone_number
- Contractor period
- Rate

Intern

The slots of an Intern are:

- Name
- Other_information
- Phone_number
- Intern period
- Rate

There is another general category named “Technician” which contains all technician sub-categories. I added it in the ontology after I created the publish technician and office technician to test the insertion function of classes.

Those six classes I used to test the bottom-up approach ontology creation. My

idea was to set the “Photographer” and “Cartoonist” under the “Columnist” class, “Office Technician” and “Publish Technician” under the “Employee” class, and “Contractor” and “Intern” under “Person” respectively. However, the Hit rate is 66% before I input the instances with the identification slots such as “Is Columnist” for “Photographer” and “Cartoonist”. The reasons for both classes failed are because the classes are sorted and stored in depth-first inside of ProtÈgÈ and I didn’t change the order of them. When my program searches the proper location for a new instance, it will compare each class in the same order and insert the new instance with its class into the first location matched. Since there is no semantic analysis ability in my program, it is not able to identify the difference of the classes with same slots. As a result it will put the input instance to the first location which meets the Rule#1 in Chapter Two.

The simple solution before implying the semantic identification is applying the Rule#2 to make each class unique. It comes out with the results I expected to organize. By adding “Is Columnist” to “Photographer” an “Cartoonist”, they are correctly located under “Columnist” category.

In contrast, “Technician”, “Office Technician”, “Shop Technician”, “Contractor” and “Intern” reside at correct location since they have some slots make them unique. For example, “Technician” contains all the slots same as the rest classes under “Employee” with one extra specific slot named “Is Technician”, which makes the program will search and put it under the “Employee” and in parallel with “Columnist”, “Editor”, etc.

In this case it shows obviously the abilities and limits of bottom-up approach with simple comparison rule without semantic identification. However, it also demonstrates the capability and possibility of improvement on ontology creation. The next case supports my point of view.

Example 2: HEV

In order to reduce the fuel consumption with the contemporary technologies,

HEVs have been invented and are popular in our community at the high gasoline price point. However, HEV is not the final solution since the improvement of fuel economy is not great enough. Therefore PHEV is introduced for much better mileage per gallon within certain of commute range. It is also a step stone for the development of pure electric vehicles. The HEV ontology shows the contemporary status of both HEV and PHEV in the United States.

In this case I use the Instance Input Interface to input all the HEVs available on the consumer market nowadays to form the HEV ontology. It demonstrates the ability of building an ontology with bottom-up approach and the improvement of input procedures. There are total 68 instances and 11 classes input and the 'hit rate' is 100%. Since I followed the rule #2 which was described in Chapter 2 when I input the instances, the ontology is formed in a good structure without moving the location of either classes or instances. It provides another supporting example to show the possibility and ability of automatic ontology creation. However, there are some restricts and weakness in my program which are caused by the approach I use and the lack of ability on data recognition. It will be discussed in the following example.

HEV

The slots of a HEV are:

- Brand: the brand of autos
- Model: the identical model of autos
- Battery: Mainly the batteries apply on HEV is either NiMH or Lithium.
- Engine: The main stream of engine types are gasoline and diesel ICE. However, the models in the concurrent market are all equipped with gasoline.
- Year: the production year of autos

- Doors: the main entrance available on an auto
- Seating: the maximum number of seats in an auto
- City mpg: the fuel consumption while driving in city
- Highway mpg: the fuel consumption while driving on highway

In order to distinguish the engine type and battery type for categorization, I add the following two slots to help identification:

- Gasoline powered: it is equipped with gasoline ICE
- NiMH battery powered: it uses NiMH battery pack to store electricity.

As I mentioned before, since the Instance Input Interface doesn't have the ability to identify the semantic information of data for categorization, I need to add those redundant-like slots for assistance. However, the strategy helps a lot on categorize ontologies in all the examples I built.

PHEV

The slots in a PHEV are:

- Brand: the brand of autos
- Model: the identical model of autos
- Battery: Mainly the batteries apply on PHEV is either NiMH or Lithium.

- Charger: the equipment to provide the ability of charging by home electricity or the high-power fast charging

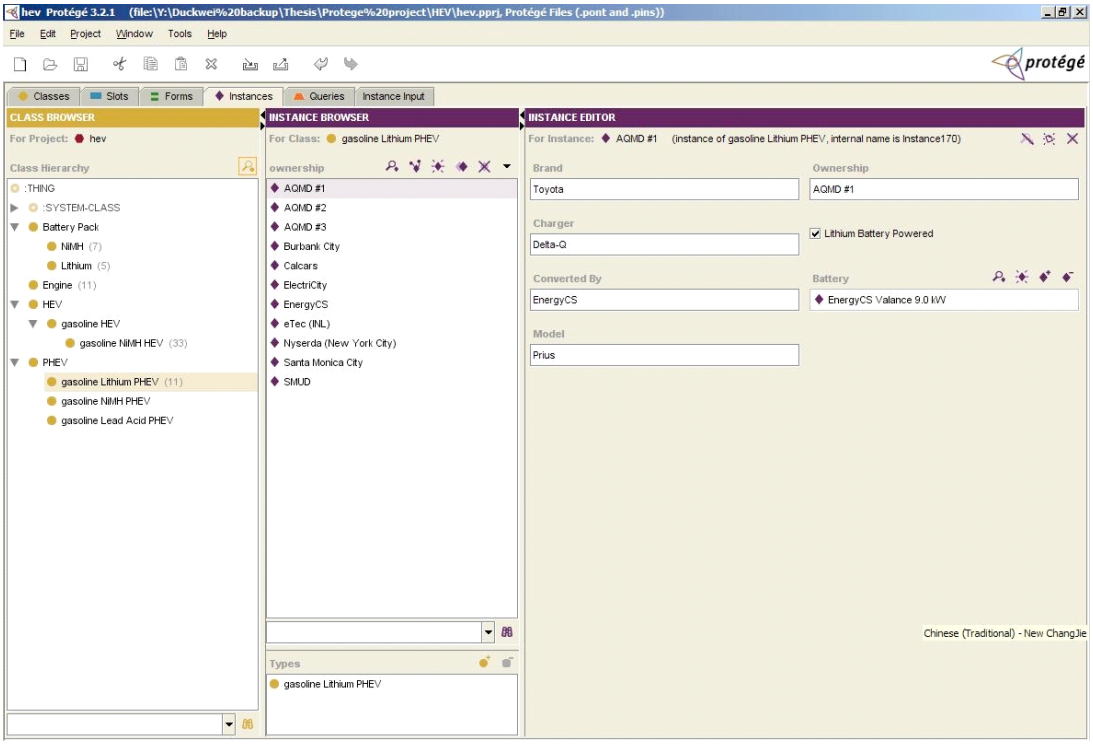

Figure 5.2 HEV Ontology

- Converted by: the company which provides the conversion service or the technology provider
- Ownership: the funder or the possessor

Since there is no OEM PHEV launched the auto market yet, all the PHEVs are converted by different companies with different technologies. The ontology briefly shows the status and the organizations which own the converted fleets.

Example 3: REA Enterprise Ontology

REA Enterprise Ontology has been initially created by William E. McCarthy,

mainly for modeling of accounting systems. |t proves useful and intuitive for better understanding of business processes. In this example I will use the interface I write to recreate the whole REA Enterprise Ontology with all the instances in it and examine how well my program can build the same ontology as it.

To be honest this is a fail case when I tried to clone it onto ProtÈgÈ with the Instance Input Interface. The reason I put here is using it as a contract to the previous two cases. It shows how it fails and gives me the idea to improve in the future work.

The main reason my program failed to correctly organize the classes in REA ontology is that many classes contain the same slots. As a result there is not enough information to categorize the classes for the Instance Input Interface I make. At most it is only able to put the classes under the same superclasses with fewer slots. With the same slots the classes would be put in parallel.

For example, Agreement class under ExchangeElement has the same slots as Contract class, which is under Agreement class in the original REA ontology. Unfortunately by using my interface to input the data, Contract class would be put under ExchangeElement instead of Agreement since the program locates it with the slots belonging to.

There are two solutions for this problem. The first solution is applying the Rule#2 in Chapter Two to insert at least one specific slot for identification. It will provide the necessary information for recognition and help the program to locate the correct position for the classes. For instance, after I added a slot named “Is_Agreement” with Boolean type to Contract class and it resided at the correct location right under Agreement.

The second solution is adding the semantic recognition ability to raise the correct rate of classification. Since most of the identification ability in an ontology is based on the information in text, the semantic recognition will provide more information which is the benefit for data classification used on ontology creation. However, it is too complicate and beyond the scope of this thesis.

As you may see in the cases I provide in this chapter, the bottom-up approach with the classification approach by slots is useful to help the users on the automation of ontology creation. It not only provides a way to simplify the ontology creation but also prove the idea of bottom-up approach for building an ontology is possible even though it is still amateur at this stage. However, with more study I believe it would be able to improve the work of creating ontologies greatly.

CHAPTER 6

CONCLUSION

From the examples in previous chapter, the new interface provides a more intuitive operation as well as shows the practicability of bottom-up approach for an ontology creation. It simplifies the process of ontology creation. In addition, it also helps users to locate the class which the new instance resides automatically. Both form the semi-automatic ontology creation procedure.

Benefits and achievement

The benefit of using the interface developed in this thesis is simplifying the procedure of instance input. Typically an ontology is built by top-down approach. It takes the following procedure:

1. To analyze information for a specific domain
2. To organize the hierarchy of classes
3. To setup the slots of each class
4. To input instances under each class

In contrast to the procedures mentioned above, the interface allows user to do the same job with the following steps:

1. To analyze information of each unique instance in a domain
2. To input the slots of an instance and value for each slot

Simply with two steps the interface helps user to do the classification work and form a prototype of ontology. It reduces some tedious job and provides a skeleton of ontology for easy modification.

As I mentioned in the very beginning, testing the ontology creation with bottom-up approach is the main idea throughout this thesis. The results confirm that it is possible

to build an ontology from bottom to top. It also shows the possibility to generate an ontology automatically with more precision categorization on the properties and values of instances.

The major achievement of this thesis is that the interface does the categorization of instances automatically. It takes the instances which users input, identifies them with their slots, and puts them into the best match locations in an ontology. The examples shown in previous chapter demonstrate that the results are satisfied with well-defined slots.

The study process of this thesis also answers some of my questions from the observation on how the fundamental knowledge forms in a kid, such as my daughter. Similar to the process of generating an ontology with bottom-up approach, the fundamental knowledge is constructed from individual instances which kids encounter everyday. From them with their own categorizing rules they build their own ontologies in their way. They are free of restricts. The learning process in school system helps to organize them and improve them. Unfortunately, it also limits the freedom of thinking. Nevertheless, this is a topic beyond the scope of this thesis. I won't address this too much here.

Future works

However, it is the first step to achieve the automation of ontology creation. In this thesis the interface simply use slots in a class as the key information to form a class organization. It is powerful but not enough for a more accurate classification. For the further improvement, there are some future works to make the automation more practical:

1. To do the classification of classes and instances with not only slots but also value of each slot.

   In order to get more precision classification, it is not enough by merely using the slots. Value in each slot is very important for categorizing it. The slots

can be used to form a rough class hierarchy and the value in each slot would help to adjust the result. However, it requires to imply some semantic analysis and information retrieval procedure which beyond the study here.

2. To make the interface contains more function.

   The interface in the thesis develops for the following targets:

   (1) Clear interface

   (2) Intuitive operation

   (3) Versatile function

   (4) Automatic classification process

As a result, to be a perfect user-friendly operation interface, there are still some improvement needs to be done for the future work:

1. The semantic ability for the identification of name and value of slots

   This is a very important factor to dramatically improve the accuracy of correct organizing the ontology since the ability of semantic identification is able to help the program to recognize the information input and find the right location in an ontology for those instances.

2. More control items for detail configuration of each slot and its value

3. To imply some external ontology for performing the classification of classes and instances.

To use the bottom-up approach with slot information to form an ontology directly from the input instances is the first step of automation of ontology creation. In this thesis I demonstrate the possibility and strength and how it works. Although it is not a very mature application which is able to be implied onto the related field, the idea shows its potential and the availability to help solving the problems on the automatic creation of ontology.

APPENDIX A

SOURCE CODE

```
package InstanceInputTab;     // Protege 3.1


import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.util.*;
import java.lang.reflect.*;

import edu.stanford.smi.protege.model.*;
import edu.stanford.smi.protege.widget.*;
import edu.stanford.smi.protege.util.*;
import edu.stanford.smi.protege.resource.*;

// an example tab
public class InstanceInputTab extends AbstractTabWidget {
        public static final int HEIGHT_DEFAULT = 25;
        public static final long serialVersionUID = 24362462L;

        Collection _clses;
        Collection _slots;
        Collection _instes;

        private JPanel PropertyPane;
        private JPanel slotInputPane;
        private JScrollPane scrollSlotInputPane;

        private JComboBox className;
        private JTextField status;
    private JButton Clear;
    private JButton Sync;
        private JButton Update;
    private String str;

        private JButton addSlot;

    private JSplitPane mainSplitter;
```

```
private Dimension slotBoxSize;
private Dimension buttonSize;
	private Dimension slotNameSize;
	private Dimension slotTypeSize;

Box emptyBox;

private int currentID;
private int slotNo;
	private int instanceNo;
	private int instMaxCount;

private int HEIGHT;
	private int WIDTH;

	private String debugStr;

Vector<Cls> classList;
	Vector<String> clsNameList;
Vector<Slot> slotList;
	Vector<String> slotNameList;
	Vector<Instance> instanceList;
Vector<String> instNameList;
	Vector<String> clsSelection;

Vector<String> slotTypeChoices;
Vector<String> booleanChoices;

	KnowledgeBase KB;

// startup code
public void initialize() {
   // initialize the tab text
   slotNo = 1;
   currentID = 0;
   HEIGHT = 0;
		WIDTH = 1248;
   setLabel("Instance Input");

		debugStr = "";

		slotBoxSize = new Dimension(WIDTH, HEIGHT_DEFAULT);
		buttonSize = new Dimension(100, HEIGHT_DEFAULT);
		slotNameSize = new Dimension(300, HEIGHT_DEFAULT);
		slotTypeSize = new Dimension(100, HEIGHT_DEFAULT);
```

```
    KB = getKnowledgeBase();

            instMaxCount = 0;
            instanceNo = KB.getInstanceCount(KB.getRootCls());



            classList = new Vector<Cls>();
            clsNameList = new Vector<String>();
            slotList = new Vector<Slot>();
            slotNameList = new Vector<String>();
       instanceList = new Vector<Instance>();
            instNameList = new Vector<String>();
            clsSelection = new Vector<String>();

            slotTypeChoices = new Vector<String>();
            booleanChoices = new Vector<String>();

            slotTypeChoices.addElement("Any");
            slotTypeChoices.addElement("Boolean");
            slotTypeChoices.addElement("Class");
            slotTypeChoices.addElement("Float");
            slotTypeChoices.addElement("Instance");
            slotTypeChoices.addElement("Integer");
            slotTypeChoices.addElement("String");
            slotTypeChoices.addElement("Symbol");

            booleanChoices.addElement("True");
            booleanChoices.addElement("False");

            readData();

  add(createMainSplitter());
}

    private void readData() { // read data from knowledge database
            _clses = Collections.EMPTY_LIST; // it was "new ArrayList()"
            _slots = Collections.EMPTY_LIST;
            _instes = Collections.EMPTY_LIST;

  try {
    _clses = KB.getClses();
                    classList.clear();
                    clsNameList.clear();
                    clsNameList.addElement("");
            for (Iterator clsIterator = _clses.iterator(); clsIterator.hasNext();) {
```

```
                Cls cls = (Cls) clsIterator.next();
                if(cls.getName().charAt(0) != ':' && !cls.isAbstract()) {
                                        classList.addElement(cls);
                                        clsNameList.addElement(cls.getName());
                }
        }

        _slots = KB.getSlots();
                        slotList.clear();
                        slotNameList.clear();
                        slotNameList.addElement("");
        for (Iterator slotIterator = _slots.iterator(); slotIterator.hasNext();) {
                Slot slot = (Slot) slotIterator.next();
                if(slot.getName().charAt(0) != ':') {
                                        slotList.addElement(slot);
                                        slotNameList.addElement(slot.getName());
                }
        }


        _instes = KB.getInstances();
                        instanceList.clear();
                        instNameList.clear();
                        instNameList.addElement("");
                for (Iterator instIterator = _instes.iterator(); instIterator.hasNext();) {
                Instance inst = (Instance) instIterator.next();
                if(inst.getDirectType().getName().charAt(0) != ':') {
                                        instanceList.addElement(inst);
                                        instNameList.addElement(inst.getBrowserText());
                                        String internalNoStr = inst.getName().substring(8);
                                        debugStr = debugStr + " " + inst.getName() +
internalNoStr;
                                        status.setText(debugStr);
                                        int internalNo = Integer.valueOf(internalNoStr);
                                        if (instMaxCount < internalNo) {
                                                instMaxCount = internalNo;
                                        }
                }
        }
                        instMaxCount++;
      } catch (Exception db) {
        System.out.println("database reading error");
      }
        }

    private JComponent createMainSplitter() {
```

```
            mainSplitter = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
                mainSplitter.setDividerLocation(75);// original: 50
        mainSplitter.setTopComponent(createInstanceInfoPane());
        mainSplitter.setBottomComponent(createPropertyPane());
        return mainSplitter;
    }

    private JComponent createInstanceInfoPane() {
        JPanel InstanceInfo = new JPanel();
        InstanceInfo.setLayout(new BorderLayout());

        InstanceInfo.add(createControlPanel(), BorderLayout.CENTER);

        InstanceInfo.setBorder(BorderFactory.createTitledBorder("Instance Information"));
        return InstanceInfo;
    }

    private JComponent createControlPanel() {
                JLabel classNameLabel = new JLabel("Class");
                classNameLabel.setPreferredSize(new Dimension(300,
HEIGHT_DEFAULT)); // was 150
                classNameLabel.setMaximumSize(new Dimension(300,
HEIGHT_DEFAULT)); // was 150
                JLabel instanceNameLabel = new JLabel("Instance Name");
                JLabel statusLabel = new JLabel("Status");
                statusLabel.setPreferredSize(new Dimension(150,
HEIGHT_DEFAULT));
                statusLabel.setMaximumSize(new Dimension(150,
HEIGHT_DEFAULT));

                Box boxLabel = Box.createHorizontalBox();
                boxLabel.add(classNameLabel);
                boxLabel.add(statusLabel);
                boxLabel.add(Box.createHorizontalGlue());

                className = new JComboBox(clsNameList);
                className.setPreferredSize(new Dimension(300, HEIGHT_DEFAULT));
//was 150
                className.setEditable(true);

            status = new JTextField();
        status.setText("");
        status.setPreferredSize(new Dimension(80, HEIGHT_DEFAULT));
                status.setEditable(true);

        Clear = new JButton("Clear");
```

```
        Update = new JButton("Update database");

    Box box1 = Box.createHorizontalBox();
            box1.add(className);
    box1.add(status);
    box1.add(Clear);
    box1.add(Update);
    box1.add(Box.createHorizontalGlue());

            Box outerBox = Box.createVerticalBox();
            outerBox.add(boxLabel);
            outerBox.add(box1);

    JPanel controlPanel = new JPanel();
    controlPanel.setLayout(new GridLayout(1, 6, 3, 3));
            controlPanel.add(outerBox);

            className.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                            JComboBox source = (JComboBox) e.getSource();
                    }
            });

            className.addItemListener(new ItemListener() {
                    public void itemStateChanged(ItemEvent ie) {
                    }
            });

            className.addPopupMenuListener(new PopupMenuListener() {
                    public void popupMenuWillBecomeVisible(PopupMenuEvent
pme) {
                    }
                    public void popupMenuWillBecomeInvisible(PopupMenuEvent
pme) {
                            JComboBox source = (JComboBox) pme.getSource();
                            String currentItem = (String) source.getSelectedItem();
                            if(currentItem.length() > 0) {
                                    clearInput();
                                    readData();

                                    Cls cls = getClsFromSelection(currentItem);

                                    Collection ownSlots = cls.getTemplateSlots();
                                    Iterator slotIterator = ownSlots.iterator();
                                    while (slotIterator.hasNext()) {
                                            Slot tmpSlot = (Slot) slotIterator.next();
```

```
                                        String slotValueTypeStr =
tmpSlot.getValueType().toString();
                                        slotInputPane.add(createSlot(tmpSlot,
slotValueTypeStr));
                                }
                                slotInputPane.add(createEmptyBox());
                                slotInputPane.revalidate();
                        }
                }
                public void popupMenuCanceled(PopupMenuEvent pme) {
                }
        });

    Clear.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
                        clearInput();
                        readData();
                        className.setSelectedIndex(0);

                        slotInputPane.add(createSlot());
                        slotInputPane.add(createEmptyBox());
                        slotInputPane.revalidate();
                }
    });

    Update.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
                        updateDatabase();
                        clearInput();
                        readData();
                        className.setSelectedIndex(1);
                        className.setSelectedIndex(0);


                        slotInputPane.add(createSlot());
                        slotInputPane.add(createEmptyBox());
                        slotInputPane.revalidate();
                }
    });

    status.addCaretListener(new CaretListener() {
      public void caretUpdate(CaretEvent e) {
      }
    });

    return controlPanel;
```

```
    }

        private void updateDatabase() {
                //update slots
                ArrayList<Slot> newSlots;
                ArrayList<Object> newValues;
                Instance newInstance;
                Box SB;
                boolean existSlot = false;

                newSlots = new ArrayList<Slot>();
                newValues = new ArrayList<Object>();
                int i;
                for(i = 0; i < slotInputPane.getComponentCount() - 1; i++) {
                        // Since the last component is "empty box" ==>
getComponentCount() needs to substract 1
                        SB = (Box) slotInputPane.getComponent(i);// slotBox
                        JComboBox slotJB = (JComboBox) SB.getComponent(1); // slot
name JComboBox
                        JComboBox typeJB = (JComboBox) SB.getComponent(2); //
value type JComboBox of slot
                        JComponent slotValueJC = (JComponent) SB.getComponent(3); //
value JComponent of slot
                        String slotNameStr = (String) slotJB.getSelectedItem();
                        String typeValueStr = (String) typeJB.getSelectedItem();
                        Object newValue;

                        if(slotNameStr.length() > 0) { // means it is a new slot
                                existSlot = false;
                                Iterator slotIterator = _slots.iterator();
                                while (slotIterator.hasNext()) {
                                        Slot existingSlot = (Slot) slotIterator.next();

        if((existingSlot.getName()).contentEquals(slotNameStr)) {
                                                newSlots.add(existingSlot);
                                                existSlot = true;
                                                if(typeValueStr.contentEquals("Any")) {
                                                        newValue = ((JTextField)
slotValueJC).getText();
                                                } else
if(typeValueStr.contentEquals("Boolean")) {
                                                        int selection = ((JComboBox)
slotValueJC).getSelectedIndex();
                                                        newValue = selection == 0 ? true :
false;
                                                } else
```

```
if(typeValueStr.contentEquals("Class")) {
                                        int selection = ((JComboBox)
slotValueJC).getSelectedIndex();
                                        newValue = classList.get(selection);
                                } else
if(typeValueStr.contentEquals("Float")) {
                                        newValue = ((JTextField)
slotValueJC).getText();
                                } else
if(typeValueStr.contentEquals("Instance")) {
                                        newValue = (String) ((JComboBox)
slotValueJC).getSelectedItem();
                                } else
if(typeValueStr.contentEquals("Integer")) {
                                        newValue = ((JTextField)
slotValueJC).getText();
                                } else
if(typeValueStr.contentEquals("String")) {
                                        newValue = ((JTextField)
slotValueJC).getText();
                                } else { // Symbol
                                        newValue = (String) ((JComboBox)
slotValueJC).getSelectedItem();
                                }
                                newValues.add(newValue);
                                break;
                        }
                }
                if(!existSlot) {
                        Slot newSlot = KB.createSlot(slotNameStr);
                        if(typeValueStr.contentEquals("Any")) {

        newSlot.setValueType(edu.stanford.smi.protege.model.ValueType.ANY);
                                newValue = ((JTextField)
slotValueJC).getText();
                        } else if(typeValueStr.contentEquals("Boolean")) {

        newSlot.setValueType(edu.stanford.smi.protege.model.ValueType.BOOLEAN);
                                int selection = ((JComboBox)
slotValueJC).getSelectedIndex();
                                newValue = selection == 0 ? true : false;
                        } else if(typeValueStr.contentEquals("Class")) {

        newSlot.setValueType(edu.stanford.smi.protege.model.ValueType.CLS);
                                int selection = ((JComboBox)
slotValueJC).getSelectedIndex();
```

```
                                        newValue = classList.get(selection);
                                    } else if(typeValueStr.contentEquals("Float")) {

        newSlot.setValueType(edu.stanford.smi.protege.model.ValueType.FLOAT);
                                        newValue = ((JTextField)
slotValueJC).getText();
                                    } else if(typeValueStr.contentEquals("Instance")) {

        newSlot.setValueType(edu.stanford.smi.protege.model.ValueType.INSTANCE);
                                        newValue = (String) ((JComboBox)
slotValueJC).getSelectedItem();
                                    } else if(typeValueStr.contentEquals("Integer")) {

        newSlot.setValueType(edu.stanford.smi.protege.model.ValueType.INTEGER);
                                        newValue = ((JTextField)
slotValueJC).getText();
                                    } else if(typeValueStr.contentEquals("String")) {

        newSlot.setValueType(edu.stanford.smi.protege.model.ValueType.STRING);
                                        newValue = ((JTextField)
slotValueJC).getText();
                                    } else { // Symbol

        newSlot.setValueType(edu.stanford.smi.protege.model.ValueType.SYMBOL);
                                        newValue = ((JTextField)
slotValueJC).getText();
                                    }
                                    newSlots.add(newSlot);
                                    newValues.add(newValue);
                                    readData();
                                }
                            }
                        }// end of for(checking slots)


                        if(newSlots.size() != 0) {
                            // Check if cls exists
                            Cls rootCls = KB.getRootCls();
                            Cls selectedCls;
                            boolean foundSuperCls = false;
                            Object classNameCurrentItem = className.getSelectedItem();
                            //String location = "run = ";
                            if (!clsNameList.contains(classNameCurrentItem) || ((String)
classNameCurrentItem).contentEquals("")) {
                                // means it's a new class
                                Collection<Cls> parentClses = new ArrayList<Cls>();
```

```
                        int maxMatch = 0;
                        parentClses.add(rootCls);
                        Collection<Cls> chkClses = new ArrayList<Cls>(); // get
all the subclasses belong to root except the system classes
                        Collection rootSubClses = rootCls.getSubclasses();
                        Iterator rootSubClsesIterator = rootSubClses.iterator();
                        while(rootSubClsesIterator.hasNext()) {
                                Cls addRootSubCls = (Cls)
rootSubClsesIterator.next();
                                if(addRootSubCls.getName().charAt(0) != ':'){
                                        chkClses.add(addRootSubCls);
                                }
                        }
                        Iterator clsIterator = chkClses.iterator();
                        while(clsIterator.hasNext()) { // find the best location of
this new class
                                Cls testCls = (Cls) clsIterator.next();
                                Collection<Slot> ownSlots =
testCls.getTemplateSlots(); // the slots belong to the testCls
                                if (ownSlots.containsAll(newSlots)) {
                                        if (newSlots.containsAll(ownSlots)) { // the
testCls has the same slots as the new Cls
                                                Collection addSuperClses =
testCls.getDirectSuperclasses();
                                                Iterator addSuperClsIterator =
addSuperClses.iterator();
                                                if(addSuperClses.size() > 1) {
                                                        parentClses.clear();
                                                }

        while(addSuperClsIterator.hasNext()) {
                                                        Cls addSuperCls = (Cls)
addSuperClsIterator.next();

        if(addSuperCls.getName().charAt(0) != ':') {
                                                                parentClses.add((Cls)
addSuperClsIterator.next());
                                                                foundSuperCls = true;
                                                                //break;
                                                        }
                                                }
                                        }
                                } else if (newSlots.containsAll(ownSlots)){
                                        if(maxMatch < ownSlots.size()) {
                                                parentClses.clear();
                                                parentClses.add(testCls);
```

```
                                        maxMatch = ownSlots.size();
                                    }
                                }
                                if (foundSuperCls) break;
                            }
                            status.setText(debugStr);
                            //create new class
                            String clsNoStr = (String) className.getSelectedItem();
                            if(clsNoStr.contentEquals("")) {
                                clsNoStr = "Cls" +
Integer.toString(KB.getClsCount()+ 1);
                            }
                            // need to assign the Root as the new class' parent class
                            Collection<Cls> rootClsAsParent = new ArrayList<Cls>();
                            rootClsAsParent.add(rootCls);
                            Cls newCls = KB.createCls(clsNoStr, rootClsAsParent);

                            Collection<Slot> existingSlots =
newCls.getTemplateSlots();
                            Iterator newSlotsIterator = newSlots.iterator();
                            while(newSlotsIterator.hasNext()) {
                                Slot newSlot = (Slot) newSlotsIterator.next();
                                if(!existingSlots.contains(newSlot)) {
                                    newCls.addDirectTemplateSlot(newSlot);
                                }
                            }

                            //attach the subclasses from the old superclass to new class
                            newCls.removeDirectSuperclass(rootCls);
                            Iterator parentClsesIterator = parentClses.iterator();
                            Collection<Slot> newClsSlots =
newCls.getTemplateSlots();
                            while(parentClsesIterator.hasNext()) {
                                Cls testSuperCls = (Cls) parentClsesIterator.next();
                                newCls.addDirectSuperclass(testSuperCls);
                                Collection subClses =
testSuperCls.getDirectSubclasses();
                                Iterator subClsesIterator = subClses.iterator();
                                while(subClsesIterator.hasNext()) {
                                    Cls testSubCls = (Cls)
subClsesIterator.next();
                                    Collection<Slot> testSubClsSlots =
testSubCls.getTemplateSlots();
                                    String newClsName = newCls.getName();
                                    if(!
newClsName.contentEquals(testSubCls.getName())) {
```

```
        if(testSubClsSlots.containsAll(newClsSlots) && !
newClsSlots.containsAll(testSubClsSlots)) {

        testSubCls.removeDirectSuperclass(testSuperCls);

        testSubCls.addDirectSuperclass(newCls);
                                                        }
                                                }
                                        }
                                }
                                selectedCls = newCls;
                        } else {
                                selectedCls = getClsFromSelection((String)
classNameCurrentItem);
                                Collection<Slot> existingSlots =
selectedCls.getTemplateSlots();
                                Iterator newSlotsIterator = newSlots.iterator();
                                while(newSlotsIterator.hasNext()) {
                                        Slot newSlot = (Slot) newSlotsIterator.next();
                                        if(!existingSlots.contains(newSlot)) {

        selectedCls.addDirectTemplateSlot(newSlot);
                                        }
                                }
                        }


                        // add new instance
                        String emptyInputTestStr = "";
                        String instNoStr = selectedCls.getName() +
Integer.toString(KB.getInstanceCount(selectedCls)+1);
                        instMaxCount++;
                        newInstance = KB.createInstance(instNoStr, selectedCls);
                        Iterator newSlotsIterator = newSlots.iterator();
                        Iterator newValuesIterator = newValues.iterator();
                        while(newSlotsIterator.hasNext()) {
                                Object tempValue = newValuesIterator.next();
                                Slot nextSlotIterator = (Slot) newSlotsIterator.next();

                                if(newSlots.indexOf(nextSlotIterator) !=
newSlots.lastIndexOf(nextSlotIterator)) {
                                        nextSlotIterator.setAllowsMultipleValues(true);
                                }
                                String slotValueTypeStr =
nextSlotIterator.getValueType().toString();
```

```
                                if(slotValueTypeStr.contentEquals("Any")) {
                                        newInstance.addOwnSlotValue(nextSlotIterator,
tempValue.toString());
                                        emptyInputTestStr = emptyInputTestStr +
tempValue.toString();
                                } else if (slotValueTypeStr.contentEquals("Boolean")) {
                                        if((Boolean) tempValue) {

        newInstance.setOwnSlotValue(nextSlotIterator, true);
                                        } else {

        newInstance.setOwnSlotValue(nextSlotIterator, false);
                                        }
                                } else if (slotValueTypeStr.contentEquals("Class")) {
                                        newInstance.addOwnSlotValue(nextSlotIterator,
(Cls) tempValue);
                                        emptyInputTestStr = emptyInputTestStr +
tempValue.toString();
                                } else if (slotValueTypeStr.contentEquals("Float")) {
                                        if(tempValue.toString().length() != 0) {

        newInstance.addOwnSlotValue(nextSlotIterator,
Float.valueOf(tempValue.toString()));
                                                emptyInputTestStr = emptyInputTestStr +
tempValue.toString();
                                        }
                                } else if (slotValueTypeStr.contentEquals("Instance")) {
                                        Collection allowedClses =
nextSlotIterator.getAllowedClses();
                                        Instance result;
                                        if(allowedClses.size() == 0) {
                                                result =
getInstanceFromSelection((String)tempValue, nextSlotIterator, false);
                                                if(result.getName().charAt(0) != ':') {
                                                        Collection<Cls> allowedCls = new
ArrayList<Cls>();

        allowedCls.add(result.getDirectType());

        nextSlotIterator.setAllowedClses(allowedCls);
                                                }
                                        } else {
                                                result =
getInstanceFromSelection((String)tempValue, nextSlotIterator, true);
                                        }
                                        if(((String)
```

```
tempValue).contentEquals(result.getBrowserText())) {
                                                emptyInputTestStr = emptyInputTestStr +
tempValue;

        newInstance.addOwnSlotValue(nextSlotIterator, result);
                                        }
                                } else if (slotValueTypeStr.contentEquals("Integer")) {
                                        if(tempValue.toString().length() != 0) {

        newInstance.addOwnSlotValue(nextSlotIterator,
Integer.valueOf(tempValue.toString()));
                                                emptyInputTestStr = emptyInputTestStr +
tempValue.toString();
                                        }
                                } else if (slotValueTypeStr.contentEquals("String")) {
                                        newInstance.addOwnSlotValue(nextSlotIterator,
tempValue.toString());
                                        emptyInputTestStr = emptyInputTestStr +
tempValue.toString();
                                } else {
                                        newInstance.addOwnSlotValue(nextSlotIterator,
tempValue.toString());
                                        emptyInputTestStr = emptyInputTestStr +
tempValue.toString();
                                }
                        }

                        if (emptyInputTestStr.length() == 0) {
                                KB.deleteInstance(newInstance);
                        }
                }//end of if(newSlots.size() != 0)
        }

        private void clearInput() {
                currentID = 0;
                int i = slotInputPane.getComponentCount();
                for (int j = 1; j <= i; j++) {
                        slotInputPane.remove(slotInputPane.getComponent(0));
                }
        }

   private JComponent createPropertyPane() {
           PropertyPane = new JPanel();
           PropertyPane.setLayout(new BorderLayout(10,10)); //gap = 10

                JPanel addButtonPane = new JPanel();
```

```
                addButtonPane.setLayout(new BorderLayout());
                addSlot = new JButton("Add");
                addButtonPane.add(addSlot, BorderLayout.EAST);

                scrollSlotInputPane = new JScrollPane(createSlotInputPane());


scrollSlotInputPane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCRO
LLBAR_AS_NEEDED);


scrollSlotInputPane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_
SCROLLBAR_NEVER);

            PropertyPane.add(addButtonPane, BorderLayout.NORTH);
            PropertyPane.add(scrollSlotInputPane, BorderLayout.CENTER);
        PropertyPane.setBorder(BorderFactory.createTitledBorder("Properties"));

                addSlot.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent event) {
                                slotInputPane.remove(emptyBox);
                                slotInputPane.add(createSlot());
                                slotInputPane.add(createEmptyBox());
                                slotInputPane.revalidate();
                        }
                });
        return PropertyPane;
    }

          private String clsMeetsSlots() {
                String result = "none";
                Collection<Slot> newSlots;
                Box SB;

                newSlots = new ArrayList<Slot>();
                int i;
                for(i = 0; i < slotInputPane.getComponentCount() - 1; i++) {
                        // Since the last component is "empty box" ==>
getComponentCount() needs to substract 1
                        SB = (Box) slotInputPane.getComponent(i);// slotBox
                        JComboBox slotJB = (JComboBox) SB.getComponent(1); // slot
name JComboBox
                        String slotNameStr = slotJB.getSelectedItem().toString();

                        if(!slotNameStr.contentEquals("")) { // means it is not an empty
slot
```

```
                        Iterator slotIterator = _slots.iterator();
                        while (slotIterator.hasNext()) {
                                Slot existingSlot = (Slot) slotIterator.next();

        if((existingSlot.getName()).contentEquals(slotNameStr)) {
                                        newSlots.add(existingSlot);
                                        break;
                                }
                        }
                }
                }// end of for(checking slots)

                if(newSlots.size() == slotInputPane.getComponentCount() - 1) {
                        Iterator clsIterator = _clses.iterator();
                        while (clsIterator.hasNext()) {
                                Cls testCls = (Cls) clsIterator.next();
                                Collection<Slot> ownSlots = testCls.getTemplateSlots();
                                if (ownSlots.containsAll(newSlots) &&
newSlots.containsAll(ownSlots) && !testCls.isAbstract()) {
                                        result = testCls.getName();
                                        break;
                                }
                        }
                }
                return result;
        }

        private JComponent createSlot() {
                Box newSlotBox = Box.createHorizontalBox();
    newSlotBox.setMaximumSize(slotBoxSize);
    currentID++;
    newSlotBox.setName("slotBox " + currentID);

                newSlotBox.add(createRemoveButton());
                newSlotBox.add(createSlotNameInput());
                newSlotBox.add(createSlotTypeInput());
                newSlotBox.add(createSlotValueOthers());

                return newSlotBox;
        }

        private JComponent createSlot(Slot slot, String slotValueType) {
                Box newSlotBox = Box.createHorizontalBox();
    newSlotBox.setMaximumSize(slotBoxSize);
    currentID++;
    newSlotBox.setName("slotBox " + currentID);
```

```
                JComboBox tempSlotName = (JComboBox) createSlotNameInput();
                JComboBox tempSlotType = (JComboBox) createSlotTypeInput();
                newSlotBox.add(createRemoveButton());
                newSlotBox.add(tempSlotName);
                newSlotBox.add(tempSlotType);
                newSlotBox.add(createSlotValueOthers());

                tempSlotName.setSelectedIndex(slotNameList.indexOf(slot.getName()));

        tempSlotType.setSelectedIndex(slotTypeChoices.indexOf(slotValueType));
                return newSlotBox;
        }

private JComponent createSlotInputPane() {
   slotInputPane = new JPanel();
   slotInputPane.setLayout(new BoxLayout(slotInputPane, BoxLayout.Y_AXIS));

                slotInputPane.add(createSlot());
                slotInputPane.add(createEmptyBox());
                slotInputPane.revalidate();

   return slotInputPane;
}

private JComponent createRemoveButton() {
   JButton remove = new JButton("Remove");
   remove.setName("remove " + currentID);
   remove.setMaximumSize(buttonSize);
   remove.setPreferredSize(buttonSize);
   remove.setMinimumSize(buttonSize);
   remove.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
                slotInputPane.remove(((JComponent) e.getSource()).getParent());

                                String testResult = clsMeetsSlots();
                                if(testResult.contentEquals("none")) {
                                        className.setSelectedIndex(0);
                                } else {

        className.setSelectedIndex(clsNameList.indexOf(testResult));
                                }

                                slotInputPane.revalidate();
      }
   });
```

```
        return remove;
    }

    private JComponent createSlotNameInput() {
            JComboBox slotName = new JComboBox(slotNameList);
        slotName.setMaximumSize(slotNameSize);
                slotName.setPreferredSize(slotNameSize);
        slotName.setMinimumSize(slotNameSize);
            slotName.setEditable(true);
                slotName.setName("slotName" + currentID);

                slotName.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                JComboBox source = (JComboBox) e.getSource();
                                String currentItem = (String) source.getSelectedItem();
                                if(e.getActionCommand().equals("comboBoxEdited")) {
                                        if((currentItem.length() != 0)) {
                                                if(!slotNameList.contains(currentItem)) {
                                                        source.addItem(currentItem);
                                                }
                                        }
                                        String testResult = clsMeetsSlots();
                                        if(testResult.contentEquals("none")) {
                                                className.setSelectedIndex(0);
                                        } else {

            className.setSelectedIndex(clsNameList.indexOf(testResult));
                                        }
                                }
                        }
                });

                slotName.addItemListener(new ItemListener() {
                        public void itemStateChanged(ItemEvent ie) {
                                JComboBox source = (JComboBox) ie.getSource();
                                String currentItem = (String) source.getSelectedItem();
                                if(currentItem.length()!= 0) {
                                        int i = 0;
                                        while(i < slotList.size() && !
currentItem.equals((slotList.get(i)).getName())) {
                                                i++;
                                        }

                                        if(i < slotList.size()) {
                                                String slotValueType =
((slotList.get(i)).getValueType()).toString();
```

```
                                                if(slotValueType.equals("Cls"))
slotValueType = "Class";
                                                i =
slotTypeChoices.indexOf(slotValueType);
                                                JComponent sourceParent = (JComponent)
source.getParent();
                                                ((JComboBox)
sourceParent.getComponent(2)).setSelectedIndex(i); // setting the slot type
                                        } else {
                                                JComponent sourceParent = (JComponent)
source.getParent();
                                                ((JComboBox)
sourceParent.getComponent(2)).setSelectedIndex(0); // setting the slot type
                                        }
                                }
                        }
                });

                slotName.addPopupMenuListener(new PopupMenuListener() {
                        public void popupMenuWillBecomeVisible(PopupMenuEvent
pme) {
                                JComboBox source = (JComboBox) pme.getSource();
                                String currentItem = (String) source.getSelectedItem();
                                source.addItem(currentItem);
                        //have to do these 2 lines since
                                source.removeItemAt(source.getItemCount() - 1);
                // the comboboxes won't refresh by themselves

        source.setSelectedIndex(slotNameList.indexOf(currentItem));
                                JComponent sourceParent = (JComponent)
source.getParent();
                                ((JComboBox)
sourceParent.getComponent(2)).setSelectedIndex(0); // setting the slot type
                        }
                        public void popupMenuWillBecomeInvisible(PopupMenuEvent
pme) {
                                String testResult = clsMeetsSlots();
                                if(testResult.contentEquals("none")) {
                                        className.setSelectedIndex(0);
                                } else {

        className.setSelectedIndex(clsNameList.indexOf(testResult));
                                }
                        }
                        public void popupMenuCanceled(PopupMenuEvent pme) {
                        }
```

```
                });

        return slotName;
    }

    private JComponent createSlotTypeInput() {
        JComboBox slotType = new JComboBox(slotTypeChoices);
      slotType.setMaximumSize(slotTypeSize);
            slotType.setPreferredSize(slotTypeSize);
      slotType.setMinimumSize(slotTypeSize);
        slotType.addItemListener(new ItemListener() {
            public void itemStateChanged(ItemEvent e) {
                Box sourceParent = (Box) ((JComponent)
e.getSource()).getParent();
                String item = (String) e.getItem();
                    Component changeItem = sourceParent.getComponent(3);
                    Component slotComboList =
sourceParent.getComponent(1);
                    String slotSelection = (String) ((JComboBox)
slotComboList).getSelectedItem();
                    Slot currentSlot = getSlotFromSelection(slotSelection);

                if(item.contentEquals("Boolean")) {
                        sourceParent.remove(changeItem);
                        sourceParent.add(createSlotValueBoolean());
                } else if(item.contentEquals("Class")) {
                        sourceParent.remove(changeItem);
                        sourceParent.add(createSlotValueClass());
                } else if(item.contentEquals("Instance")) {
                        sourceParent.remove(changeItem);
                        Vector<String> instSelection = new
Vector<String>();
                        instSelection.addElement("");
                        if(currentSlot.getAllowedClses().size() != 0) {
                            Iterator clsIterator =
currentSlot.getAllowedClses().iterator();
                            while(clsIterator.hasNext()) {
                                Cls allowedCls = (Cls)
clsIterator.next();
                                Iterator instIterator =
allowedCls.getInstances().iterator();
                                while(instIterator.hasNext()) {
                                    instSelection.add(((Instance)
instIterator.next()).getBrowserText());
                                }
                            }
```

```
        sourceParent.add(createSlotValueInstance(instSelection));
                                        } else {
                                                Iterator instIterator = instanceList.iterator();
                                                while(instIterator.hasNext()) {
                                                        Instance tempInst = (Instance)
instIterator.next();

        instSelection.add(tempInst.getBrowserText());
                                                }

        sourceParent.add(createSlotValueInstance(instSelection));
                                        }
                                } else if(item.contentEquals("Symbol")) {
                                        sourceParent.remove(changeItem);
                                        Vector<String> symbolSelection = new
Vector<String>();
                                        Iterator symbolIterator =
currentSlot.getAllowedValues().iterator();
                                        while(symbolIterator.hasNext()) {
                                                symbolSelection.add((String)
symbolIterator.next());
                                        }

        sourceParent.add(createSlotValueSymbol(symbolSelection));
                                } else {
                                        sourceParent.remove(changeItem);
                                        sourceParent.add(createSlotValueOthers());
                                }
                                slotInputPane.revalidate();
                        }
                });
                return slotType;
        }


        private Cls getClsFromSelection(String clsSelection) {
                Iterator clsIterator = _clses.iterator();
                Cls gotCls = (Cls) clsIterator.next();
                while (clsIterator.hasNext()) {
                        if(clsSelection.contentEquals(gotCls.getName())) {
                                break;
                        } else {
                                gotCls = (Cls) clsIterator.next();
                        }
                }
```

```
            return gotCls;
        }

        private Instance getInstanceFromSelection(String instSelection, Slot ownedSlot,
boolean hasAllowedClses) {
            Iterator instIterator = _instes.iterator();
            Instance gotInst = (Instance) instIterator.next();
            if(hasAllowedClses) {
                Collection ownedClses = ownedSlot.getAllowedClses();
                while (instIterator.hasNext()) {
                    if(instSelection.contentEquals(gotInst.getBrowserText())
&& ownedClses.contains(gotInst.getDirectType())) {
                        break;
                    } else {
                        gotInst = (Instance) instIterator.next();
                    }
                }
            } else {
                while (instIterator.hasNext()) {
                    if(instSelection.contentEquals(gotInst.getBrowserText())) {
                        break;
                    } else {
                        gotInst = (Instance) instIterator.next();
                    }
                }
            }

            return gotInst;
        }

        private Slot getSlotFromSelection(String slotSelection) {
            Iterator slotIterator = _slots.iterator();
            Slot gotSlot = (Slot) slotIterator.next();
            while (slotIterator.hasNext()) {
                if(slotSelection.contentEquals(gotSlot.getName())) {
                    break;
                } else {
                    gotSlot = (Slot) slotIterator.next();
                }
            }
            return gotSlot;
        }

    private JComponent createSlotValueOthers() {
            JTextField slotValueOthers = new JTextField();
            slotValueOthers.addMouseListener(new MouseInputAdapter() {
```

```
                    public void mouseClicked(MouseEvent me) {
                    }
                });

        return slotValueOthers;
    }

    private JComponent createSlotValueBoolean() {
        JComboBox slotValueBoolean = new JComboBox(booleanChoices);
        slotValueBoolean.setEditable(false);

        return slotValueBoolean;
    }

    private JComponent createSlotValueClass() {
        JComboBox slotValueClass = new JComboBox(clsNameList);
        slotValueClass.setEditable(false);

        return slotValueClass;
    }

    private JComponent createSlotValueInstance(Vector<String> instSelection) {
        JComboBox slotValueInstance = new JComboBox(instSelection);
        slotValueInstance.setEditable(false);

        return slotValueInstance;
    }

    private JComponent createSlotValueSymbol(Vector<String> symbolSelection) {
        JComboBox slotValueSymbol = new JComboBox(symbolSelection);
        slotValueSymbol.setEditable(true);

        return slotValueSymbol;
    }
    private Box createEmptyBox() {
       emptyBox = Box.createHorizontalBox();
       emptyBox.add(Box.createHorizontalGlue());
       emptyBox.add(Box.createHorizontalStrut(250));
       return emptyBox;
    }

    // this method is useful for debugging
    public static void main(String[] args) {
       edu.stanford.smi.protege.Application.main(args);
    }
}
```

APPENDIX B

DATA LIST

REFERENCES

[1] Sachs, Eliza. “Getting Started with ProtÈgÈ-Frames”, 2006.
protege.stanford.edu/doc/tutorial/get_started/get-started.pdf

[2] Ying Ding, Schubert Foo, “Ontology Research and Development Part 1 – A Review of Ontology Generation”, 2002.

http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=F53A2BD31E7446D4358E1C61CACD6BFD?doi=10.1.1.86.4634&rep=rep1&type=pdf

[3] Natalya Fridman Noy, Ray W. Fergerson, Mark A. Musen, “The knowledge model of ProtÈgÈ-2000 combining interoperability and flexibility”, 2000.
www.pms.ifi.lmu.de/mitarbeiter/ohlbach/Ontology/Protege/SMI-2000-0830.pdf

[4] John H. Gennari, Mark A. Musen, Ray W. Fergerson, William E. Grosso, Monica, Crubezy, Henrik Eriksson, Natalya F. Noy, Samson W. Tu, “The Evolution of ProtÈgÈ: An Environment for Knowledge-Based Systems Development”, 2000.
citeseer.ist.psu.edu/545954.html

[5] Gerd Stumme, Alexander Maedche, “FCA-Marge: Bottom-Up Merging of Ontologies”, 2001.
www.dit.unitn.it/~accord/RelatedWork/Matching/**FCA**01.pdf

[6] Peter Haase, Johanna Völker, “Ontology Learning and Reasoning – Dealing with Uncertainty and Inconsistency”, 2005.
www.aifb.uni-karlsruhe.de/WBS/jvo/publications/**Uncertainty**_2005.pdf

[7] Harith Alani, Sanghee Kim, David E. Millard, Mark J. Weal, Wendy Hall, Paul H. Lewis, Nigel R. Shadbolt, “Automatic Ontology-based Knowledge Extraction from Web Documents ”, 2003.
eprints.aktors.org/105/01/IEEE-Artequakt.pdf

[8] Paul E. Vet, Nicolaas J. Mars, “Bottom-up construction of ontologies: the case of an ontology of pure substances”, 1995.
portal.acm.org/citation.cfm?id=627317.627926&coll=GUIDE&dl=GUIDE&CFID=9740613&CFTOKEN=32340529

[9] Natalya F. Noy, Deborah L. McGuinness, “Ontology Development 101, A Guide to Creating Your First Ontology”, 2000.
protege.stanford.edu/publications/ontology_development/ontology101.pdf

[10] Peter Burmeister, “Formal Concept Analysis with ConImp: Introduction to the Basic Features”, 2003.
www.mathematik.tu-darmstadt.de/~burmeister/ConImpIntro.pdf

[11] Natalya Fridman Noy, Mark A. Musen, “Algorithm and Tool for Automated Ontology Merging and Alignment”, 2005.
dit.unitn.it/~p2p/RelatedWork/Matching/SMI-2000-0831.pdf

[12] Alexander Maedche, Steffen Staab, “Ontology Learning for the Semantic Web”, 2001.
www.aifb.uni-karlsruhe.de/WBS/sst/Research/Publications/ieee_semweb.pdf

[13] Mary Elaine Califf, Raymond J. Mooney, “Relational learning of pattern-match rules for information extraction”, 1999.
www.aclweb.org/anthology-new/W/W97/W97-1002.pdf

[14] York Sure, Juergen Angele, and Steffen Staab, “OntoEdit: Multifaceted Interfencing for Ontology Engineering”, 2003.

[15] Pat Hayes, Raul Saavedra, Thomas Reichherzer, “A Collaborative Development Environment for Ontologies (CODE)”, 2005.
www.cs.indiana.edu/~treichhe/**code**.pdf

[16] Asuncion Gomez-Perez, “Knowledge Sharing and Reuse: Ontolgies and Applications”, 1999.
icc.mpei.ru/documents/00000831.pdf

[17] Karl Erich Wolff, “A first course in Formal Concept Analysis”, 1993.
www.fbmn.fh-darmstadt.de/~wolff/Publikationen/A**_First_Course_in_Formal_Concept_Analysis**.pdf

[18] Frank Buchli, “A Short FCA Primer”, 2003.

[19] G Tao, “Using Formal Concept Analysis (FCA) for Ontology Structuring and

Building", 1992.
www.springerlink.com/index/926053735j142745.pdf

[20] Asuncion Gomez-Perez, V. Richard Benjamins, "Applications of Ontologies and Problem-Solving Methods", 1999.
www.aaai.org/ojs/index.php/aimagazine/article/viewFile/1445/1344

[21] Bastian Wormuth, Peter Becker, "Introduction to Formal Concept Analysis", 2004.
www.wormuth.info/ICFCA04/Introduction_to_FCA_ICFCA2004.pdf

[22] Philipp Cimiano, Johanna Volker, "A Framework for Ontology Learning and Data-driven Change Discovery", 2005.
citeseer.ist.psu.edu/731219.html