NTNU

Det skapende universitet

Protégé

Contents

1. Introduction

Using ontology it is possible to capture knowledge about some domain. An ontology describes the concepts in the domain and also the relationships that hold between those concepts. Protégé is a free, open source ontology editor and knowledge-base framework. The Protégé platform supports two main ways of modeling ontology via the Protégé-Frames and Protégé-OWL editors. Protégé ontology can be exported into a variety of formats including RDF(S), OWL, and XML Schema. One of the advantages of using protégé in complex concepts is reasoner, which can check whether or not all of the statements and definitions in the ontology are consistent and can also recognize which concepts fit under which definitions.

ACC system stands for Adaptive Cruise Control system, and it is a system to control vehicle speed adaptively to a forward vehicle by using information about: (1) ranging to forward vehicles (2) the motion of the subject (ACC equipped) vehicle and (3) driver commands. Based on these information acquired, the controller sends commands to actuators for carrying out its longitudinal control strategy and it also sends status information to the driver.

This guide introduces Protégé 4 for creating OWL ontology. As an example, during this guide, ontology for ACC System will be created and the creation process is explained step by step.

2. Requirement

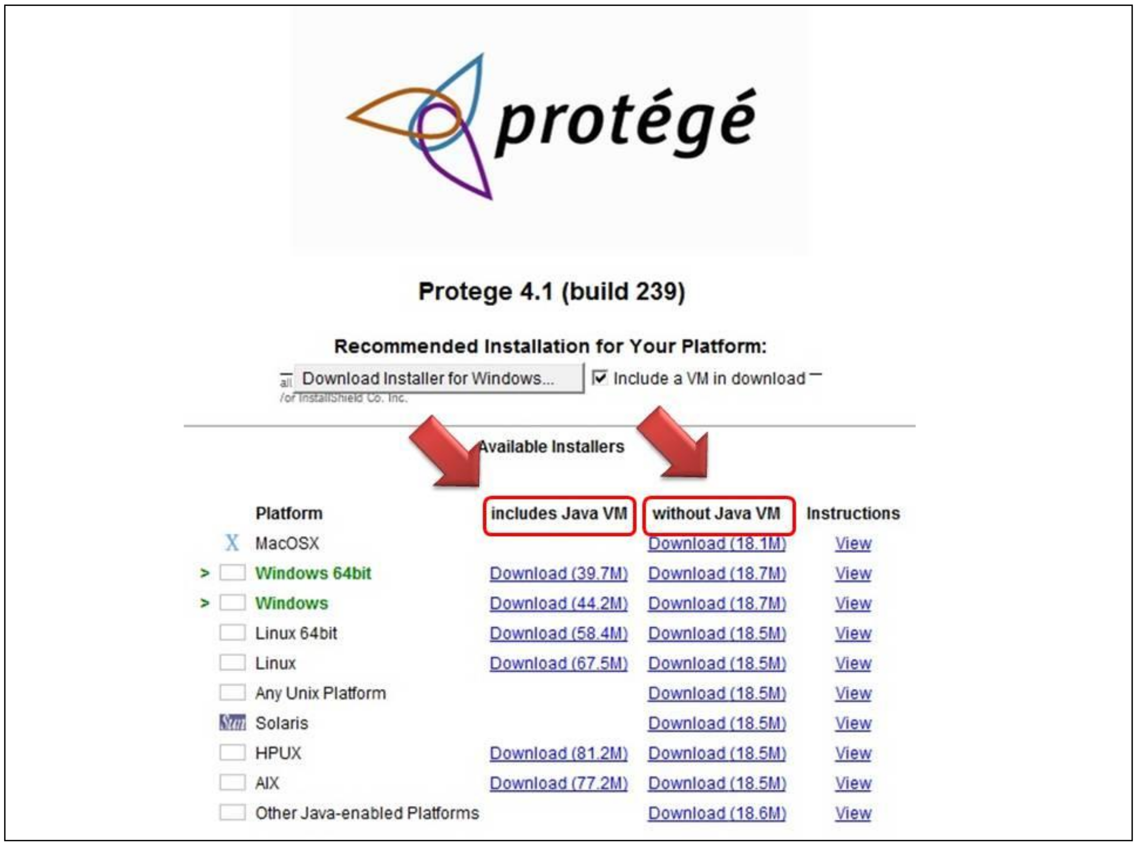
In order to follow this tutorial you need to install Protégé 4.1. The web page you can download the Protégé and the instruction for installation are described in section 3. To see your classification hierarchies of the domain more visualised, you can use the OWLViz plugin that is also described in section 3. You can install more Protégé Plugins which are available via the Protégé ' s website.

3. Protégé Installation

3.1. Download protégé

At the first step, in order to download Protégé go to this webpage , then select the compatible Protégé, based on your platform.

As for some platforms in order to run Protégé, it is required version 1.5 of the Java Virtual Machine has been installed before, please choose from 'Includes Java VM', which will install JVM on your computer simultaneously, otherwise select from 'without Java VM' **(Figure 3.1)**.


protégé
Protege 4.1 (build 239)
Recommended Installation for Your Platform:
Download Installer for Windows...
Include a VM in download
Available Installers

Platform	includes Java VM	without Java VM	Instructions
MacOSX		Download (18.1M)	View
Windows 64bit	Download (39.7M)	Download (18.7M)	View
Windows	Download (44.2M)	Download (18.7M)	View
Linux 64bit	Download (58.4M)	Download (18.5M)	View
Linux	Download (67.5M)	Download (18.5M)	View
Any Unix Platform		Download (18.5M)	View
Solaris		Download (18.5M)	View
HPUX	Download (81.2M)	Download (18.5M)	View
AIX	Download (77.2M)	Download (18.5M)	View
Other Java-enabled Platforms		Download (18.6M)	View

Figure (3.1)

3.2 Install Protégé

After downloading Protégé, double-click **'install_Protégé_4.1.exe'** then:

1. A window will open, and you have to follow the installation steps by selecting 'Next'.

2. In the 'Choose Install Folder' step select the path you want to install the Protégé and click 'Next'.

3. In the 'Choose Shortcut Folder' step select a place you want to create product icon and then click 'Next' button and follow.

4. In the 'Choose Java Virtual Machine' step, select the first option 'Use the Java VM installed with this application'.

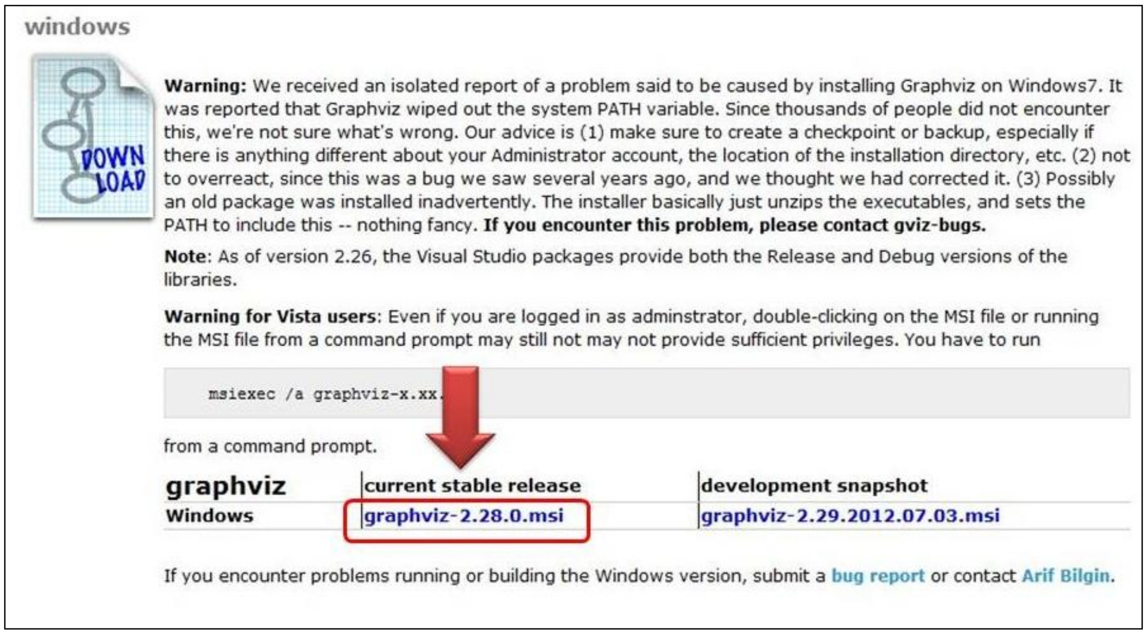
5. In the next step please read the summary and select install button.

6. Then the installation will start automatically, at the end press 'Done' button to finish installation.

3.3 Graphviz Installation

Graphviz has been designed to represent your classification hierarchies of the domain more visualized. If you want to use Graphviz, you have to install it. Graphviz installation can be start, after installing Protégé.

First you have to go to this web page and based on your platform, download the compatible installation package of Graphviz**.** For example **(Figure 3.2)**, represents the available installation package for windows.


windows

Warning: We received an isolated report of a problem said to be caused by installing Graphviz on Windows7. It was reported that Graphviz wiped out the system PATH variable. Since thousands of people did not encounter this, we're not sure what's wrong. Our advice is (1) make sure to create a checkpoint or backup, especially if there is anything different about your Administrator account, the location of the installation directory, etc. (2) not to overreact, since this was a bug we saw several years ago, and we thought we had corrected it. (3) Possibly an old package was installed inadvertently. The installer basically just unzips the executables, and sets the PATH to include this -- nothing fancy. **If you encounter this problem, please contact gviz-bugs.**

Note: As of version 2.26, the Visual Studio packages provide both the Release and Debug versions of the libraries.

Warning for Vista users: Even if you are logged in as adminstrator, double-clicking on the MSI file or running the MSI file from a command prompt may still not may not provide sufficient privileges. You have to run

```
msiexec /a graphviz-x.xx.
```

from a command prompt.

graphviz	current stable release	development snapshot
Windows	graphviz-2.28.0.msi	graphviz-2.29.2012.07.03.msi

If you encounter problems running or building the Windows version, submit a bug report or contact Arif Bilgin.

Figure (3.2)

1. After downloading, double-click on the file that you have downloaded and select 'Run'.

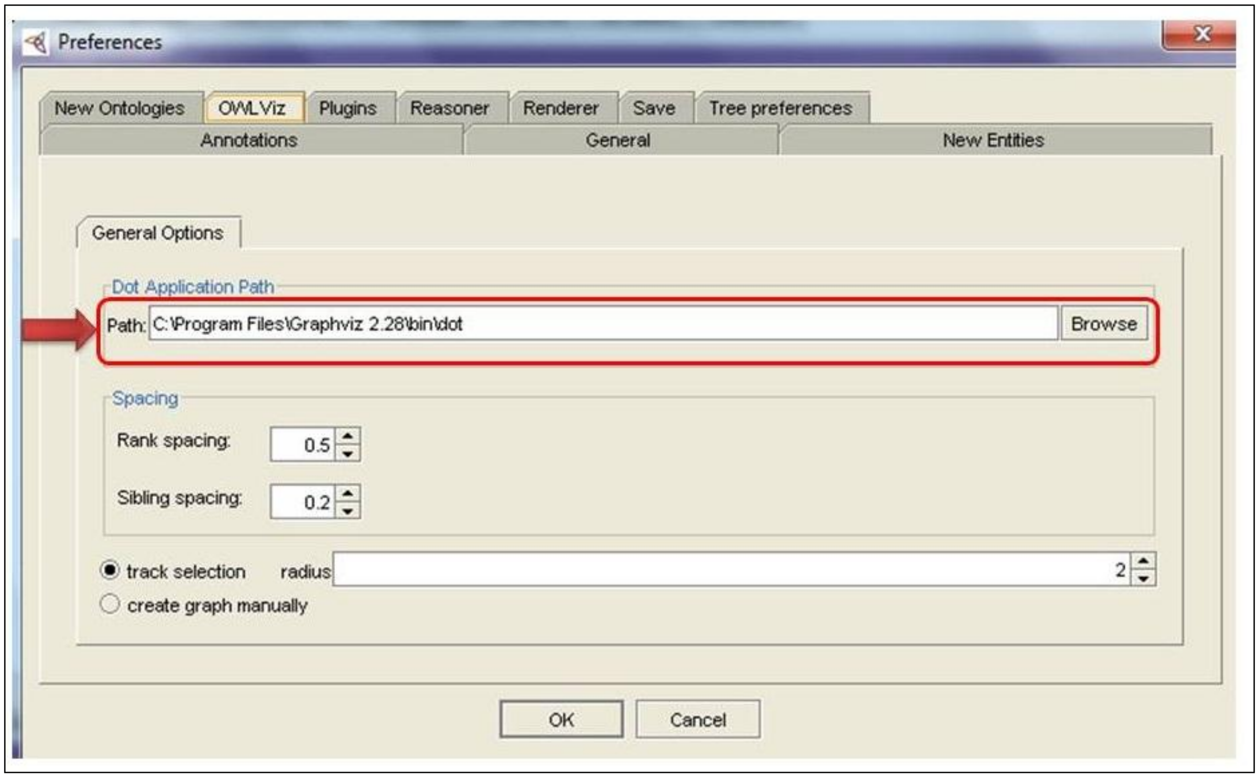
2. Follow the installation steps and pay attention to the path that you select as an installation folder. This path will be required later.

When **Graphviz** installation completed, Protégé will make an educated guess about where to find graphviz depending on your operating system, but if it does not get it right you will need to configure this yourself:

These settings can be finding in the File ---> preferences, OWLViz tab. As shown in the

(Figure 3.3), specify the path where you have installed Graphviz to the dot executable file in the appropriate place.

Example: C:\Program Files\Graphviz 2.28\bin\dot



Path: C:\Program Files\Graphviz 2.28\bin\dot

Rank spacing: 0.5

Sibling spacing: 0.2

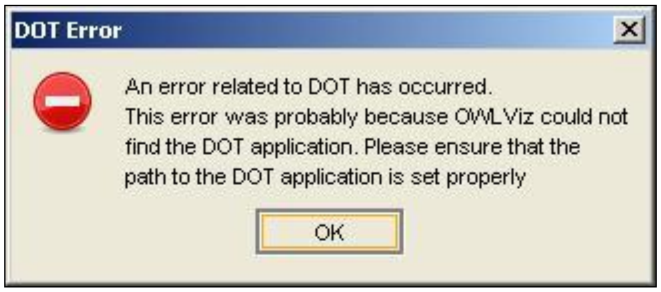
[✓ track selection] radius 2

create graph manually

Figure (3.3)

The following error may be occurs when you want to use OWLViz **(Figure 3.4)**, if:

1) You did not installed Graphviz and/or
2) You did not specify the path where you have installed Graphviz in the preferences.



An error related to DOT has occurred.
This error was probably because OWLViz could not find the DOT application. Please ensure that the path to the DOT application is set properly

Figure (3.4)

4. Overview

Ontologies are generally used to describe some domain of interest by capturing knowledge about the domain, containing a set of concepts and the relationship between them. The Web Ontology Language (OWL) is the most recent standard language for modeling ontologies which is developed by [W3C Web Ontology Working Group]. The current version of OWL which has published as 'OWL 2' in 2009 is presented with three different level of expressiveness including OWL Lite, OWL DL and OWL Full. Despite researchers' effort to create languages in the hope of developing a Semantic Web, there is still lack of understanding of a standard. According to definition of Protégé in its website, 'Protégé is a free, open source ontology editor and knowledge-base framework' which was developed by Stanford Center for Biomedical Informatics Research at the Stanford University School of Medicine, in order to editing and designing models in various Semantic Web Languages.

4.1. Components of OWL Ontology

The components used in OWL have equivalent substitutes in Protégé. But terminologies used to describe these components are not same in OWL and Protégé. The OWL consists of three elements of Individuals, Properties, and Classes, which in Protégé frames respectively are defined as Instances, Slots and Classes.

Individuals

In OWL Individual represent basic components of ontology which known as objects in the domain of interest. Ii addition to use different term to represent object, there is an important difference between OWL and Protégé that OWL does not use the Unique Name Assumption (UNA). In other words, in OWL an individual may be referred by more than one name. For instance in the domain of Operating Systems, 'Windows', 'Microsoft Windows', 'Windows os' and 'Microsoft's os' might all refer to the same individual. So it should be precisely stated that different individual are the same as each other or different to each other. In this tutorial individuals are represented as squares in diagrams. In (**Figure 4.1**) some individuals of some domain are represented.

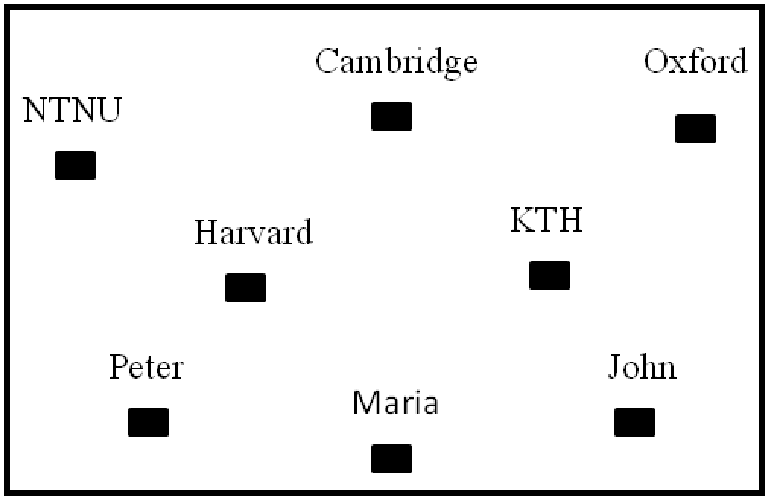


Figure (4.1)

4.2. Properties

Properties are binary relations. It means every relation is between only two things. In OWL three distinguished relationships are defined:

- **Object properties:** The relationship between two individuals. For example, the property studyIn might connect the individual Peter to the individual NTNU, or the property isFriend might connect individual Maria to individual John.
- **Datatype properties:** The relationship between an individual and a data values.
- **Annotation properties:** Annotation properties are used to add metadata (information of data about data) to classes, individuals and datatype properties.

4.3. Classes

In OWL classes are considered as sets containing individuals. All the individuals in a class has some properties in common which these properties are exactly requirements for those individuals that are members of that class. For example the University class contains all universities in our domain of interest like NTNU, Harvard, KTH and so on. (**Figure 4.2**) shows a representation of some classes containing individuals and the relationship between individuals. In this figure classes are represented as circles and properties as directed arrows. Classes can be organized into a hierarchy of superclasses-subclasses. For instance consider the classes

University and Department - all the Departments are members of University so being department implies that it is a member of an University. (**Figure 4.3**) shows a representation of a class hierarchy.

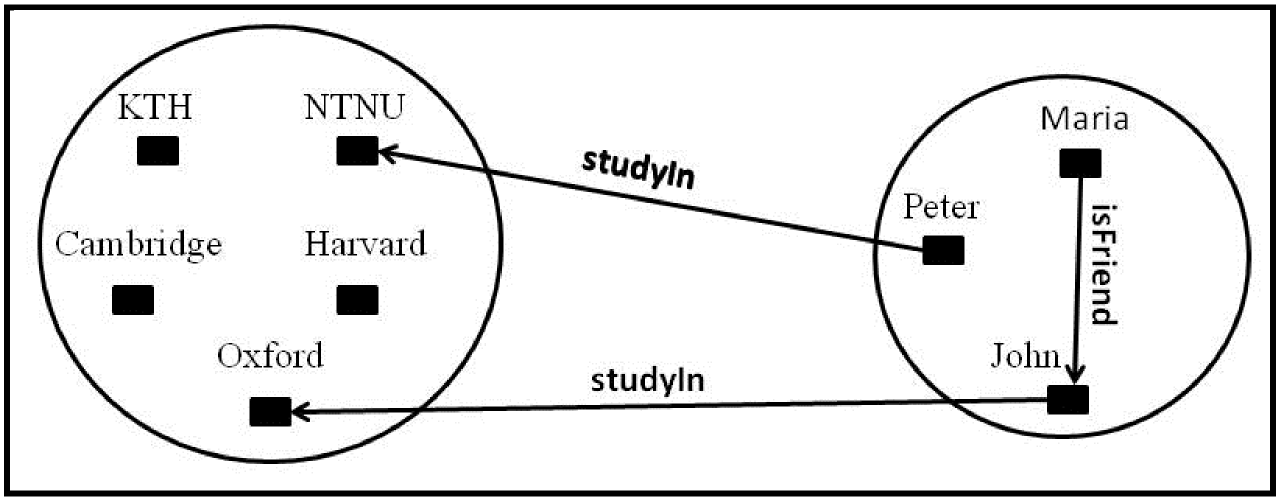


Figure (4.2)

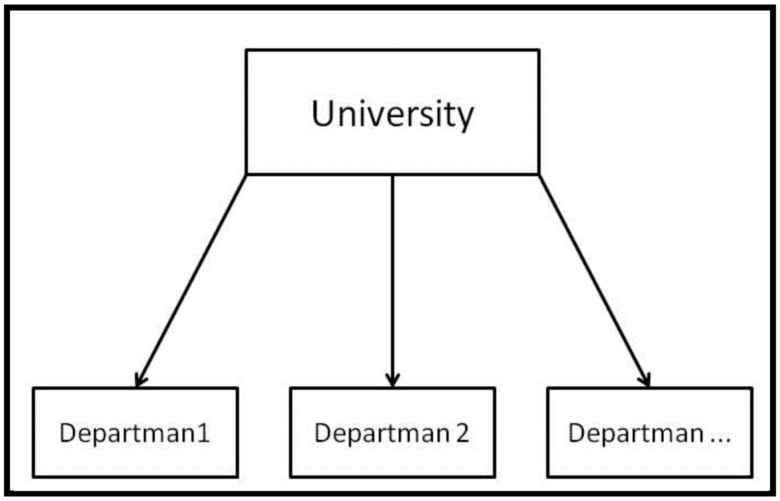


Figure (4.3)

5. How to Build OWL Ontology (ACC Example)

5.1. Create a New Owl Ontology

We previously described what ACC system is and how it works; here we describe how to define the ontology of ACC system by protégé. We will capture different concepts and relationships in the ACC domain. Create a new OWL Ontology:

1. Start Protégé.

2. When the 'Welcome to Protégé' dialog box appears, press the 'Create New OWL Ontology'.

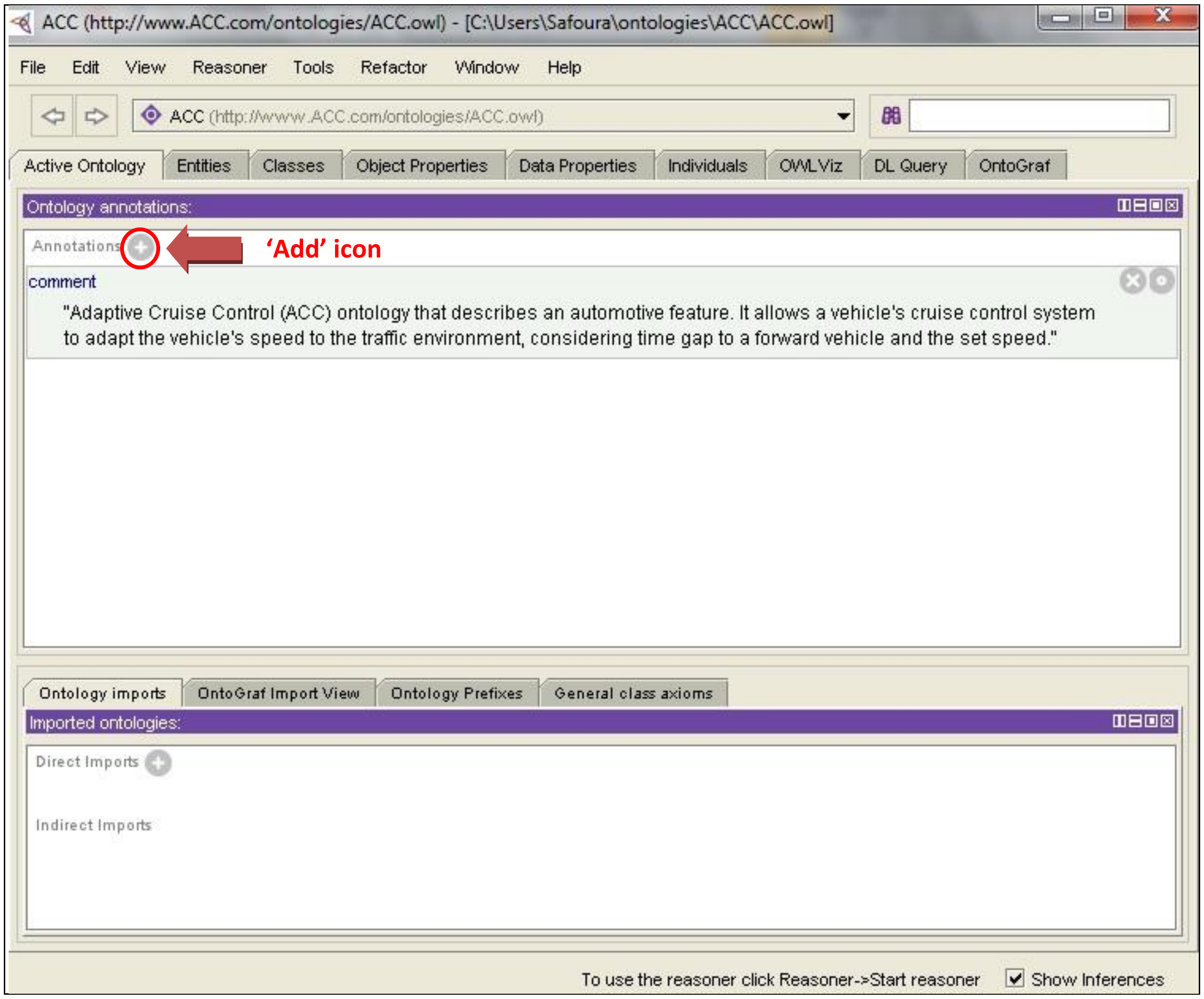
3. A 'Create Ontology URI Wizard' will appear. Every ontologies is named using a Unique Resource Identifier (URI). Replace the default URI with 'http://www.ACC.com/ontologies/ACC.owl' and press 'continue'.

4. You may want to save your Ontology to a file on your PC. So you can browse your hard disk and save your ontology to a new file, you can name your file 'ACC.owl'.

5. After pressing 'continue', you can select the format in which the ontology will be saved (e.g. You can select 'OWL functional syntax') then press 'Finish'.

Now you have been created a new empty Protégé file. The 'Active Ontology Tab' shown in **(Figure 5.1)** will be visible.

In the active ontology tab we can add annotations on the ontology such as some comments to describe ACC system ontology. Press 'Add' icon and write some comment. For example: *"Adaptive Cruise Control (ACC) ontology that describes an automotive feature. It allows a vehicle's cruise control system to adapt the vehicle's speed to the traffic environment, considering time gap to a forward vehicle and the set speed. "*


ACC (http://www.ACC.com/ontologies/ACC.owl) - [C:\Users\Safoura\ontologies\ACC\ACC.owl]
File Edit View Reasoner Tools Refactor Window Help
ACC (http://www.ACC.com/ontologies/ACC.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Ontology annotations:
Annotations
'Add' icon
comment
Ontology imports OntoGraf Import View Ontology Prefixes General class axioms
Imported ontologies:
Direct Imports
Indirect Imports
To use the reasoner click Reasoner->Start reasoner
Show Inferences

"Adaptive Cruise Control (ACC) ontology that describes an automotive feature. It allows a vehicle's cruise control system to adapt the vehicle's speed to the traffic environment, considering time gap to a forward vehicle and the set speed."

Figure (5.1)

5.2. Named Classes

Ontology contains classes and various relations between these classes. In Protégé 4, editing of classes is carried out using the 'Classes tab' shown in (**Figure 5.2)** The empty ontology contains one class called 'Thing'. All classes that we create will become subclasses of Thing.

Now we want to start defining some classes to the ACC ontology in order to clarify how ACC system works.

Create classes 'ACC_System' and 'Time_Gap':

1. Ensure that the 'Classes tab' is selected.

2. **2.** Press the ‘Add' icon shown in **(Figure 5.2)**. This icon creates a new class as a subclass of the selected class. So if you select ‘Thing’ class, and then press ‘add Subclass’ you will add a subclass of ‘Thing’.

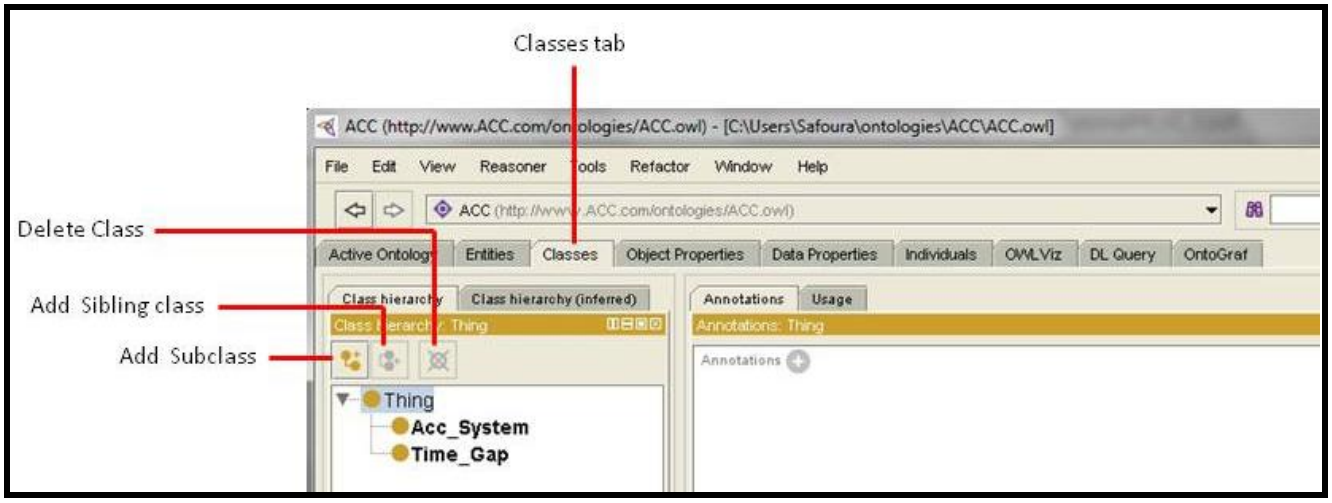


Figure (5.2)

3. **3.** A dialog will appear for you to name your class, enter ACC_System and press ‘Ok’ **(Figure 5.3)**.

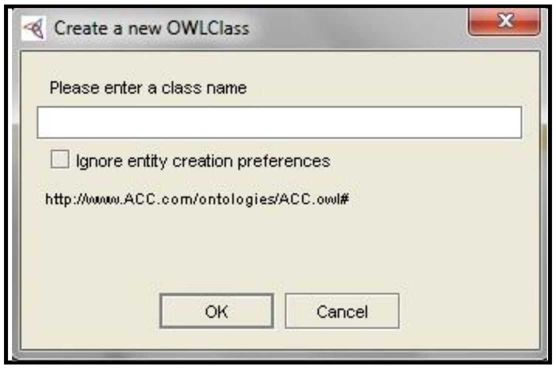

Create a new OWLClass
Please enter a class name
Ignore entity creation preferences
http://www.ACC.com/ontologies/ACC.owl#
OK
Cancel

Figure (5.3)

4. **4.** Repeat the previous steps to add the class ‘Time_Gap’, ensuring that Thing is selected before the 'Add' icon is pressed so these classes are created as subclasses of Thing.

Instead of selecting ‘Thing’ class, you can also select ACC_System class and press the icon for ‘add sibling class’ which is next to ‘add class’ icon **(Figure 5.2)**. This is another way to create the class ‘Time_Gap’ as subclass of ‘Thing’ and a sibling class of ‘ACC_System’. The class hierarchy should look like **(Figure 5.2)**.

To make it more understandable, we recommend that all class names start with a capital letter and not contain spaces, you can also use underline to join words together. For example Time_Gap.

5.3. Disjoint Classes

OWL Classes are assumed to ‘overlap’; it means that by default individuals from one class may be also from another class. Therefore we cannot assume that an individual is only a member of a particular class simply because it has been asserted to be a member of that class.

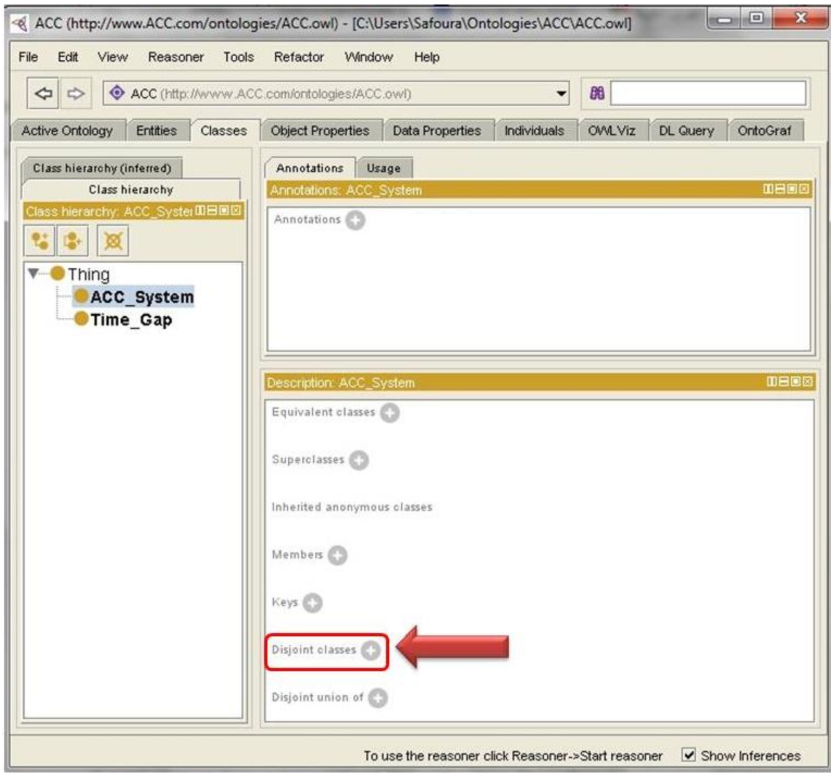
In order to ‘separate’ a group of classes we must make them **disjoint** from one another. This ensures that an individual who has been asserted to be a member of one of the classes in the group cannot be a member of any other classes in that group.

In ACC example, ACC_System and Time_Gap classes should be disjoint from each other. In ACC domain it would make no sense for an individual to be an ACC_System and a Time_Gap! This means that it is not possible for an individual to be a member of a combination of these classes.

So now, we need to make the ACC_System and Time_Gap classes disjoint from each other.

1. Select the ACC_System in the class hierarchy.

2. Press the ‘Disjoint classes' at the bottom in the ‘class description’ view **(Figure 5.4)**.


ACC (http://www.ACC.com/ontologies/ACC.owl) - [C:\Users\Safoura\Ontologies\ACC\ACC.owl]
File Edit View Reasoner Tools Refactor Window Help
ACC (http://www.ACC.com/ontologies/ACC.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Class hierarchy (inferred)
Class hierarchy
Class hierarchy: ACC_System
Thing
ACC_System
Time_Gap
Annotations Usage
Annotations: ACC_System
Annotations
Description: ACC_System
Equivalent classes
Superclasses
Inherited anonymous classes
Members
Keys
Disjoint classes
Disjoint union of
To use the reasoner click Reasoner->Start reasoner
Show Inferences

(Figure 5.4)

This will bring up a window where you can select multiple classes to be disjoint from ACC_System class. If in this window, ACC_System and Time_Gap classes are not present, double click 'Thing' class to make ACC_System and Time_Gap classes become visible **(Figure 5.5)**.

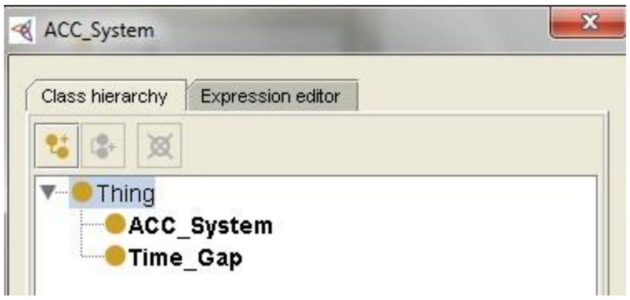


Figure (5.5)

3. The disjoint classes' window now displays ACC_System and Time_Gap classes. Select the class 'Time_Gap' and then 'Ok'.

Notice that the disjoint classes view displays the class that is now disjoint to ACC_System, which is Time_Gap **(Figure 5.6)**

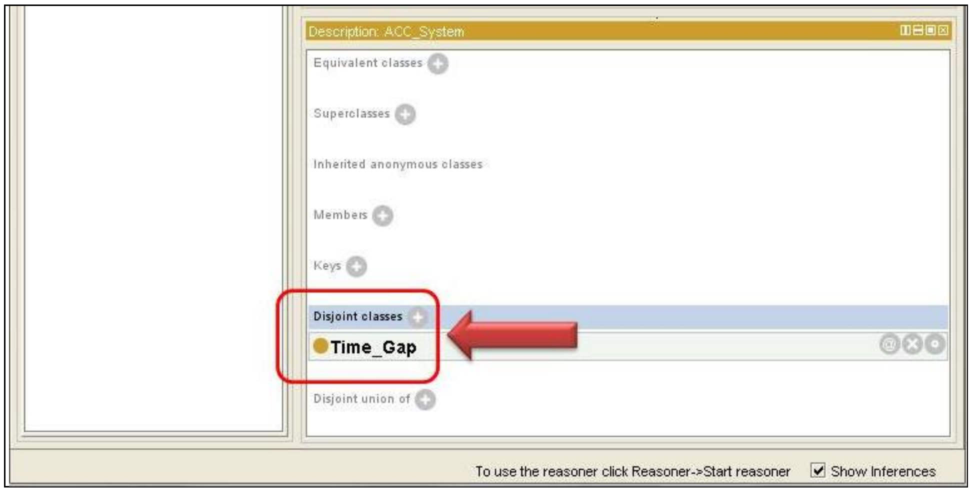


Figure (5.6)

5.4. Using Create Class Hierarchy to Create Classes

In this section we will use the 'Create Class Hierarchy' tool to add some more subclasses of the class 'Thing'. This is a useful way to add classes and subclasses when there are many classes and subclasses to add.

1. Select the class 'Thing' in the class hierarchy.

2. From the 'Tools' menu on the Protégé menu bar select 'Create Class Hierarchy...'.

3. The tools shown in **(Figure 5.7)** will appear. Here we have to select the class, which we want to create subclasses under that. Since we preselected the class 'Thing', the first class at the top of the tool should be prompting us to create the classes under the class 'Thing'. If we want to create subclasses for another class we have to select it here. All the classes under 'Thing' will become visible by double clicking on 'Thing' class. Because we want to add subclasses for 'Thing', so we just select 'Thing'.

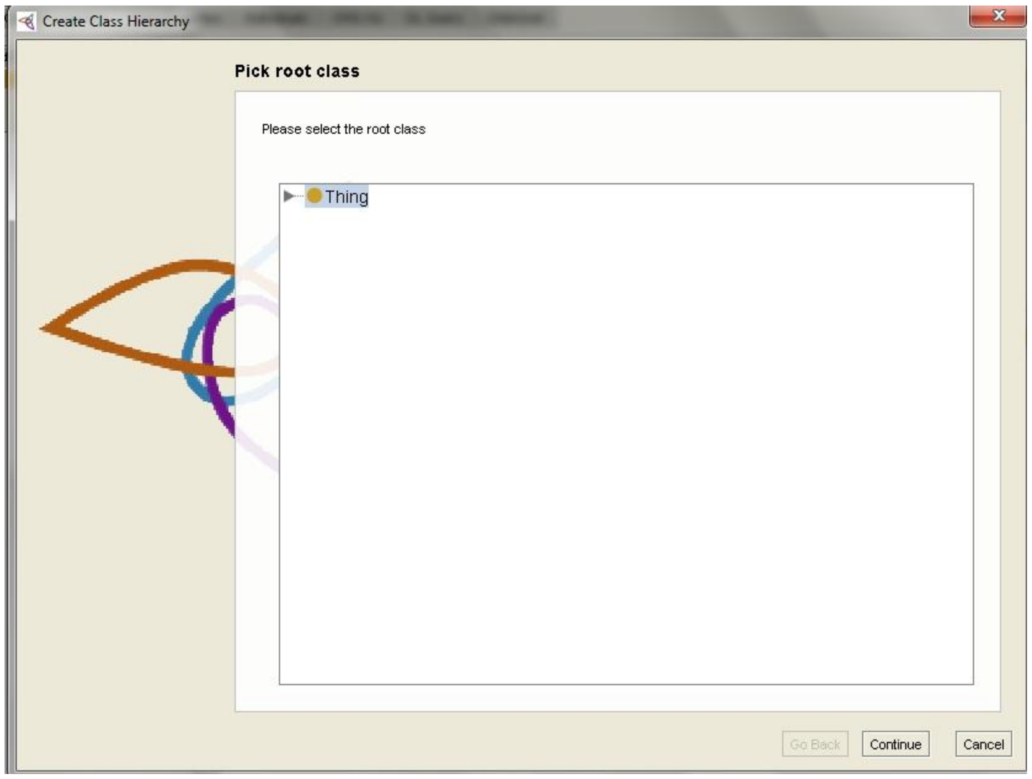
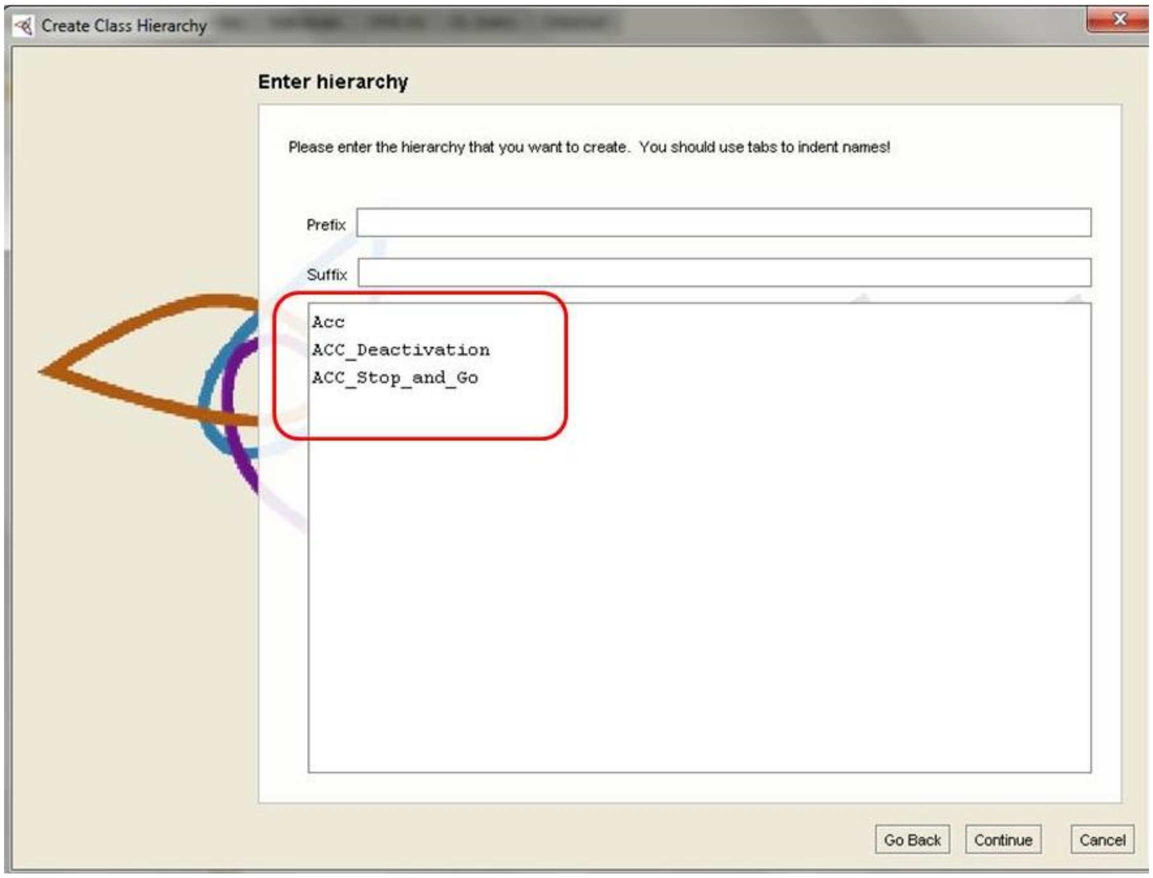


Figure (5.7)

4. Press the 'continue' button on the tool. Now we have to write the subclasses of 'Thing' that we want to create. In the large text area, type the class name 'ACC' and 'ACC_Deactivation' and also 'ACC_Stop_and_Go'. Now it should be like **(Figure 5.8)** If we had to create a lot of classes that had the same 'prefix' or 'suffix' we could use the options to automatically adding prefix and suffix to the class names that we entered.



Please enter the hierarchy that you want to create. You should use tabs to indent names!

Prefix

Suffix

```
Acc
ACC_Deactivation
ACC_Stop_and_Go
```

Figure (5.8)

5. Click the 'Continue' button on the tool. The names entered should be consistent with the naming style that has been mentioned before (No spaces etc.). Also all the created classes should be unique (no two classes with the same name), If there are any errors in the class names, they will be presented on this page, along with suggestions for corrections.

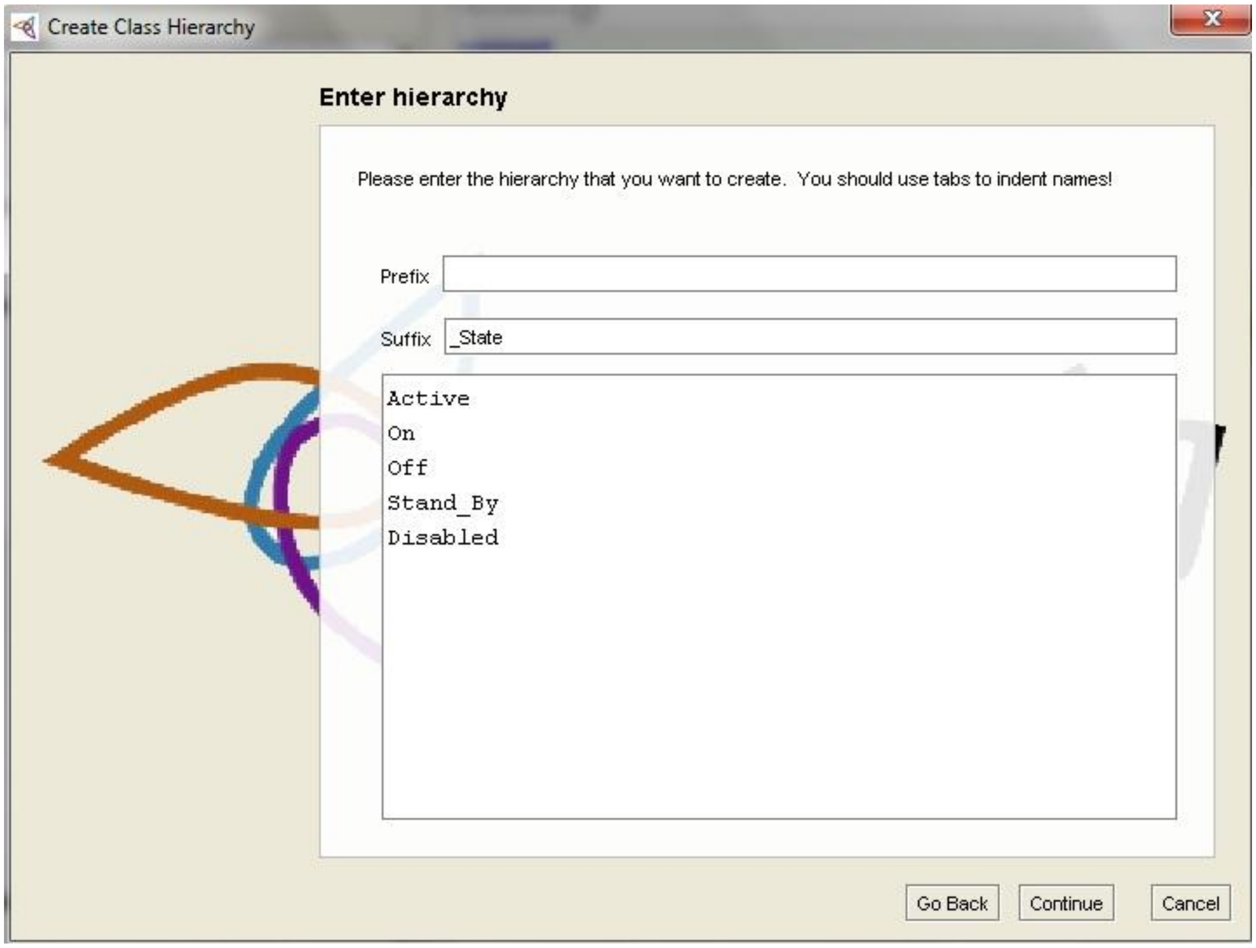
6. In order to make the new made classes 'Disjoint', ensure the tick box for 'Make all new classes disjoint' is ticked, this automatically makes new classes disjoint.

7. Then press 'Finish'. Now the tool creates the classes and makes them disjoint from each other.

The ontology should now have 'ACC', 'ACC_Sytem', ' ACC_Deactivation', 'ACC_Stop_and_Go', 'Time_Gap' as subclasses of 'Thing'. So 'Create Class Hierarchy' tool would simplify the process of adding classes, when we have a lot of classes to add to the ontology. After each modification make sure to press 'ctrl+s' and then click 'Ok' to save modifications that you have made.

For an instance you can create a new class and name it 'State'. ACC has different states such as: 'Active state', 'Standby state', 'Off state', 'On state' and 'Disabled state'. These states can be grouped as subclasses for class 'State'.

In order to add these classes, you have to follow the above mentioned steps. Notice that we want to add subclasses for 'State', so after you create class 'State', you have to select this class to add subclasses. You can use the suffix row, to add subclasses with the same suffix. **(Figure 5.9)** Shows how you can use suffix row.



Please enter the hierarchy that you want to create. You should use tabs to indent names!

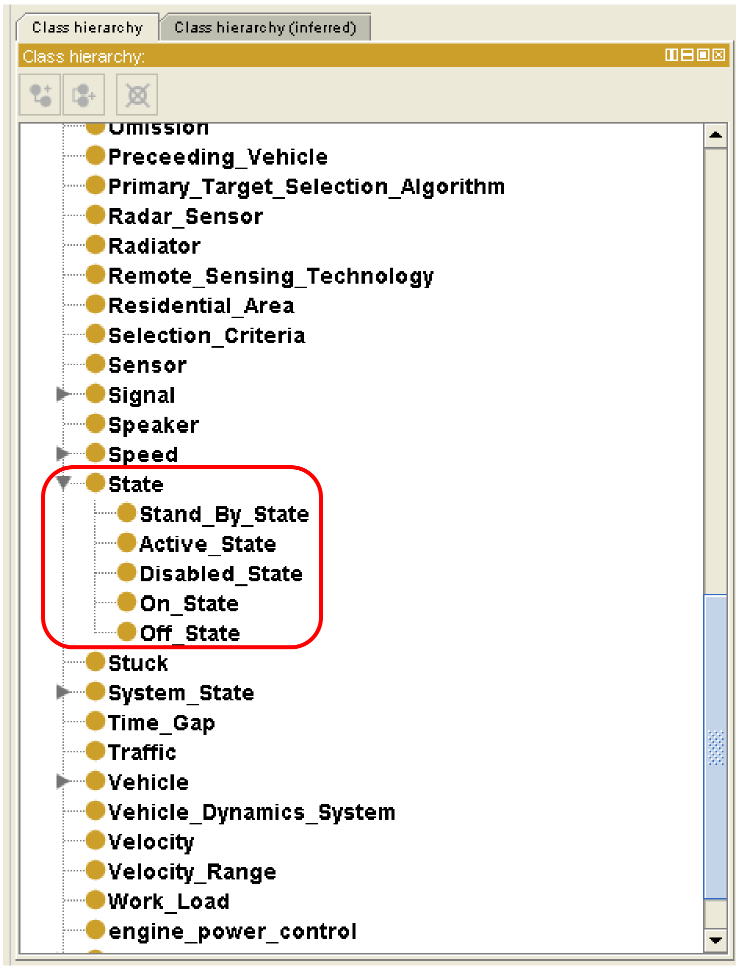
Prefix

Suffix _State

```
Active
On
Off
Stand_By
Disabled
```

Figure (5.9)

The class hierarchy should now look similar to **(Figure 5.10)**. The ordering of classes may be slightly different.



```
Omission
Preceeding_Vehicle
Primary_Target_Selection_Algorithm
Radar_Sensor
Radiator
Remote_Sensing_Technology
Residential_Area
Selection_Criteria
Sensor
Signal
Speaker
Speed
State
    Stand_By_State
    Active_State
    Disabled_State
    On_State
    Off_State
Stuck
System_State
Time_Gap
Traffic
Vehicle
Vehicle_Dynamics_System
Velocity
Velocity_Range
Work_Load
engine_power_control
```

Figure (5.10)

5.5. OWL Properties

There are three main types of properties, 'Object properties' and 'Data type properties' and 'Annotation properties'. In this tutorial we will focus on Object properties. Object properties are relationships between two individuals that link an individual to an individual.

(Figure 5.11) depicts an example of Object property. This figure shows that an object property linking the individual 'NTNU' to the individual 'Peter'.

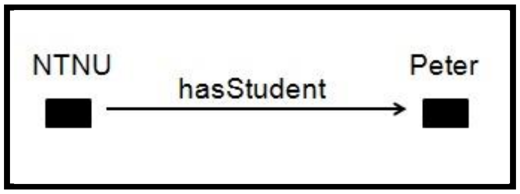


Figure (5.11)

Properties can be created using the ‘Object Properties’ tab shown in **(Figure 5.12)**.

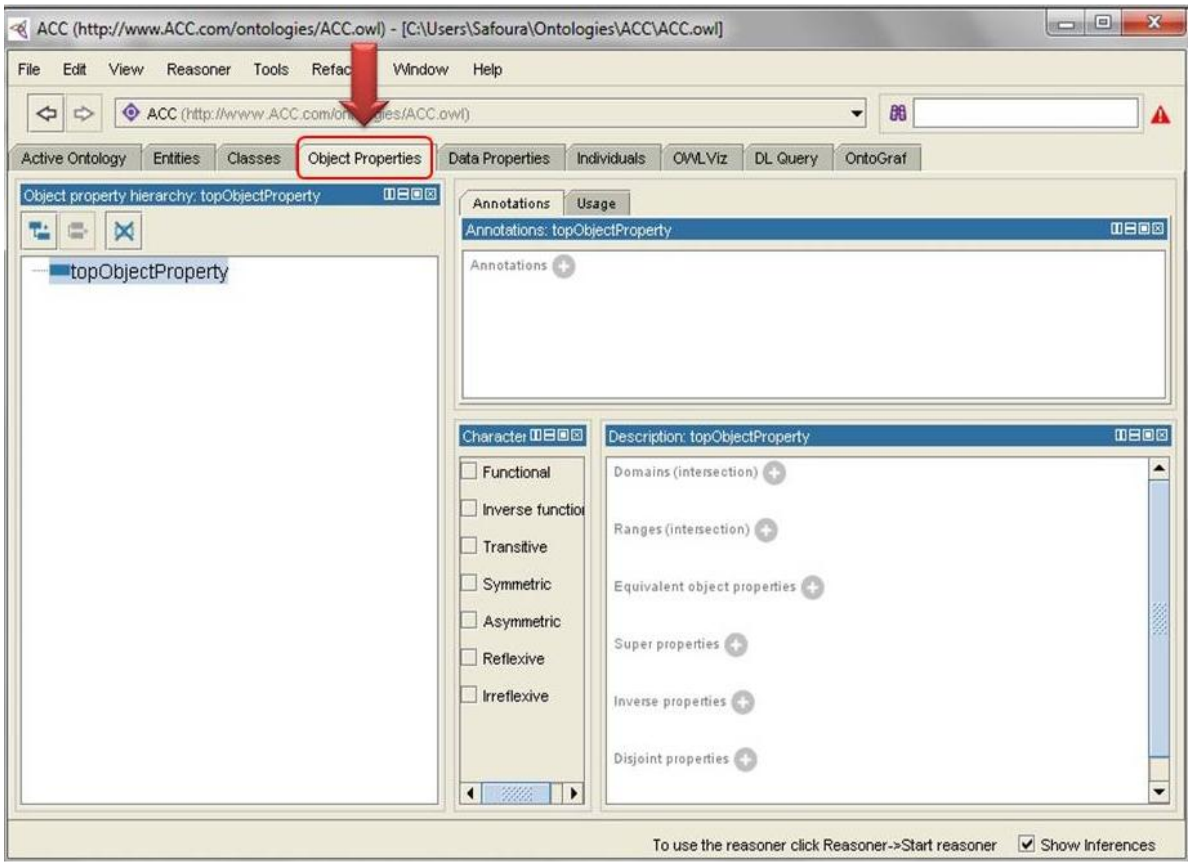


Figure (5.12)

(Figure 5.13) shows the buttons located in the top left hand corner of the ‘Object Properties’ tab that are used for creating OWL properties.

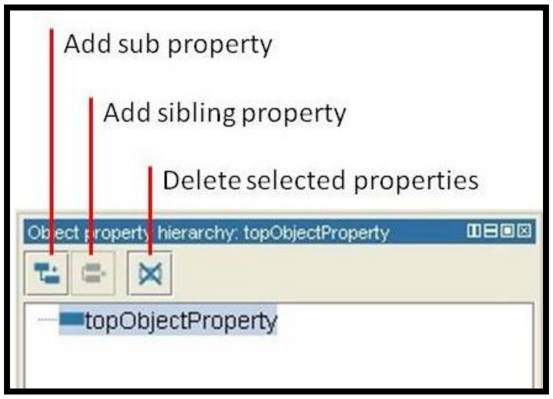

Add sub property
Add sibling property
Delete selected properties
Object property hierarchy: topObjectProperty
topObjectProperty

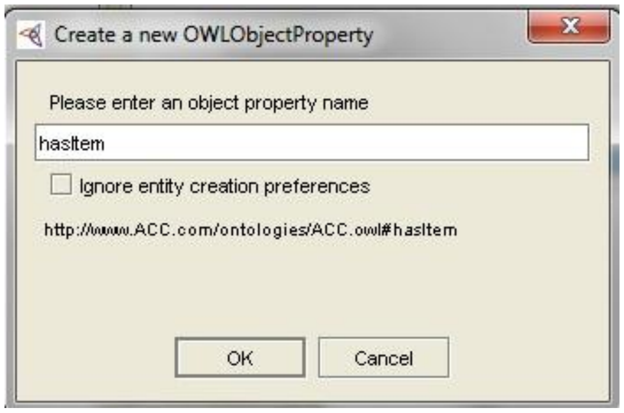
Figure (5.13)

Here we will create an object property called ‘hasItem’:

1. Switch to the ‘Object Properties’ tab.

2. After selecting ‘topObjectProperty’ in the object property tab, Use the ‘Add sub property’ button see **(Figure 5.13)** to create a new property.

3. In the ‘Create a new ‘OWLObjectProperty’ window which is opened, type the property name ‘hasItem’ and press ‘Ok’, as shown in **(Figure 5.14)**.


Create a new OWLObjectProperty

Please enter an object property name

hasItem

Ignore entity creation preferences

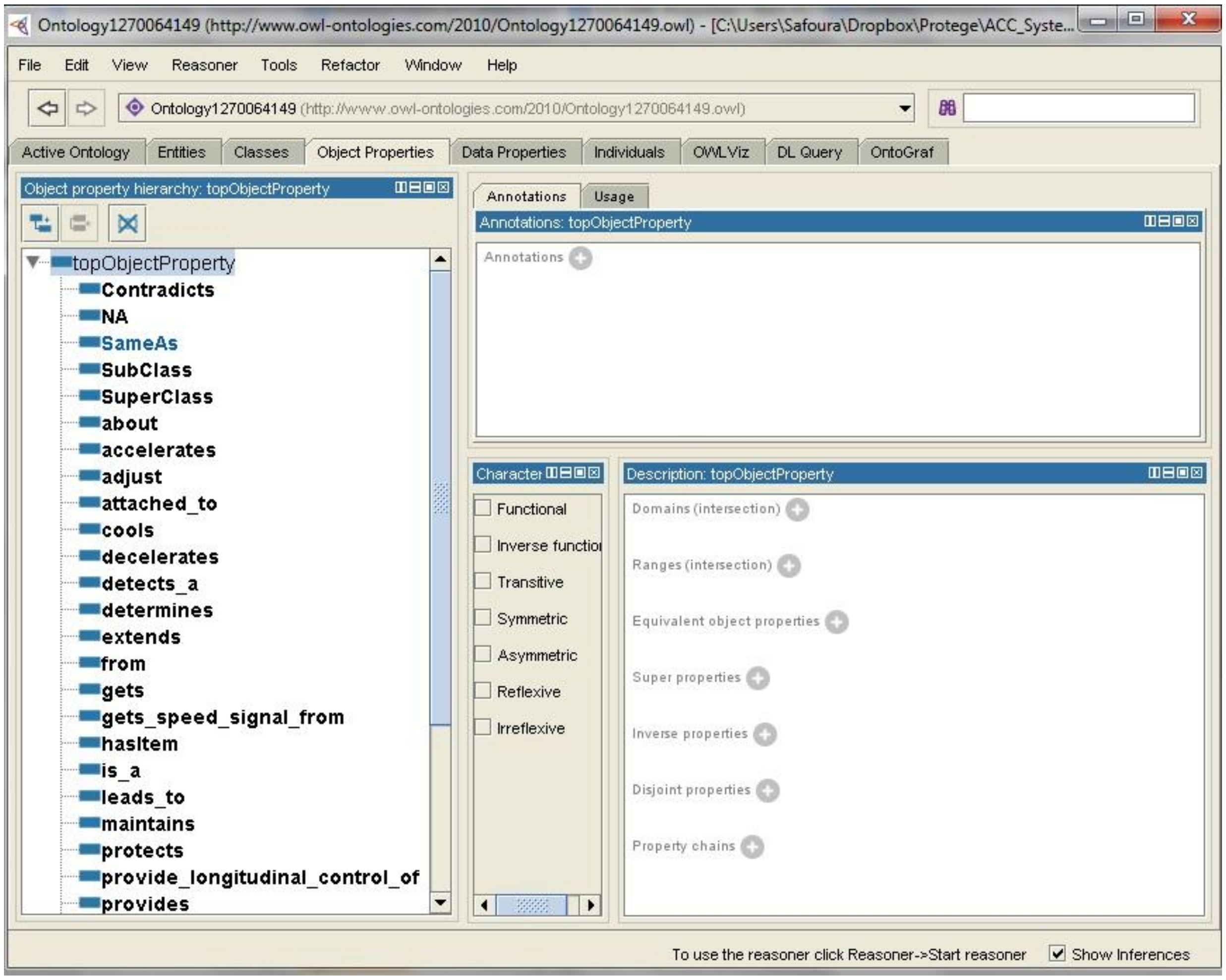
http://www.ACC.com/ontologies/ACC.owl#hasItem

Figure (5.14)

Now this property should be added to the list of the properties in the Object Properties tab.

For naming a property, although there is no strict naming rule for properties, we recommend you to start property names with a lowercase letter, have no spaces and have the remaining words capitalised. We also recommend that properties are starts with the word ‘has’, or the word ‘is’, for example ‘hasItem’, ‘isItemOf’.

Now we can add more properties to develop some relationships in the ACC domain. **(Figure 5.15)** shows some properties that have been added to the ACC ontology. For example, the property ‘SameAs’ is defined to relate two individuals that has the same concepts. For instance ‘ACC’ and ‘ACC_Syetem’.


Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\Safoura\Dropbox\Protege\ACC_Syste...

Object property hierarchy: topObjectProperty

- topObjectProperty
  - Contradicts
  - NA
  - SameAs
  - SubClass
  - SuperClass
  - about
  - accelerates
  - adjust
  - attached_to
  - cools
  - decelerates
  - detects_a
  - determines
  - extends
  - from
  - gets
  - gets_speed_signal_from
  - hasItem
  - is_a
  - leads_to
  - maintains
  - protects
  - provide_longitudinal_control_of
  - provides

Figure (5.15)

5.6. Inverse Properties

For each object property we can have a corresponding inverse property. If some property links individual ‘A’ to individual ‘B’ then its inverse property will link individual ‘B’ to individual ‘A’. **(Figure 5.16)** shows the property ‘hasStudent’ and its inverse property ‘studyIn’. If NTNU hasStudent Peter, then because of the inverse property, we can conclude that Peter studyIn NTNU.

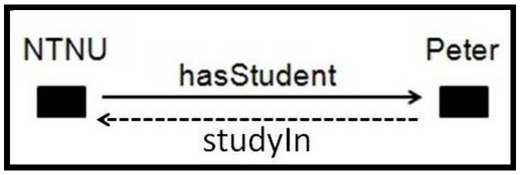


Figure (5.16)

To make it more clear, we will create an inverse property for the property 'hasItem' that we created previously by using the 'Inverse properties' button shown in **(Figure 5.17)**.

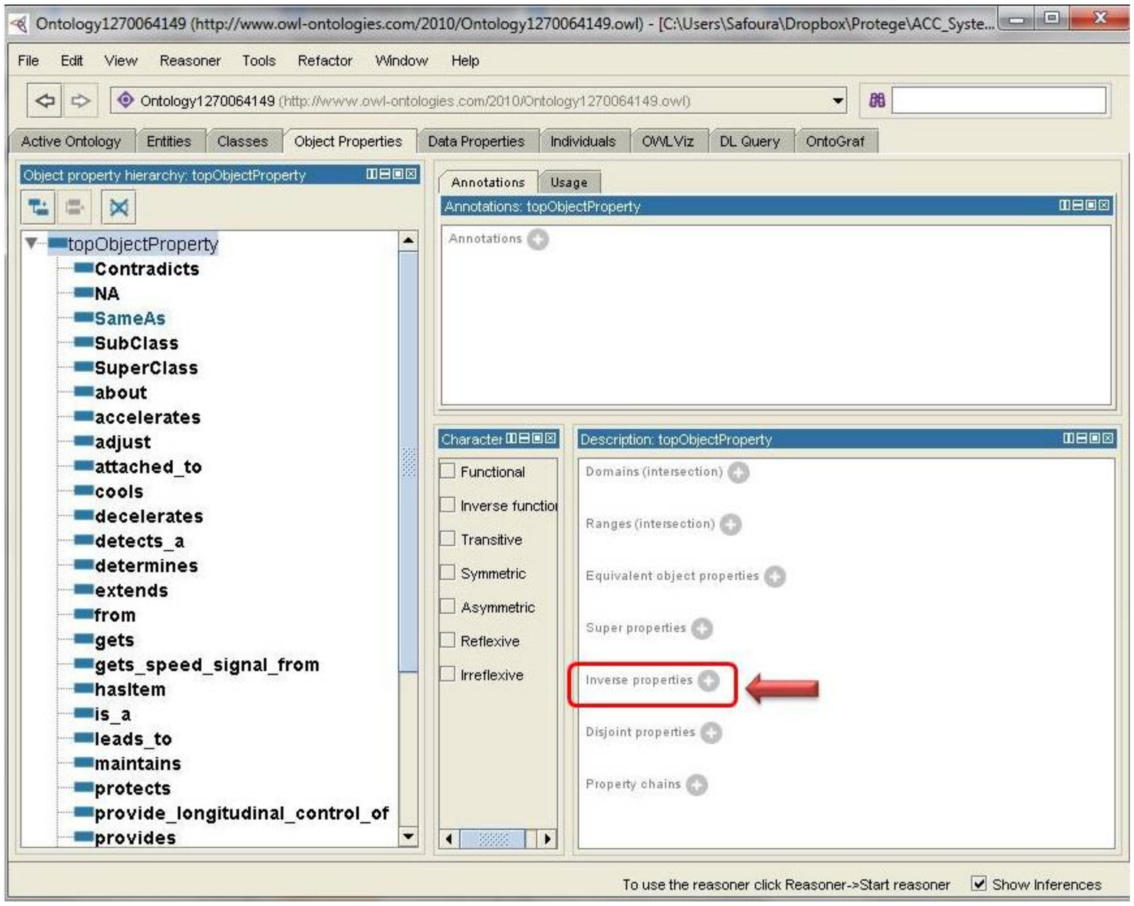


Figure (5.17)

1. The same as what we did in the section 5.5, use the 'Add sub property' button on the 'Object Properties' tab to create a new Object property called 'isItemOf', this will become the inverse property of 'hasItem'.
2. Press the 'Add' button which is next to 'Inverse properties' on the property 'Description' view **(Figure 5.17)**. A window will become open and you can select a property from the property list. Select the 'hasItem' property and press `OK' **(Figure 5.18)**.

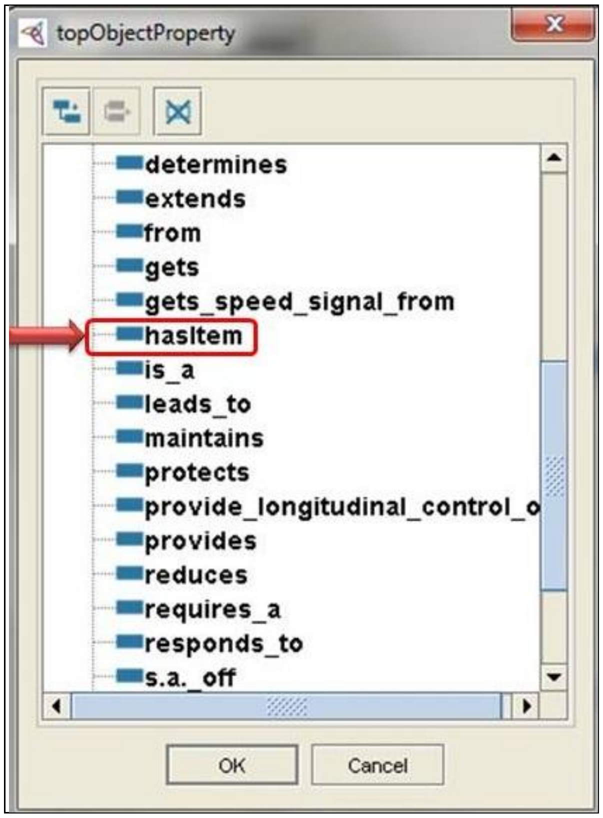


Figure (5.18)

The property 'hasItem' should now be displayed as inverse property for 'isItemOf' property in the property 'Description' view **(Figure 5.19).**

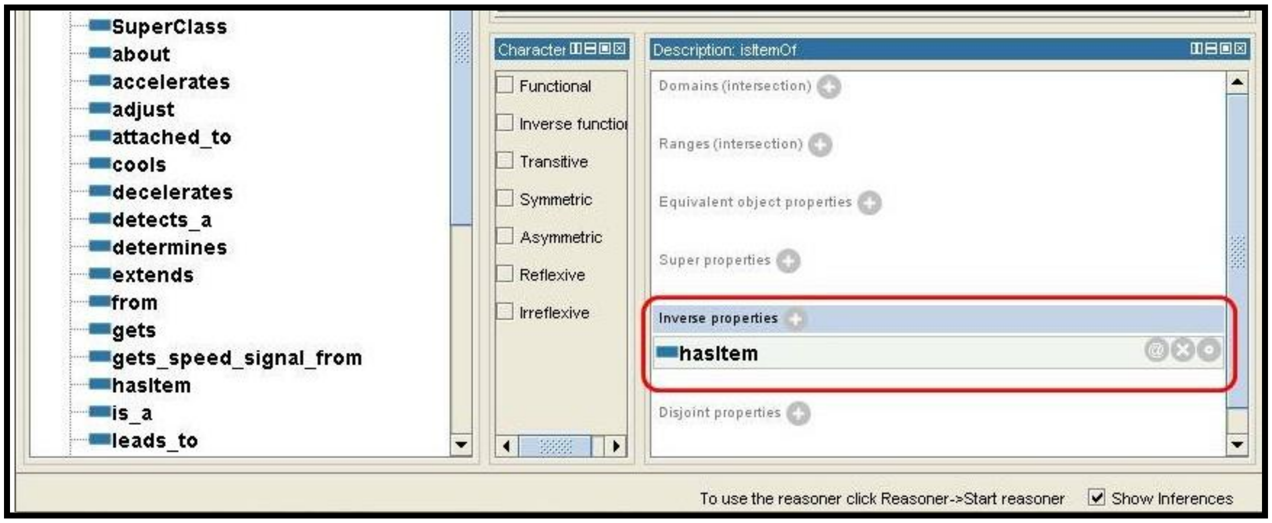


Figure (5.19)

5.7. Object Property Characteristics

In OWL, object properties can have different characteristics. In this section we will discuss some of these characteristics.

5.7.1. Functional Properties:

When for an individual, there can be at most one individual that is related to the individual via the property, so we define this property as 'Functional'. **(Figure 5.20)** shows an example of a functional property.

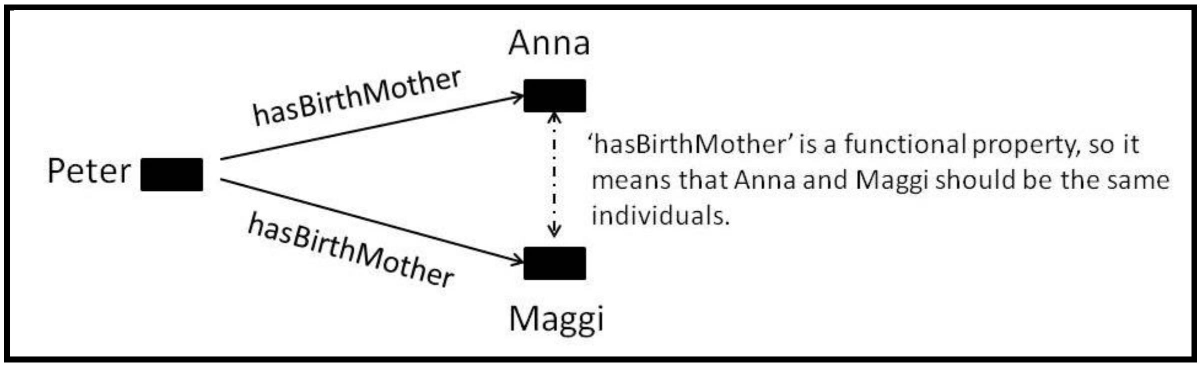

Anna
hasBirthMother
Peter
hasBirthMother
'hasBirthMother' is a functional property, so it means that Anna and Maggi should be the same individuals.
Maggi

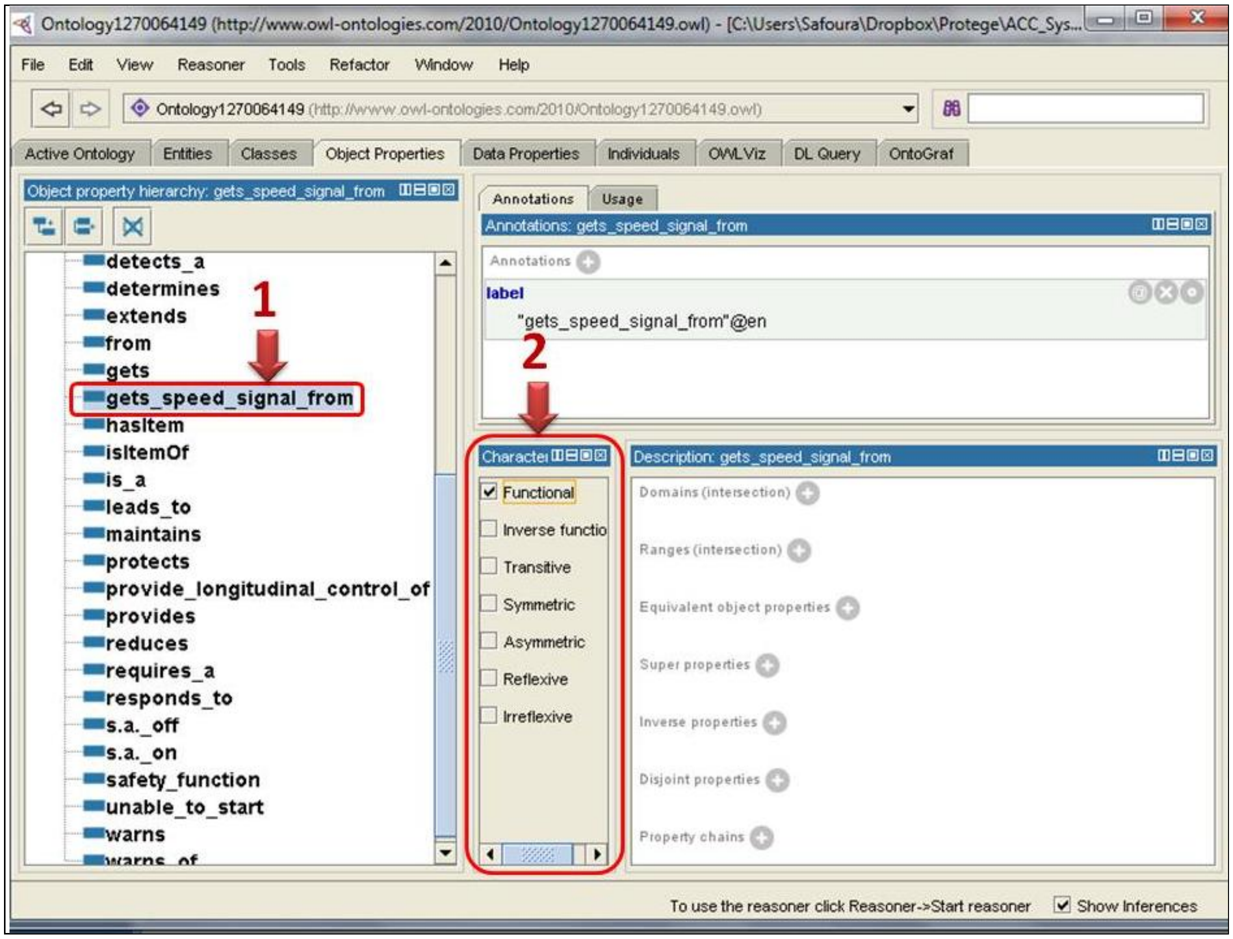
Figure (5.20)

If we say that an individual Peter hasBirthMother Anna, and also Peter hasBirthMother Maggi then because hasBirthMother is a 'Functional' property, and for each person there can exist just one birth mother, so we can conclude that Anna and Maggi must be the same individuals otherwise we would face inconsistency.

To make a property 'Functional':

1. In the 'Object Properties' tab, first we have to select the property, for example 'gets_speed_signal_from' property, as shown in **(Figure 5.21).**

2. Then select 'Functional', using the 'Characteristics' view. **(Figure 5.21)** shows that because ACC_Sytem can get speed signal only from one individual of 'Driveshaft', so 'gets_speed_signal_from' property is a 'Functional' property.


Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\Safoura\Dropbox\Protege\ACC_Sys...
File Edit View Reasoner Tools Refactor Window Help
Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Object property hierarchy: gets_speed_signal_from
detects_a
determines
extends
from
gets
gets_speed_signal_from
hasItem
isItemOf
is_a
leads_to
maintains
protects
provide_longitudinal_control_of
provides
reduces
requires_a
responds_to
s.a._off
s.a._on
safety_function
unable_to_start
warns
warns_of
1
2
Annotations Usage
Annotations: gets_speed_signal_from
Annotations
label
"gets_speed_signal_from"@en
Characte
Functional
Inverse functio
Transitive
Symmetric
Asymmetric
Reflexive
Irreflexive
Description: gets_speed_signal_from
Domains (intersection)
Ranges (intersection)
Equivalent object properties
Super properties
Inverse properties
Disjoint properties
Property chains
To use the reasoner click Reasoner->Start reasoner
Show Inferences

Characteristics: [✓ Functional] Inverse functional, Transitive, Symmetric, Asymmetric, Reflexive, Irreflexive

Figure (5.21)

5.7.2. Inverse Functional Properties:

When a property is 'Inverse functional' then it means that the inverse property is functional. For better explanation we can consider the example we had for functional property, 'hasBirthMother'. The inverse property for this property is 'isBirthMotherOf' and since 'hasBirthMother' is functional, 'isBirthMotherOf' is inverse functional. If we say that Anna is the birth mother of Peter, and we also say that Maggi is the birth mother of Peter, so we can conclude that Anna and Maggi are both the same individuals.

5.7.3. Symmetric Properties:

When individual A relates to individual B with property P, and also individual B relates to individual A with property P, therefore property P is a 'Symmetric' property.

(Figure 5.22) shows an example of a symmetric property. If the individual Peter is related to the individual Bob via the hasSibling property, then we can infer that Bob must also be related to Peter via the hasSibling property. Because when Peter is a sibling of Bob, so Bob is also a sibling of Peter. Therefore hasSibling is a 'Symmetric' property.

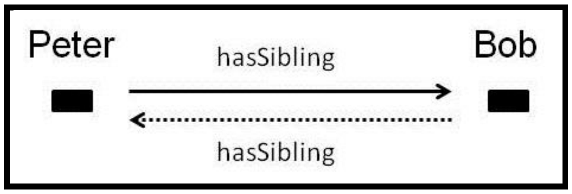


Figure (5.22)

In the ACC system ontology, the property 'Contradicts' can be considered as a symmetric property , because the Brake contradicts Clutch and also Clutch contradicts Brake, therefore the contradict property is a 'Symmetric' property. 'SameAs' property is also a 'Symmetric' property.

To make a property 'Symmetric', we should first select the property and then use the 'Characteristics' view, the same as what we did for making a property functional but here we select symmetric instead.

5.7.4 Asymmetric properties:

If a property P is 'Asymmetric', then the relation we defined in the above cannot exist. For example for the property 'isChildOf', when Peter isChildOf Anna, it is not possible to say that

Anna is also child of Peter, and this relation is not exist. It is not correct to say that Peter isChildOf Anna, and also Anna isChildOf Peter because 'isChildOf' property is not a 'Symmetric' property and it is an 'Asymmetric' property. **(Figure 5.23)** shows an example of an 'Asymmetric' property.

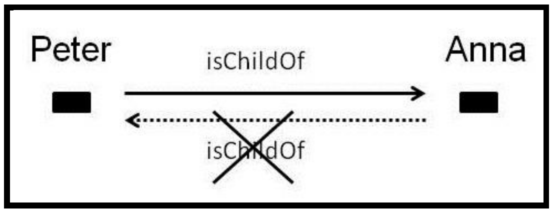
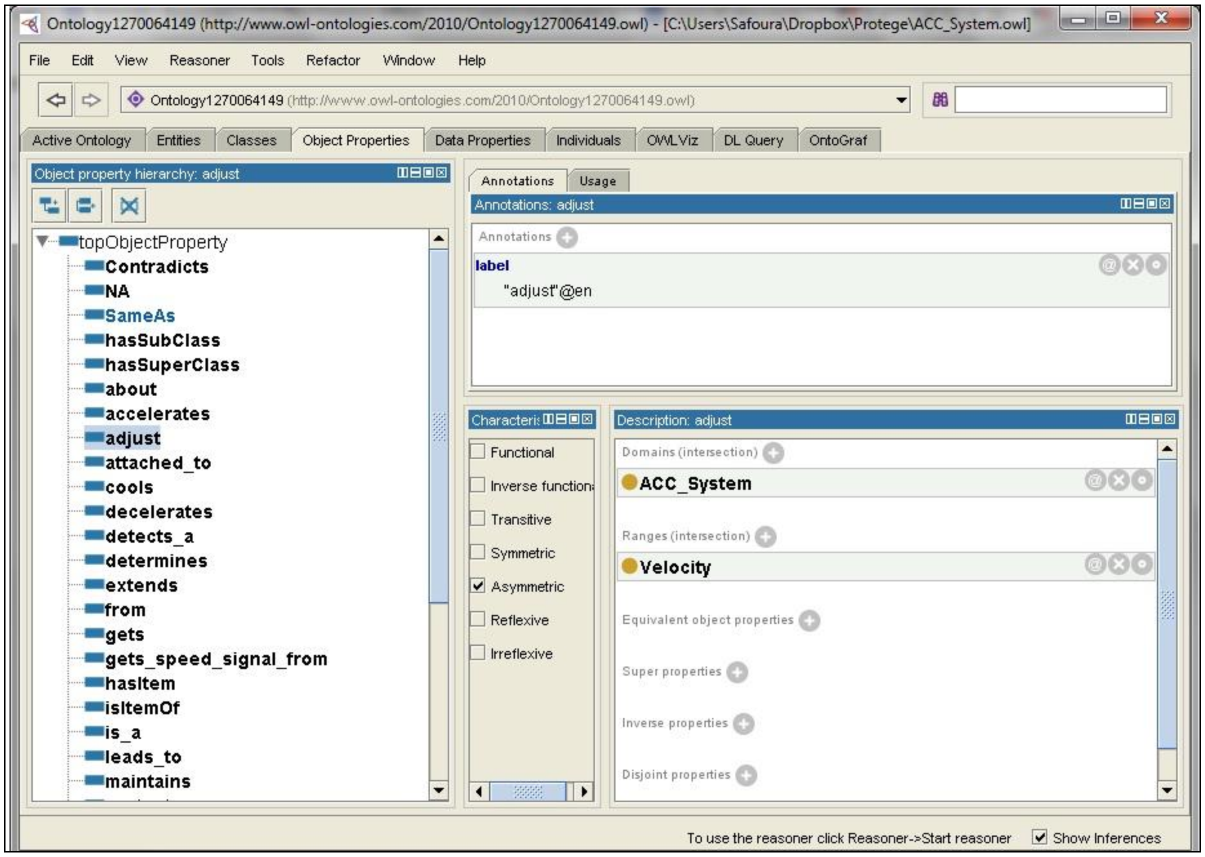


Figure (5.23)

In the ACC system ontology, the property 'adjust' is an 'Asymmetric' property. Because we can say that ACC_System adjust Velocity, but the relation Velocity adjust ACC_Sytem is not exist. **(Figure 5.24)** shows that 'Asymmetric' characteristic is selected for 'adjust' property.


Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\Safoura\Dropbox\Protege\ACC_System.owl]

Object property hierarchy: adjust

topObjectProperty
- Contradicts
- NA
- SameAs
- hasSubClass
- hasSuperClass
- about
- accelerates
- adjust
- attached_to
- cools
- decelerates
- detects_a
- determines
- extends
- from
- gets
- gets_speed_signal_from
- hasItem
- isItemOf
- is_a
- leads_to
- maintains

Annotations: adjust

label "adjust"@en

Characteristics:
- [ ] Functional
- [ ] Inverse functional
- [ ] Transitive
- [ ] Symmetric
- [x] Asymmetric
- [ ] Reflexive
- [ ] Irreflexive

Description: adjust

Domains (intersection): ACC_System

Ranges (intersection): Velocity

Equivalent object properties

Super properties

Inverse properties

Disjoint properties

To use the reasoner click Reasoner->Start reasoner ☑ Show Inferences

Figure (5.24)

5.8. Property Domain and range

In OWL properties may have domain and range. In other words properties link individuals from domain to range. For example in University ontology the studyIn property links individuals from Student's class to individuals in University class. In this example Student is domain and University is range. In situations that two property are inverse, like studyIn and hasStudent in University ontology, domain of studyIn property is range of hasStudent and vice-versa. It is shown in (**Figure 5.25).**

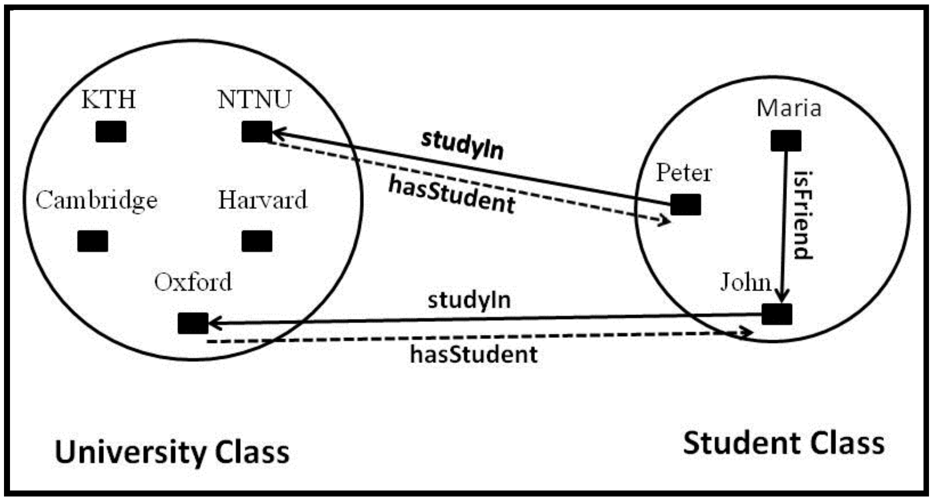


Figure (5.25)

In order to specify domain and range for a property please follow the steps:

Specifying the range for a property:

1. Select Object property tab in Protégé at first and then select 'adjust' property in property hierarchy

2. In the property description view window press the 'Add' icon next to the 'Range'.

3. In the opened dialog you can choose a class from ontology class hierarchy. Select Velocity and press 'ok' button. So Velocity should now be displayed in the range list. All these three steps are shown in **(Figure 5.26).**

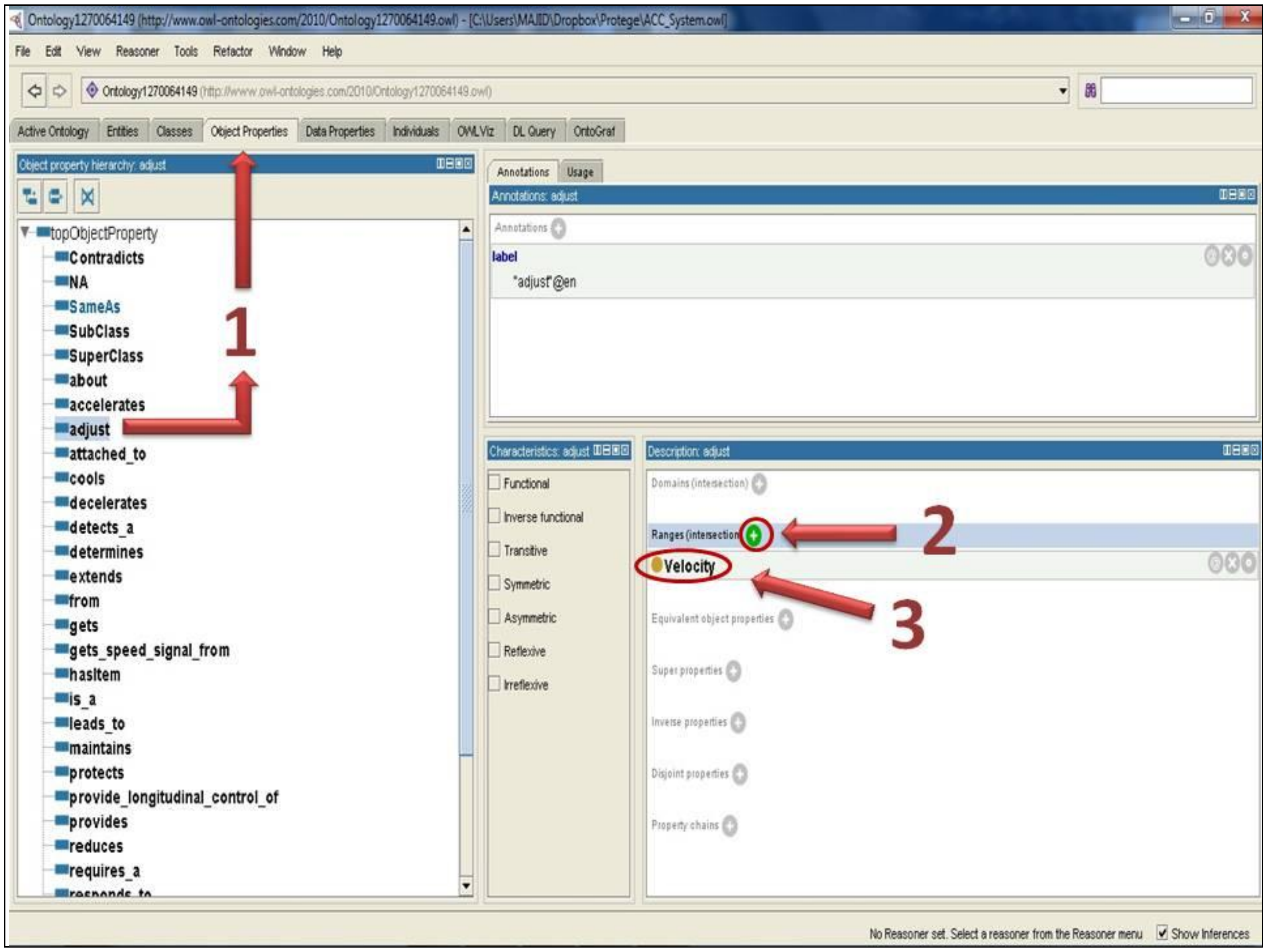


Figure (5.26)

It is possible to specify multiple classes as the range for a property. In Protégé multiple classes as the range is interpreted to be the intersection of the classes.

Specifying domain for a property:

1. Select Object property tab in Protégé at first and then select 'adjust' property in property hierarchy.

2. In the property description view window press the ‘Add’ icon next to the 'Domain'.

3. In the opened dialog you can choose a class from ontology class hierarchy. Select 'ACC_System ' and press 'ok' button. So 'ACC_System ' should now be displayed in the 'Domain' list. All these three steps are shown in **(Figure 5.27).**

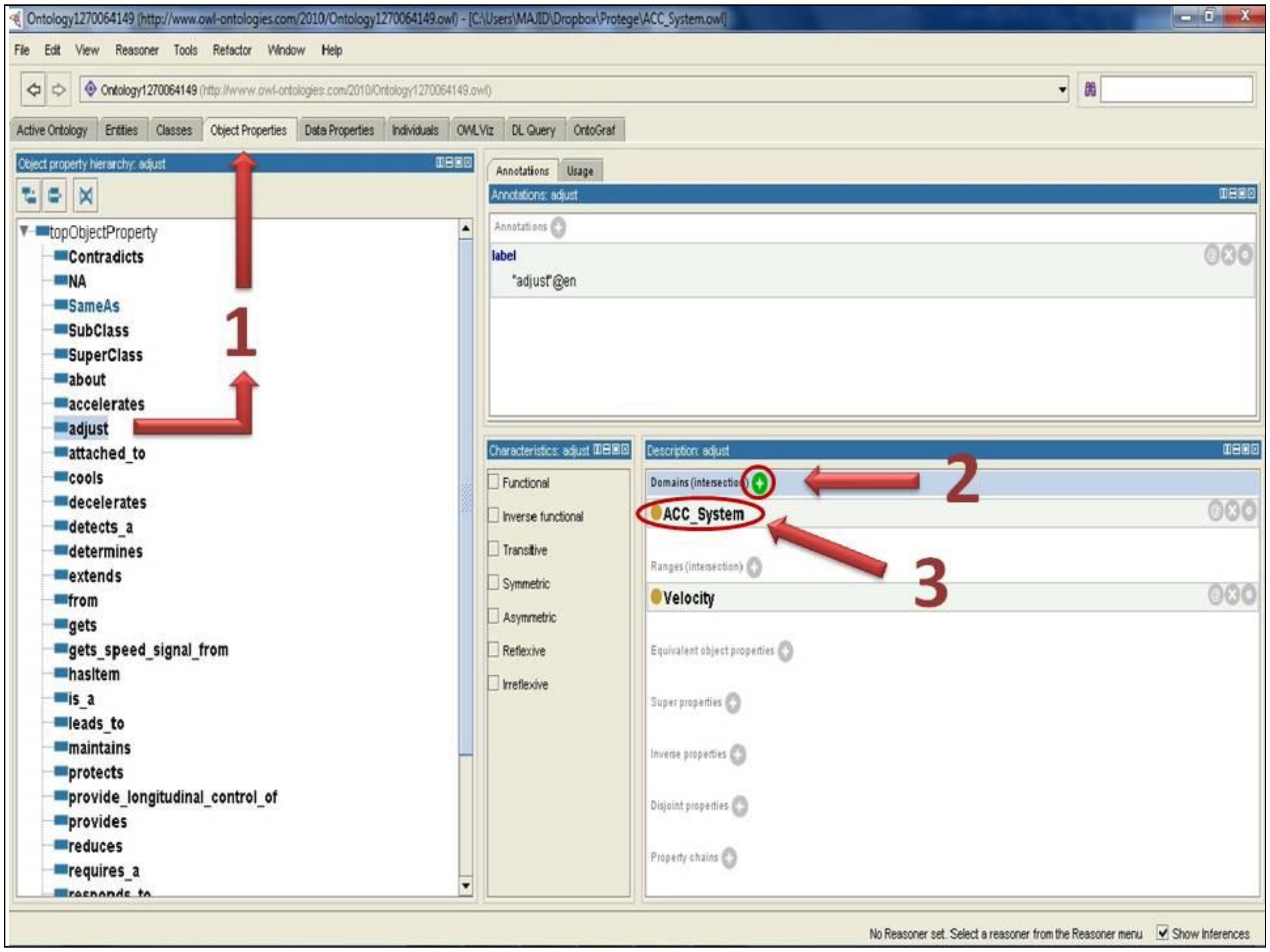


Figure (5.27)

5.9. Property restriction

A restriction describes a class of individuals based on the relationships that members of the class participate in. In other words a restriction is a kind of class, in the same way that a named class is a kind of class. As it represented before properties are binary relationship between individuals that object properties describe relationship between individuals and data properties describe relationship between individuals and data values. In OWL all restrictions fall into three main categories:

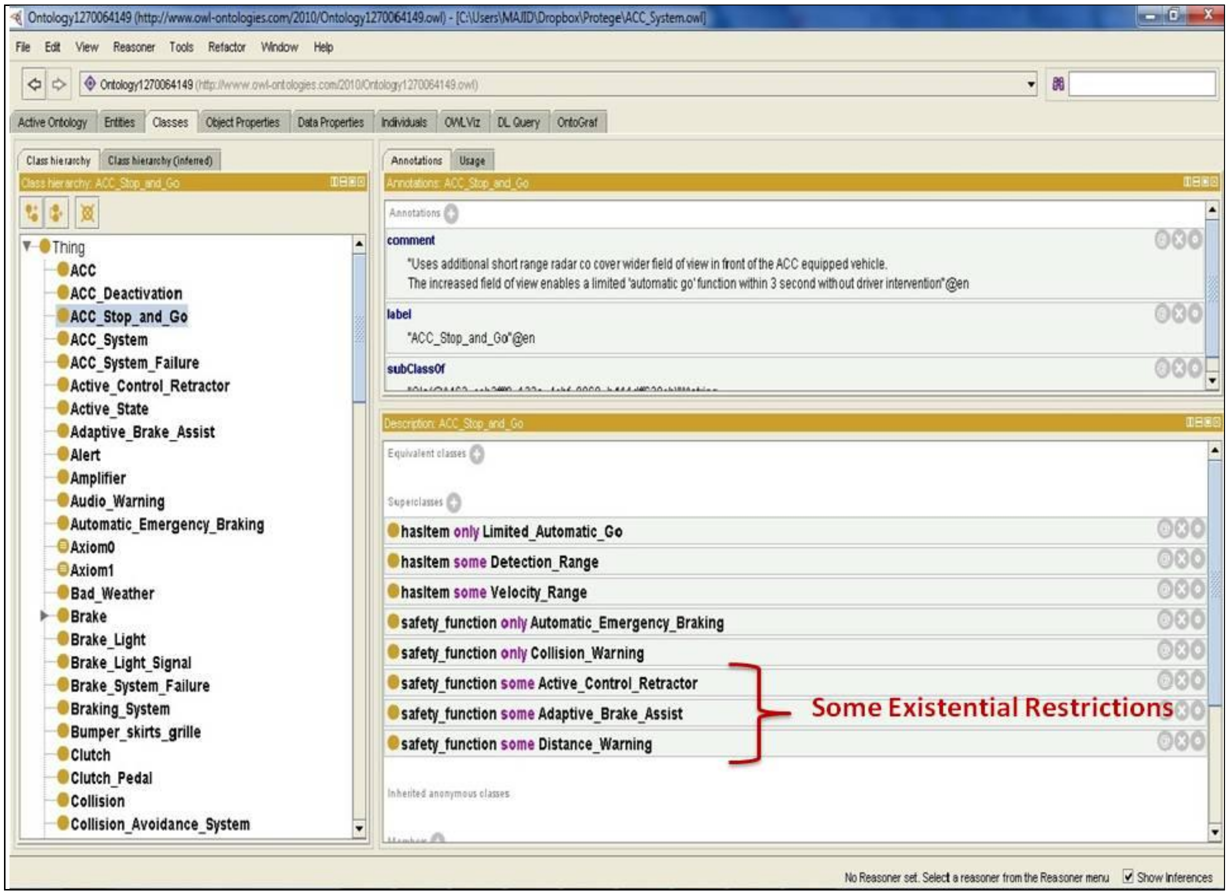
- Quantifier Restrictions
- Cardinality Restrictions
- hasValue Restrictions

Quantifier restriction is categorized into **existential** restriction and **universal** restriction.

5.9.1. Existential Restriction:

Existential restrictions describe classes of individuals that participate in at least one relationship along a specified property to individuals that are members of a specified class. For example, 'safety_function some Active_Control_Recractor' describes all of the class of individuals that have at least one (some) 'safety_function' relationship to members of Active_Control_Recractor'. In Protégé the keyword 'some' is used to denote existential restrictions [1]. In **(Figure 5.28)** some existential restrictions are represented. Existential restrictions may be denoted by the existential quantifier (∃). They are also known as `someValuesFrom' restrictions in OWL speak.

1- Existential restrictions are also known as Some Restrictions, or as some values from restrictions.


Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\MAJID\Dropbox\Protege\ACC_System.owl]
File Edit View Reasoner Tools Refactor Window Help
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Class hierarchy Class hierarchy (inferred)
Annotations Usage
Some Existential Restrictions
No Reasoner set. Select a reasoner from the Reasoner menu
Show Inferences

Class hierarchy: ACC_Stop_and_Go

- Thing
  - ACC
  - ACC_Deactivation
  - ACC_Stop_and_Go
  - ACC_System
  - ACC_System_Failure
  - Active_Control_Retractor
  - Active_State
  - Adaptive_Brake_Assist
  - Alert
  - Amplifier
  - Audio_Warning
  - Automatic_Emergency_Braking
  - Axiom0
  - Axiom1
  - Bad_Weather
  - Brake
  - Brake_Light
  - Brake_Light_Signal
  - Brake_System_Failure
  - Braking_System
  - Bumper_skirts_grille
  - Clutch
  - Clutch_Pedal
  - Collision
  - Collision_Avoidance_System

Annotations: ACC_Stop_and_Go

comment
"Uses additional short range radar co cover wider field of view in front of the ACC equipped vehicle.
The increased field of view enables a limited 'automatic go' function within 3 second without driver intervention"@en

label
"ACC_Stop_and_Go"@en

subClassOf

Description: ACC_Stop_and_Go

Equivalent classes

Superclasses
- hasItem only Limited_Automatic_Go
- hasItem some Detection_Range
- hasItem some Velocity_Range
- safety_function only Automatic_Emergency_Braking
- safety_function only Collision_Warning
- safety_function some Active_Control_Retractor
- safety_function some Adaptive_Brake_Assist
- safety_function some Distance_Warning

Inherited anonymous classes

Figure (5.28)

In order to add existential restrictions please follow these steps:

1. Select 'ACC_stop_and_go' from the class hierarchy on the 'Classes' tab. **(Figure 5.29).**

2. Select the 'Add' icon next to 'Superclasses' header in the 'Class Description View' (**Figure 5.29)**. This will open a text box where we can enter our restrictions **(Figure 5.30).**

The create restriction text box allows you construct restrictions using class, property and individual names. You can drag and drop classes, properties and individuals into the text box or type them in, the text box with check all the values you enter and alert you to any errors. To create a restriction we have to do three things:

- Enter the property to be restricted from the property list.
- Enter a type of restriction from the restriction types e.g. `some' for an existential restriction.
- Specify a filler for the restriction

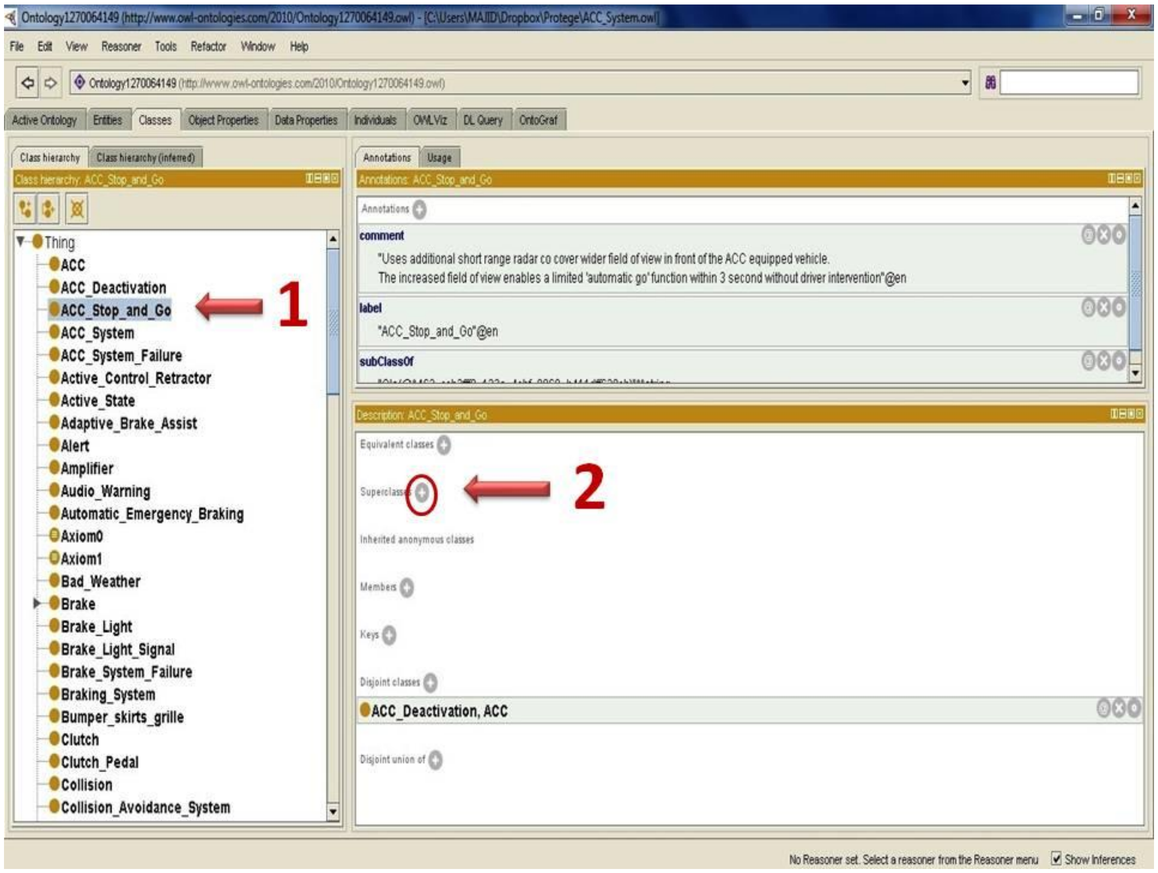
3. Select 'Class expression editor' tab at first **(Figure 5.30).**

4. You can either drag or drop ‘safty_function’ from the property list into the create restriction text box, or type it in **(Figure 5.30).**

5. Now add the type or restriction, we will use an existential restriction so type 'some' **(Figure 5.30).**

6. Specify that the filler is 'Active_Control_Recractor'. to do this either: type 'Active_Control_Recractor' into the filler edit box, or drag and drop'Active_Control_Recractor' into the text box **(Figure 5.30).**

7. Press 'Ok' button to create the restriction **(Figure 5.30)**. If all information was entered correctly the dialog will close and the restriction will be displayed in the 'Class Description View' **(Figure 5.31).** If there were errors they will be underlined in red in the text box, or popup will give some hints to the cause of the error. If this is the case, recheck that the type of restriction, the property and filler have been specified correctly.


Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\MAJID\Dropbox\Protege\ACC_System.owl]
File Edit View Reasoner Tools Refactor Window Help
Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Class hierarchy Class hierarchy (inferred)
Class hierarchy: ACC_Stop_and_Go
Thing
ACC
ACC_Deactivation
ACC_Stop_and_Go
1
ACC_System
ACC_System_Failure
Active_Control_Retractor
Active_State
Adaptive_Brake_Assist
Alert
Amplifier
Audio_Warning
Automatic_Emergency_Braking
Axiom0
Axiom1
Bad_Weather
Brake
Brake_Light
Brake_Light_Signal
Brake_System_Failure
Braking_System
Bumper_skirts_grille
Clutch
Clutch_Pedal
Collision
Collision_Avoidance_System
Annotations Usage
Annotations: ACC_Stop_and_Go
Annotations
Description: ACC_Stop_and_Go
Equivalent classes
Superclasses
2
Inherited anonymous classes
Members
Keys
Disjoint classes
Disjoint union of
No Reasoner set. Select a reasoner from the Reasoner menu
Show Inferences

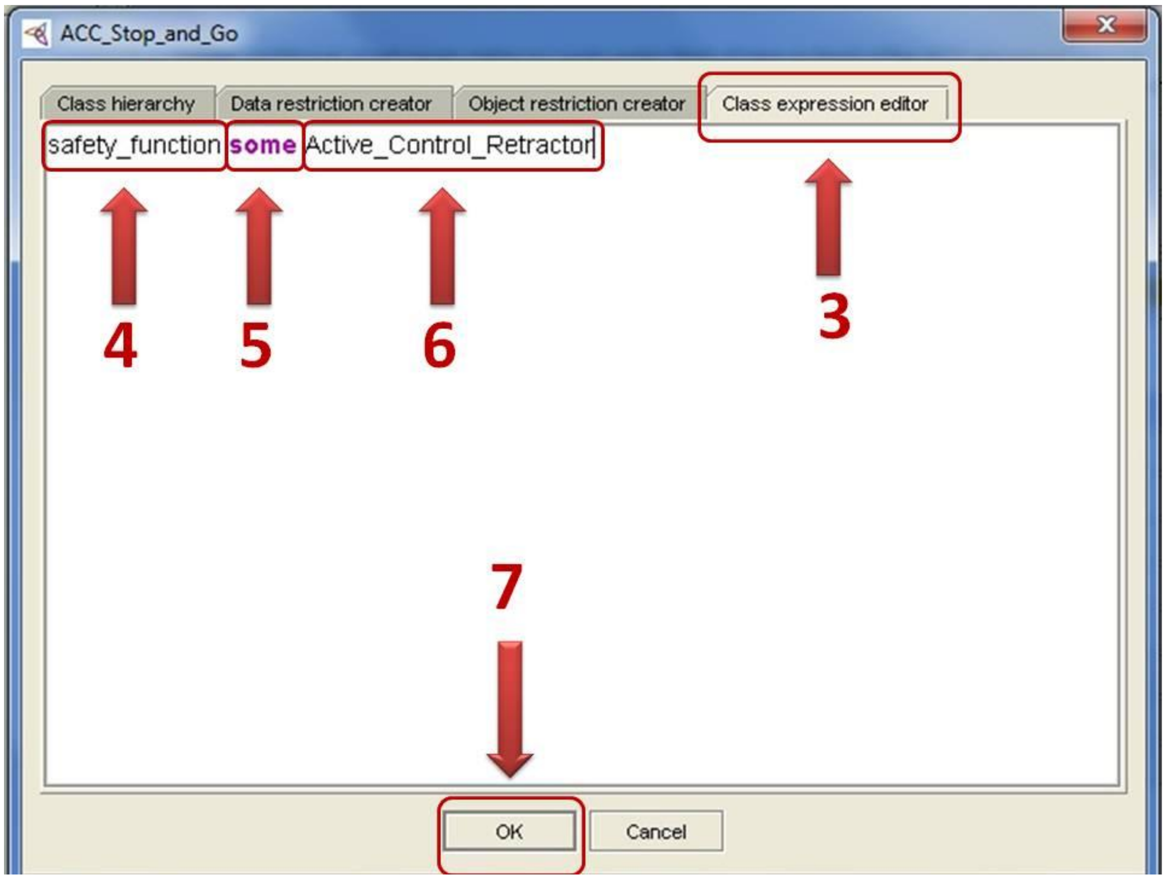
comment
"Uses additional short range radar co cover wider field of view in front of the ACC equipped vehicle.
The increased field of view enables a limited 'automatic go' function within 3 second without driver intervention"@en

label
"ACC_Stop_and_Go"@en

subClassOf

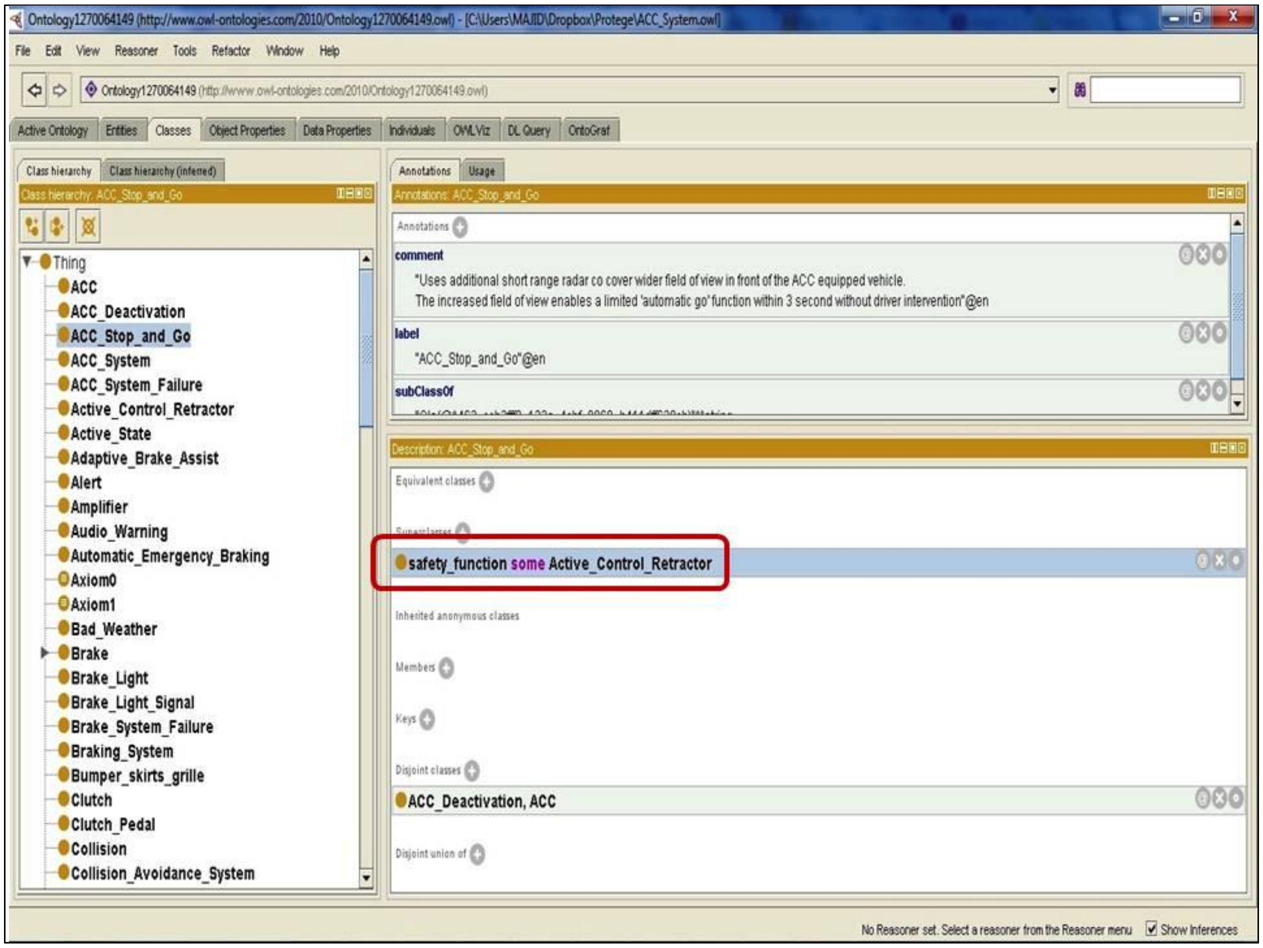
Disjoint classes: ACC_Deactivation, ACC

(Figure 5.29)



safety_function some Active_Control_Retractor

(Figure 5.30)


Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\MAJID\Dropbox\Protege\ACC_System.owl]
File Edit View Reasoner Tools Refactor Window Help
Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Class hierarchy Class hierarchy (inferred)
Class hierarchy: ACC_Stop_and_Go
Thing
ACC
ACC_Deactivation
ACC_Stop_and_Go
ACC_System
ACC_System_Failure
Active_Control_Retractor
Active_State
Adaptive_Brake_Assist
Alert
Amplifier
Audio_Warning
Automatic_Emergency_Braking
Axiom0
Axiom1
Bad_Weather
Brake
Brake_Light
Brake_Light_Signal
Brake_System_Failure
Braking_System
Bumper_skirts_grille
Clutch
Clutch_Pedal
Collision
Collision_Avoidance_System
Annotations Usage
Annotations: ACC_Stop_and_Go
Annotations
Description: ACC_Stop_and_Go
Equivalent classes
Superclasses
Inherited anonymous classes
Members
Keys
Disjoint classes
Disjoint union of
No Reasoner set. Select a reasoner from the Reasoner menu
Show Inferences

comment
"Uses additional short range radar co cover wider field of view in front of the ACC equipped vehicle.
The increased field of view enables a limited 'automatic go' function within 3 second without driver intervention"@en

label
"ACC_Stop_and_Go"@en

subClassOf

safety_function some Active_Control_Retractor

ACC_Deactivation, ACC

(Figure 5.31)

5.9.2. Universal restriction:

Universal restrictions describe classes of individuals that for a given property only have relationships along this property to individuals that are members of a specified class. For example, 'the class of individuals that only have 'safty_function' relationships to members of 'Automatic_Emergency_Breaking'. In Protégé the keyword 'only' is used to denote universal restrictions.

In compare to existential restriction, we could use an existential restriction 'safety_function some Active_Control_Recractor' to describe the individuals that have at least one relationship along the property safety_function to an individual that is a member of the class ACC_stopp_and_go. This restriction does not imply that all of the safety_function relationships must be to a member of the class Active_Control_Recractor. To restrict the relationships for a given property to individuals that are members of a specific class we must use a universal restriction. Universal restrictions are given the symbol(∀) . Universal restrictions are also known as AllValuesFrom Restrictions.

In order to add an existential restriction please follow these steps:

1. Making sure that ACC_stop_and_go is selected, click on the `Add' icon next to the 'Superclasses' header in the 'Class Description View'. **(Figure 5.32).**

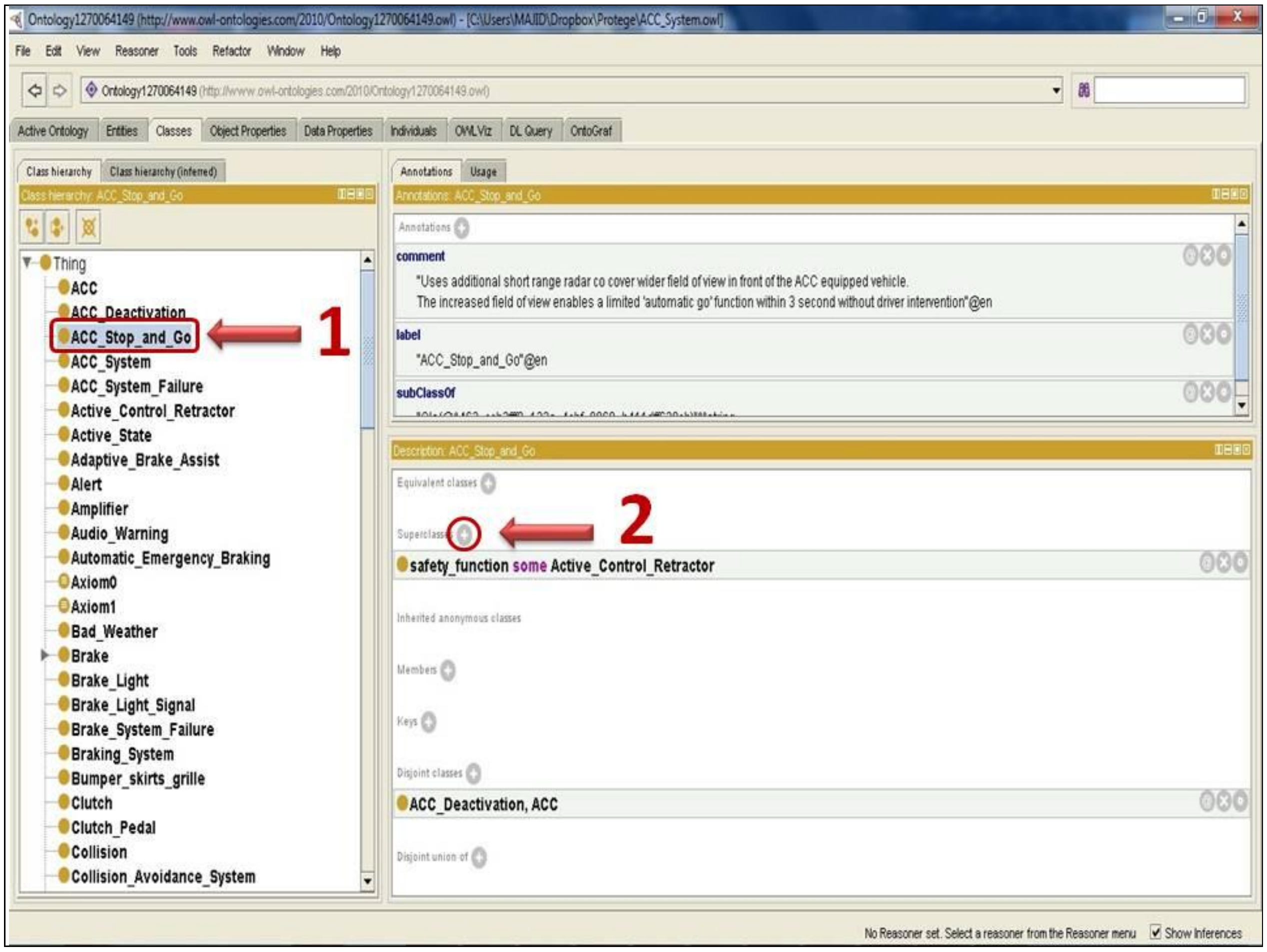
2. Select 'Class expression editor' tab at first **(Figure 5.33).**

3. Type 'safty_function' as the property to be restricted **(Figure 5.33).**

4. Type 'only' in order to create a universally quantified restriction **(Figure 5.33).**

5. Specify that the filler is 'Automatic_Emergency_Breaking'. to do this either: type 'Automatic_Emergency_Breaking' into the filler edit box, or drag and drop 'Automatic_Emergency_Breaking' into the text box **(Figure 5.33).**

6. Press 'OK' to close the dialog and create the restriction. If there are any errors (due to typing errors etc.) they will be underlined in red **(Figure 5.33).**


Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\MAJID\Dropbox\Protege\ACC_System.owl]
File Edit View Reasoner Tools Refactor Window Help
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Class hierarchy Class hierarchy (inferred)
Class hierarchy: ACC_Stop_and_Go
Thing
ACC
ACC_Deactivation
ACC_Stop_and_Go
1
ACC_System
ACC_System_Failure
Active_Control_Retractor
Active_State
Adaptive_Brake_Assist
Alert
Amplifier
Audio_Warning
Automatic_Emergency_Braking
Axiom0
Axiom1
Bad_Weather
Brake
Brake_Light
Brake_Light_Signal
Brake_System_Failure
Braking_System
Bumper_skirts_grille
Clutch
Clutch_Pedal
Collision
Collision_Avoidance_System
Annotations Usage
Annotations: ACC_Stop_and_Go
Annotations
Description: ACC_Stop_and_Go
Equivalent classes
Superclasses
2
Inherited anonymous classes
Members
Keys
Disjoint classes
Disjoint union of
Show Inferences

comment
"Uses additional short range radar co cover wider field of view in front of the ACC equipped vehicle.
The increased field of view enables a limited 'automatic go' function within 3 second without driver intervention"@en

label
"ACC_Stop_and_Go"@en

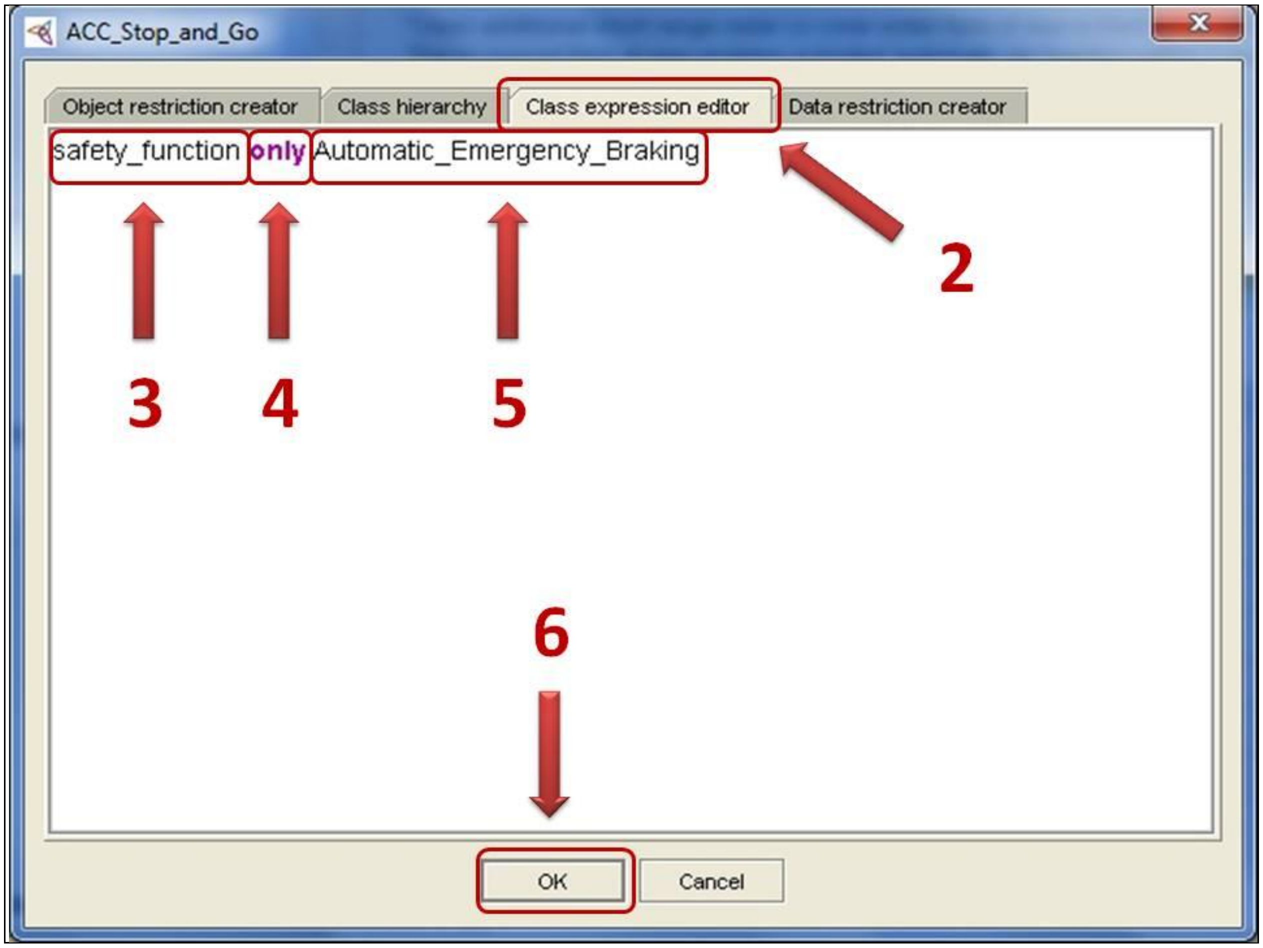
subClassOf

safety_function some Active_Control_Retractor

ACC_Deactivation, ACC

No Reasoner set. Select a reasoner from the Reasoner menu

Figure (5.32)



safety_function only Automatic_Emergency_Braking

Figure (5.33)

6. What is Reasoner?

One of the key features of ontologies that are described using OWL-DL is that they can be processed by a reasoner. One of the main services offered by a reasoner is to test whether or not one class is a subclass of another class. By performing such tests on the classes in an ontology it is possible for a reasoner to compute superclass-subclass relationships (subsumption relationships) automatically to infer ontology class hierarchy. Another standard service that is offered by reasoners is consistency checking. Based on the description (conditions) of a class the reasoner can check whether or not it is possible for the class to have any instances. A class is deemed to be inconsistent if it cannot possibly have any instances. So reasoner can help to maintain the hierarchy correctly.

6.1. Invoking The Reasoner:

It is possible to plug in various OWL reasoners to Protégé 4. Two versions of reasoner are shipped with Protégé are named FaCT++ and HermiT 1.3.6. In order to compute the classification hierarchy and also to check the logical consistency of the ontology automatically you can use the reasoner. (Figure 6.1) In Protégé 4 the `manually constructed' class hierarchy is called the asserted hierarchy. The class hierarchy that is automatically computed by the reasoner is called the inferred hierarchy. (Figure 6.2). If a class has been reclassified (i.e. if it's superclasses have changed) then the class name will appear in a blue color in the inferred hierarchy. If a class has been found to be inconsistent it's icon will be highlighted in red.

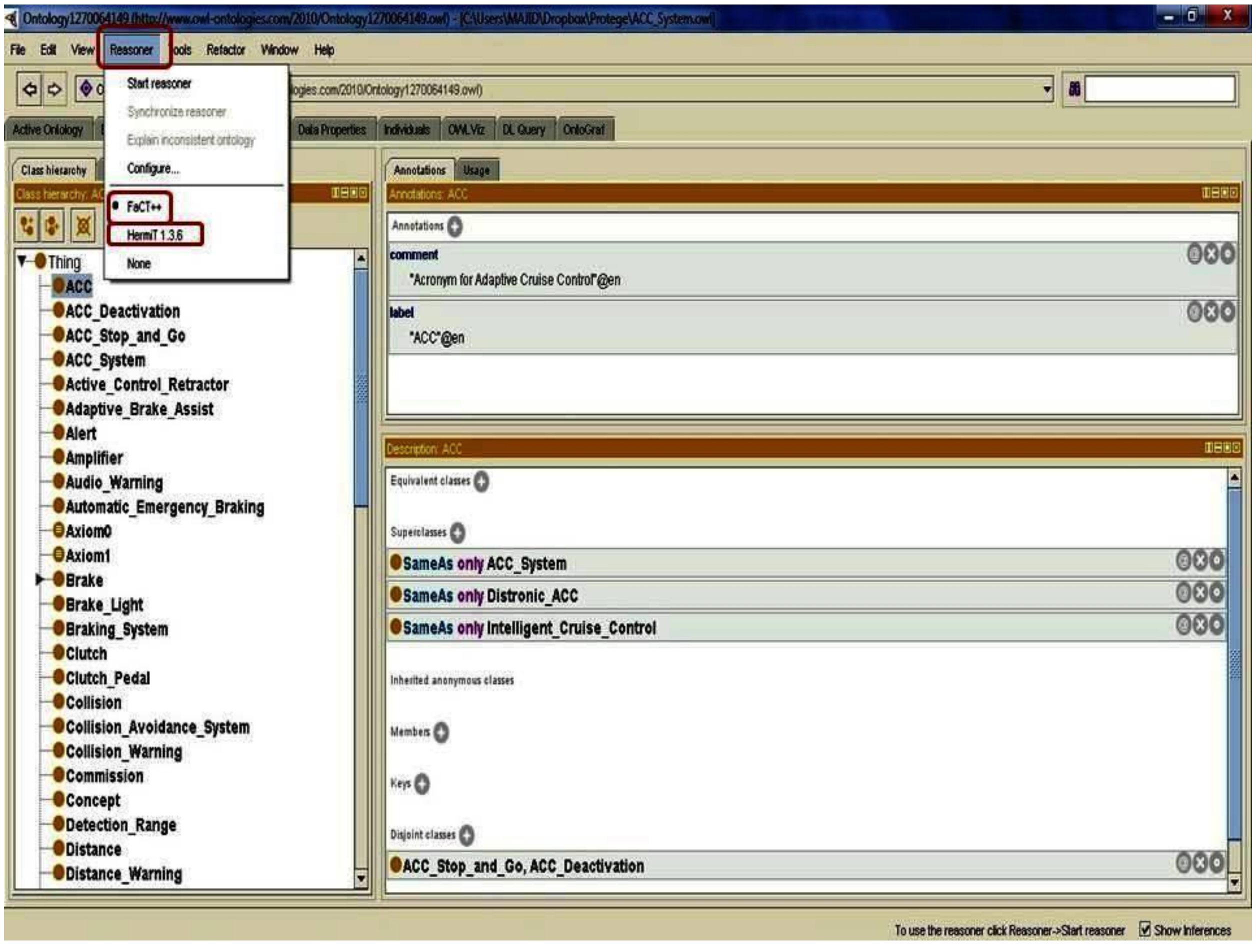

Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\MAJID\Dropbox\Protege\ACC_System.owl]
File
Edit
View
Reasoner
Tools
Refactor
Window
Help
Start reasoner
Synchronize reasoner
Explain inconsistent ontology
Configure...
FaCT++
HermiT 1.3.6
None
Active Ontology
Data Properties
Individuals
OWLViz
DL Query
OntoGraf
Class hierarchy
Annotations
Usage
Thing
ACC
ACC_Deactivation
ACC_Stop_and_Go
ACC_System
Active_Control_Retractor
Adaptive_Brake_Assist
Alert
Amplifier
Audio_Warning
Automatic_Emergency_Braking
Axiom0
Axiom1
Brake
Brake_Light
Braking_System
Clutch
Clutch_Pedal
Collision
Collision_Avoidance_System
Collision_Warning
Commission
Concept
Detection_Range
Distance
Distance_Warning
Annotations: ACC
Annotations
comment
"Acronym for Adaptive Cruise Control"@en
label
"ACC"@en
Description: ACC
Equivalent classes
Superclasses
SameAs only ACC_System
SameAs only Distronic_ACC
SameAs only Intelligent_Cruise_Control
Inherited anonymous classes
Members
Keys
Disjoint classes
ACC_Stop_and_Go, ACC_Deactivation
To use the reasoner click Reasoner->Start reasoner
Show Inferences

Figure (6.1)

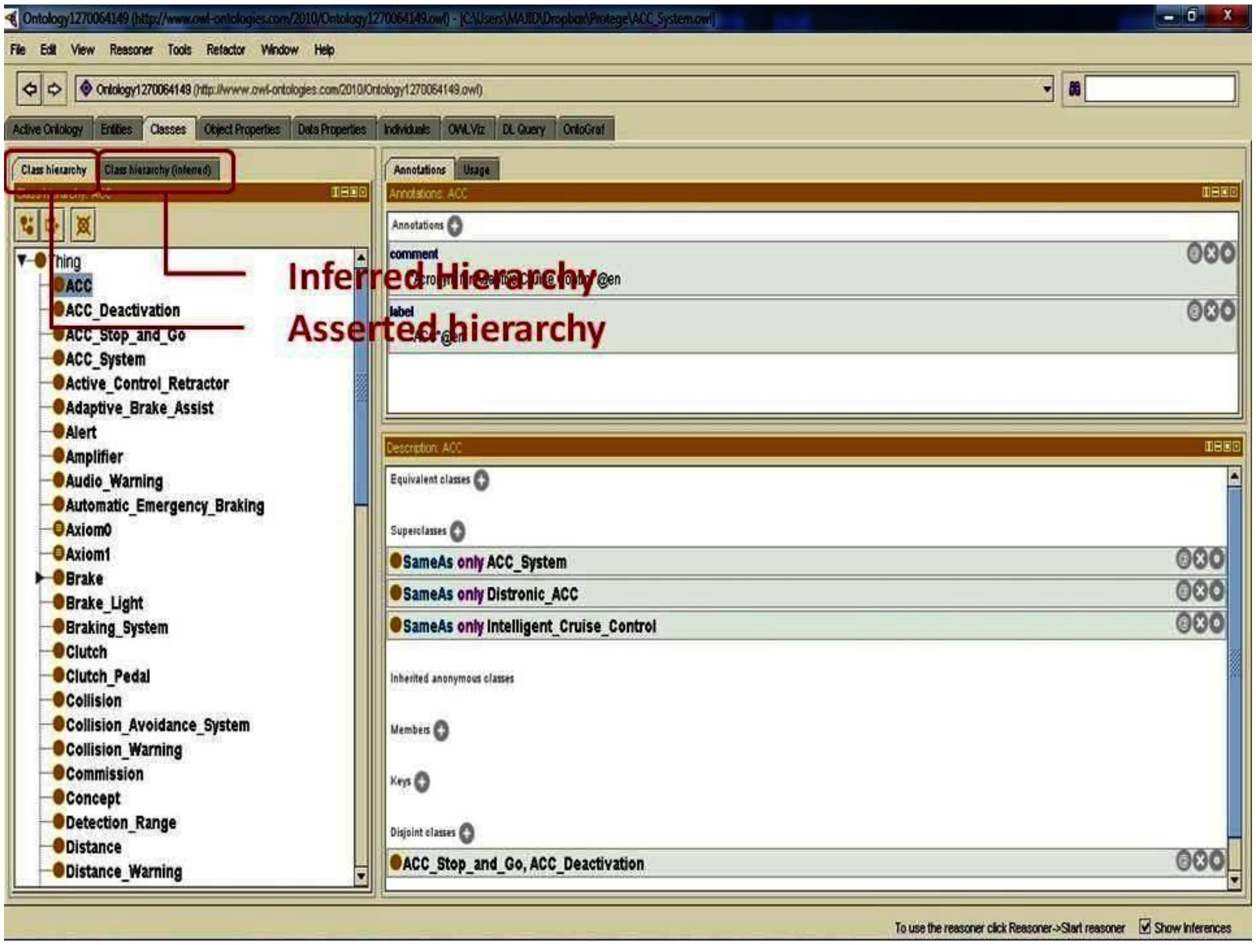
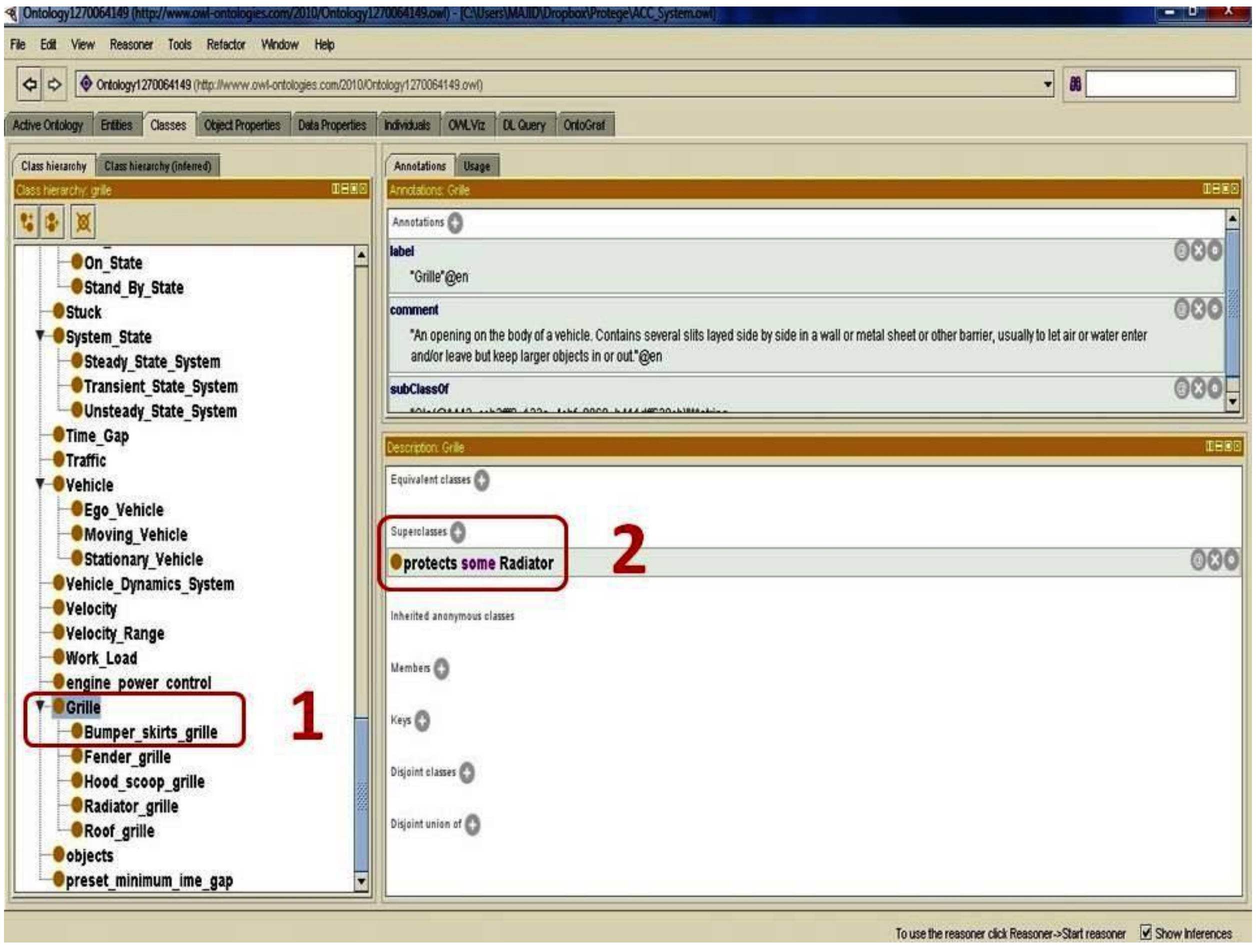


Figure (6.2)

6.2. Necessary And Sufficient Conditions (Primitive and Defined Classes):

All of the classes that we have created so far have only used necessary conditions to describe them. Necessary conditions can be translated as, (If something is a member of this class then it is necessary to fulfill these conditions. in other words, with necessary conditions alone, we cannot say that, If something fulfils these conditions then it must be a member of this class. Those classes that only has necessary conditions are known as Primitive Classes. Let's describe this with an example. At first we need to create Grille class and its subclass Bumper_skirts_grille as it described in section 5.2. Then as it has explained in section 5.9.1 a property restriction (protects some Radiator) has specified like the picture shown in (Figure 6.3).



label
"Grille"@en

comment
"An opening on the body of a vehicle. Contains several slits layed side by side in a wall or metal sheet or other barrier, usually to let air or water enter and/or leave but keep larger objects in or out."@en

subClassOf

protects some Radiator

Figure (6.3)

Our current description of Bumper_skirt_grille says that if something is a Bumper_skirt_grille it is necessarily a Grille and it is necessary for it to have at least one Protects that is a kind of Radiator. We have used necessary conditions to say this. Now consider some (random) individual. Suppose that we know that this individual is a member of the class Grille. We also know that this individual has at least one kind of Radiator. However, given our current description of Bumper_skirt_grille this knowledge is not sufficient to determine that the individual is a member of the class Bumper_skirt_grille. To make this possible we need to change the conditions for Bumper_skirt_grille from necessary conditions to necessary AND sufficient conditions. This means that not only are the conditions necessary for membership of the class Bumper_skirt_grille, they are also sufficient to determine that any (random) individual that satisfies them must be a member of the class Bumper_skirt_grille.

A class that has at least one set of necessary and sufficient conditions is known as a Defined Class. Also necessary conditions are simply called Superclasses in Protégé 4. Necessary and sufficient condition are called Equivalent classes.

In order to convert necessary conditions to necessary and sufficient conditions, the conditions must be moved from under the 'Superclasses' header in the class description view to be under the 'Equivalent classes' header. This can be done with the 'Convert to defined class' option in the 'Edit' menu.

In order to convert the necessary condition for Bumper_skirt_grille into necessary and sufficient conditions please follow these steps:

1. Ensure that Bumper_skirt_grille is selected in the class hierarchy.

2. Select ‘protects some Radiator’ property restriction in description view.

3. Right click on ‘protects some Radiator’, and then select ‘Convert selected rows to defined class’, shown in **Figure 6.4**, or you can go to Edit in menu bar and select the ‘ Convert to defined class ’ shown in **Figure 6.5**

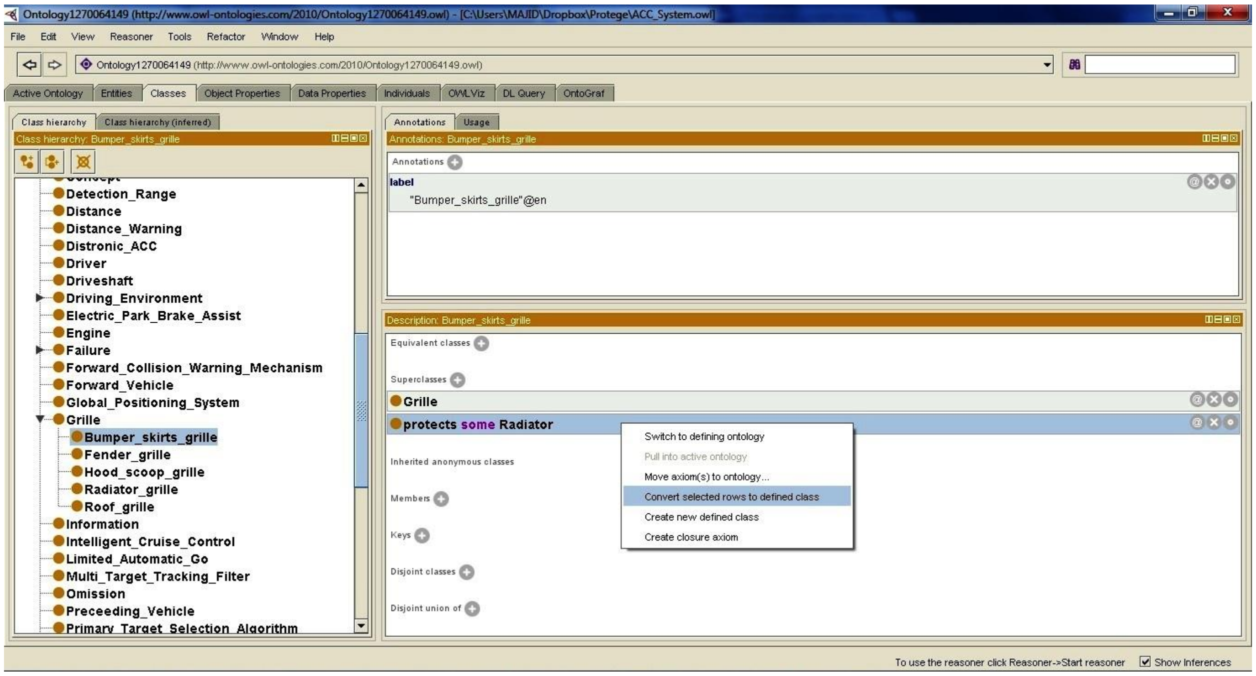

Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\MAJID\Dropbox\Protege\ACC_System.owl]
File Edit View Reasoner Tools Refactor Window Help
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Class hierarchy Class hierarchy (inferred)
Class hierarchy: Bumper_skirts_grille
Detection_Range
Distance
Distance_Warning
Distronic_ACC
Driver
Driveshaft
Driving_Environment
Electric_Park_Brake_Assist
Engine
Failure
Forward_Collision_Warning_Mechanism
Forward_Vehicle
Global_Positioning_System
Grille
Bumper_skirts_grille
Fender_grille
Hood_scoop_grille
Radiator_grille
Roof_grille
Information
Intelligent_Cruise_Control
Limited_Automatic_Go
Multi_Target_Tracking_Filter
Omission
Preceeding_Vehicle
Primary_Target_Selection_Algorithm
Annotations Usage
Annotations: Bumper_skirts_grille
Annotations
label
"Bumper_skirts_grille"@en
Description: Bumper_skirts_grille
Equivalent classes
Superclasses
Grille
protects some Radiator
Inherited anonymous classes
Members
Keys
Disjoint classes
Disjoint union of
Switch to defining ontology
Pull into active ontology
Move axiom(s) to ontology...
Convert selected rows to defined class
Create new defined class
Create closure axiom
To use the reasoner click Reasoner->Start reasoner
Show Inferences

Figure (6.4)

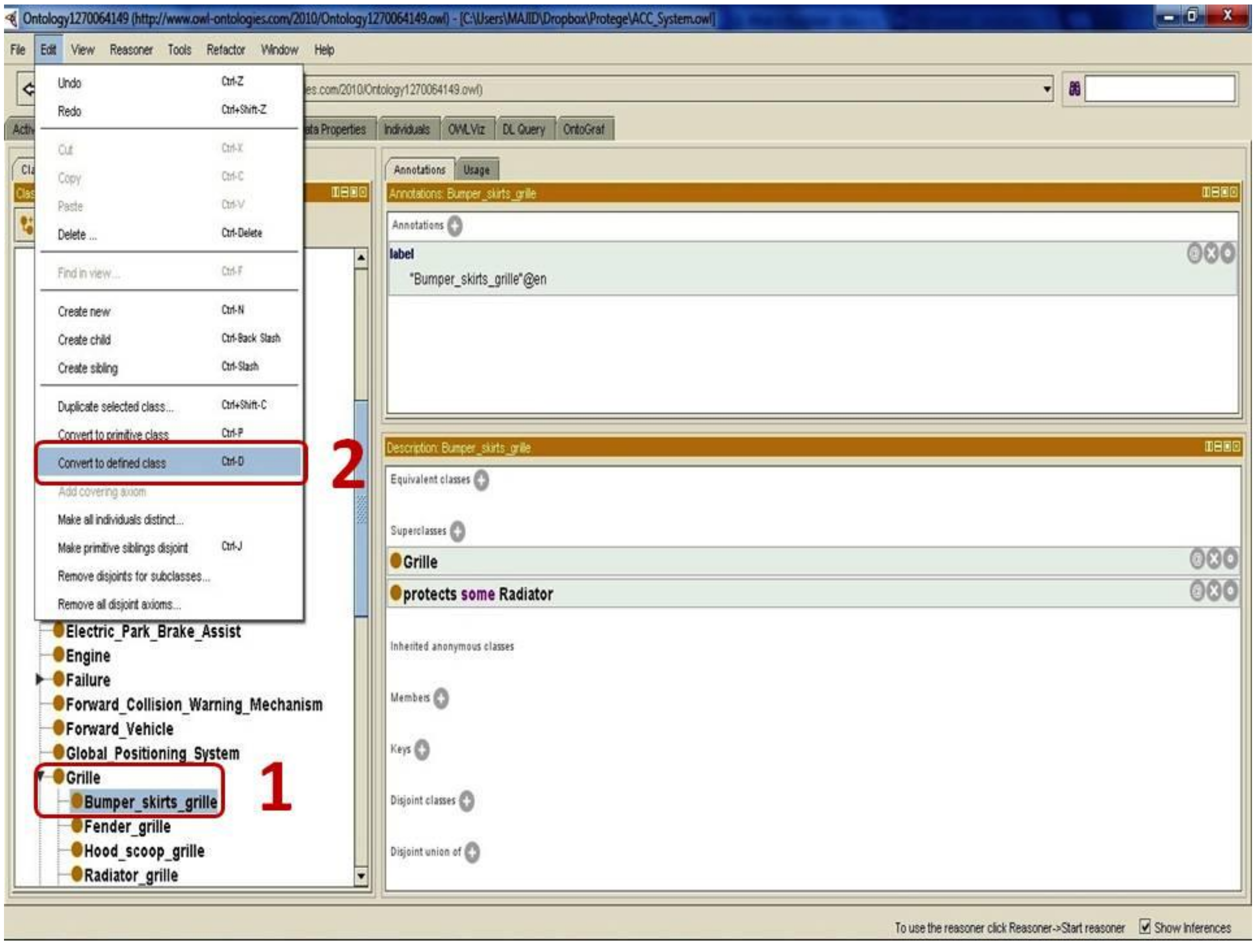
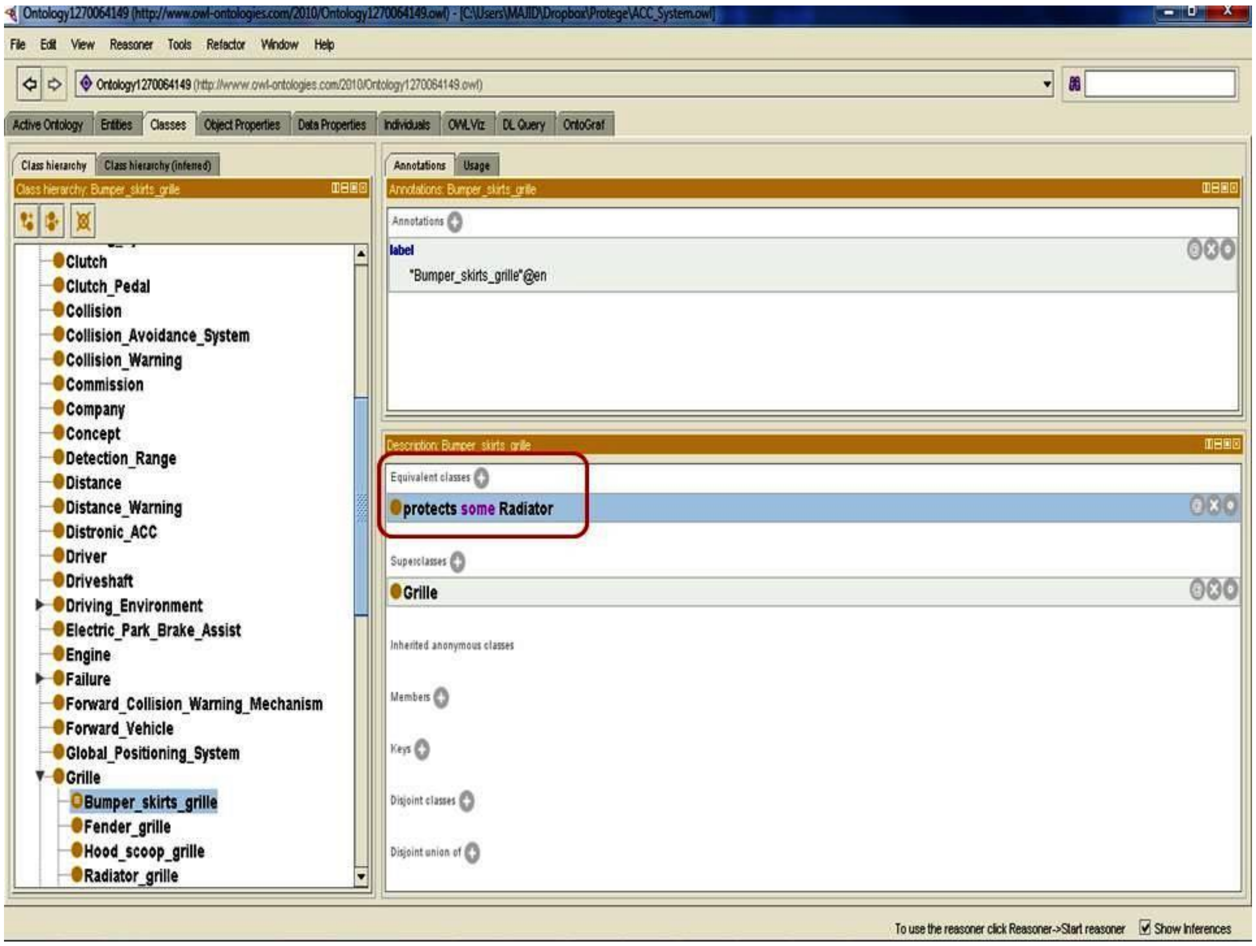

Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\MAJID\Dropbox\Protege\ACC_System.owl]
File Edit View Reasoner Tools Refactor Window Help
Undo Ctrl-Z
Redo Ctrl+Shift-Z
Cut Ctrl-X
Copy Ctrl-C
Paste Ctrl-V
Delete ... Ctrl-Delete
Find in view... Ctrl-F
Create new Ctrl-N
Create child Ctrl-Back Slash
Create sibling Ctrl-Slash
Duplicate selected class... Ctrl+Shift-C
Convert to primitive class Ctrl-P
Convert to defined class Ctrl-D
Add covering axiom
Make all individuals distinct...
Make primitive siblings disjoint Ctrl-J
Remove disjoints for subclasses...
Remove all disjoint axioms...
2
Individuals OWLViz DL Query OntoGraf
Annotations Usage
Annotations: Bumper_skirts_grille
label
"Bumper_skirts_grille"@en
Description: Bumper_skirts_grille
Equivalent classes
Superclasses
Grille
protects some Radiator
Inherited anonymous classes
Members
Keys
Disjoint classes
Disjoint union of
Electric_Park_Brake_Assist
Engine
Failure
Forward_Collision_Warning_Mechanism
Forward_Vehicle
Global_Positioning_System
Grille
Bumper_skirts_grille
Fender_grille
Hood_scoop_grille
Radiator_grille
1
To use the reasoner click Reasoner->Start reasoner
Show Inferences

Figure (6.5)

After conversion the class description view should look like picture shown in **Figure 6.6**. How is this useful in practice? Suppose we have another class B, and we know that any individuals that are members of class B also satisfy the conditions that define class A. We can determine that class B is subsumed by class A. In other words, B is a subclass of A. Checking for class subsumption is a key task of a description logic reasoner and we will use the reasoner to automatically compute a classification hierarchy in this way.


File Edit View Reasoner Tools Refactor Window Help
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Class hierarchy Class hierarchy (inferred)
Annotations Usage
Annotations
Equivalent classes
Superclasses
Inherited anonymous classes
Members
Keys
Disjoint classes
Disjoint union of
To use the reasoner click Reasoner->Start reasoner
Show Inferences

Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\MAJID\Dropbox\Protege\ACC_System.owl]

Class hierarchy: Bumper_skirts_grille

- Clutch
- Clutch_Pedal
- Collision
- Collision_Avoidance_System
- Collision_Warning
- Commission
- Company
- Concept
- Detection_Range
- Distance
- Distance_Warning
- Distronic_ACC
- Driver
- Driveshaft
- Driving_Environment
- Electric_Park_Brake_Assist
- Engine
- Failure
- Forward_Collision_Warning_Mechanism
- Forward_Vehicle
- Global_Positioning_System
- Grille
  - Bumper_skirts_grille
  - Fender_grille
  - Hood_scoop_grille
  - Radiator_grille

Annotations: Bumper_skirts_grille

label
"Bumper_skirts_grille"@en

Description: Bumper_skirts_grille

Equivalent classes
protects some Radiator

Superclasses
Grille

Figure (6.6)

6.3 Primitive And Defined Classes

Classes that have at least one set of necessary and sufficient conditions are known as defined classes. They have a definition, and any individual that satisfies the definition will belong to the class. Classes that do not have any sets of necessary and sufficient conditions (only have necessary conditions) are known as primitive classes. In Protégé 4 defined classes have a class icon with three horizontal white lines in them. Primitive classes have a class icon that has a plain yellow background. It is also important to understand that the reasoner can only automatically classify classes under defined classes - i.e. classes with at least one set of necessary and sufficient conditions.

6.4 Automated Classification

Being able to use a reasoner to automatically compute the class hierarchy is one of the major benefits of building an ontology using the OWL-DL sub-language. Indeed, when constructing very large ontologies (with upwards of several thousand classes in them) the use of a reasoner to compute subclass-superclass relationships between classes becomes almost vital. Without a reasoner it is very difficult to keep large ontologies in a maintainable and logically correct state. In cases where ontologies can have classes that have many superclasses (multiple inheritance) it is nearly always a good idea to construct the class hierarchy as a simple tree. Classes in the asserted hierarchy (manually constructed hierarchy) therefore have no more than one superclass. Computing and maintaining multiple inheritance is the job of the reasoner. This technique helps to keep the ontology in a maintainable and modular state. Not only does this promote the reuse of the ontology by other ontologies and applications, it also minimises human errors that are inherent in maintaining a multiple inheritance hierarchy. Having created a definition of a Bumper_skirt_grille we can use the reasoner to automatically compute the subclasses of Bumper_skirt_grille.

In order to use reasoner and please follow these steps:

1. Select Reasoner in menu bar.

2. Select Start reasoner as shown in **Figure 6.7.**

After pushing reasoner the inferred hierarchy is computed and you can see the result in inferred hierarchy window as shown in **Figure 6.8.**

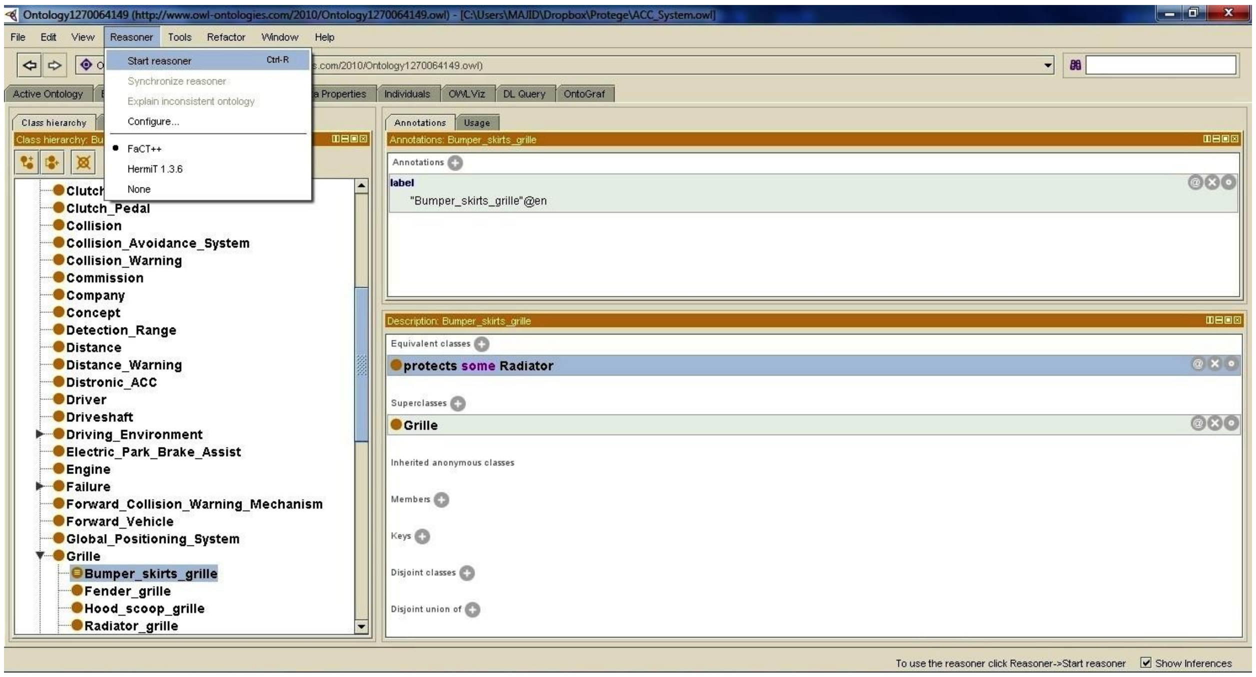
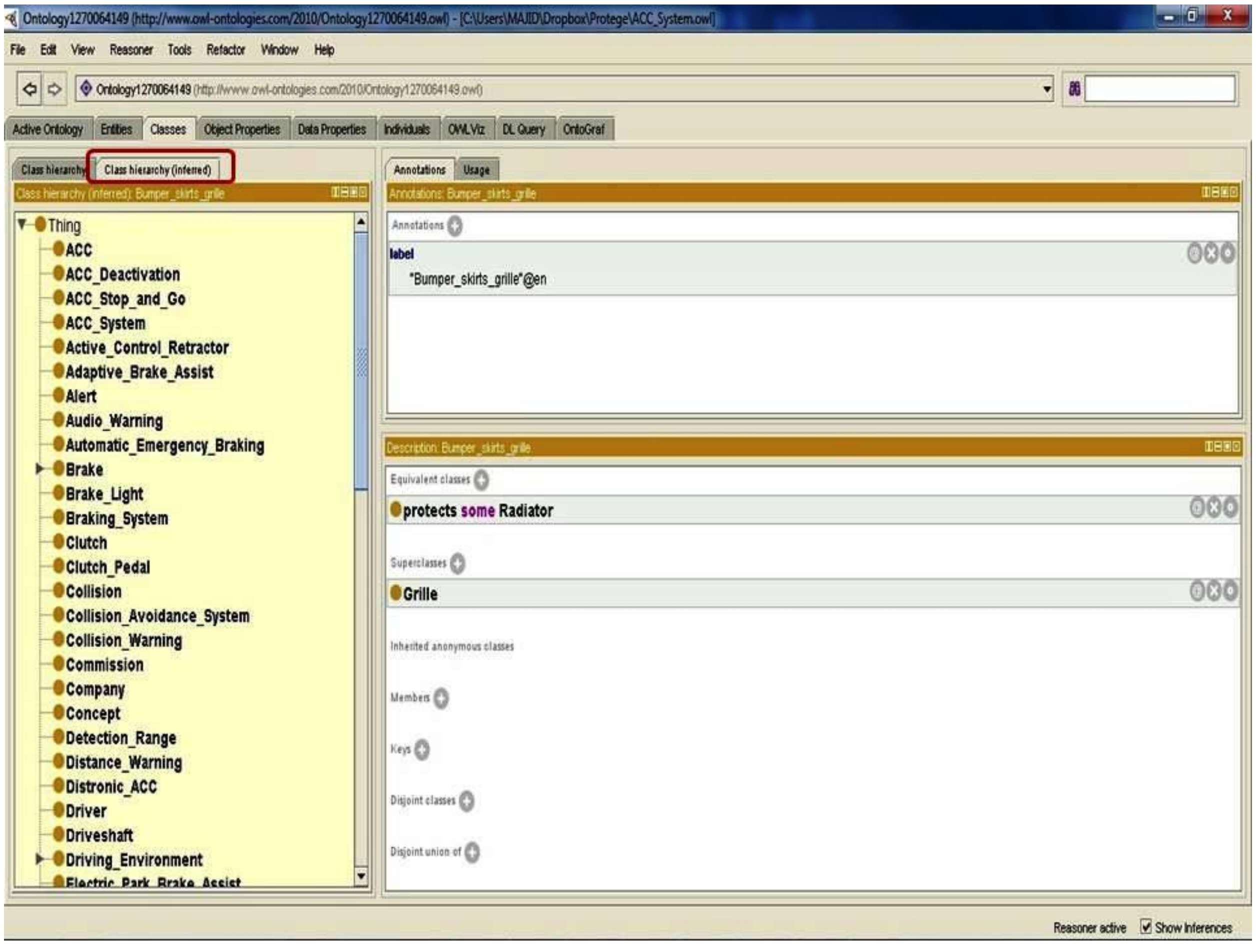

Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\MAJID\Dropbox\Protege\ACC_System.owl]
File Edit View Reasoner Tools Refactor Window Help
Start reasoner Ctrl-R
Synchronize reasoner
Explain inconsistent ontology
Configure...
FaCT++
HermiT 1.3.6
None
Active Ontology
Individuals
OWLViz
DL Query
OntoGraf
Class hierarchy
Annotations
Usage
Clutch
Clutch_Pedal
Collision
Collision_Avoidance_System
Collision_Warning
Commission
Company
Concept
Detection_Range
Distance
Distance_Warning
Distronic_ACC
Driver
Driveshaft
Driving_Environment
Electric_Park_Brake_Assist
Engine
Failure
Forward_Collision_Warning_Mechanism
Forward_Vehicle
Global_Positioning_System
Grille
Bumper_skirts_grille
Fender_grille
Hood_scoop_grille
Radiator_grille
Annotations: Bumper_skirts_grille
Annotations
label
"Bumper_skirts_grille"@en
Description: Bumper_skirts_grille
Equivalent classes
protects some Radiator
Superclasses
Grille
Inherited anonymous classes
Members
Keys
Disjoint classes
Disjoint union of
To use the reasoner click Reasoner->Start reasoner
Show Inferences

Figure (6.7)


Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\MAJID\Dropbox\Protege\ACC_System.owl]
File Edit View Reasoner Tools Refactor Window Help
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Class hierarchy Class hierarchy (inferred)
Annotations Usage
Reasoner active Show Inferences

Class hierarchy (inferred): Bumper_skirts_grille

- Thing
  - ACC
  - ACC_Deactivation
  - ACC_Stop_and_Go
  - ACC_System
  - Active_Control_Retractor
  - Adaptive_Brake_Assist
  - Alert
  - Audio_Warning
  - Automatic_Emergency_Braking
  - Brake
  - Brake_Light
  - Braking_System
  - Clutch
  - Clutch_Pedal
  - Collision
  - Collision_Avoidance_System
  - Collision_Warning
  - Commission
  - Company
  - Concept
  - Detection_Range
  - Distance_Warning
  - Distronic_ACC
  - Driver
  - Driveshaft
  - Driving_Environment
  - Electric_Park_Brake_Assist

Annotations: Bumper_skirts_grille

Annotations
label
"Bumper_skirts_grille"@en

Description: Bumper_skirts_grille

Equivalent classes
protects some Radiator

Superclasses
Grille

Inherited anonymous classes

Members

Keys

Disjoint classes

Disjoint union of

Figure (6.8)

6.5 Closure Axioms

A closure axiom on a property consists of a universal restriction that acts along the property to say that it can only be filled by the specified fillers. The restriction has a filler that is the union of the fillers that occur in the existential restrictions for the same property. For example, the closure axiom on the about property for Information is a universal restriction that acts along the about property, with a filler that is the union of Alert, Collision, Distance, Failure and Preceding_Vehicle.

In order to add a closure axiom on the 'about' property for 'Information' follow these steps:

1. Select Information class in class hierarchy on the class tab. **(Figure 6.9).**

2. Press the 'Add' icon next to the 'Superclasses' section of the 'Class Description' view to open the edit text box. **(Figure 6.9).**

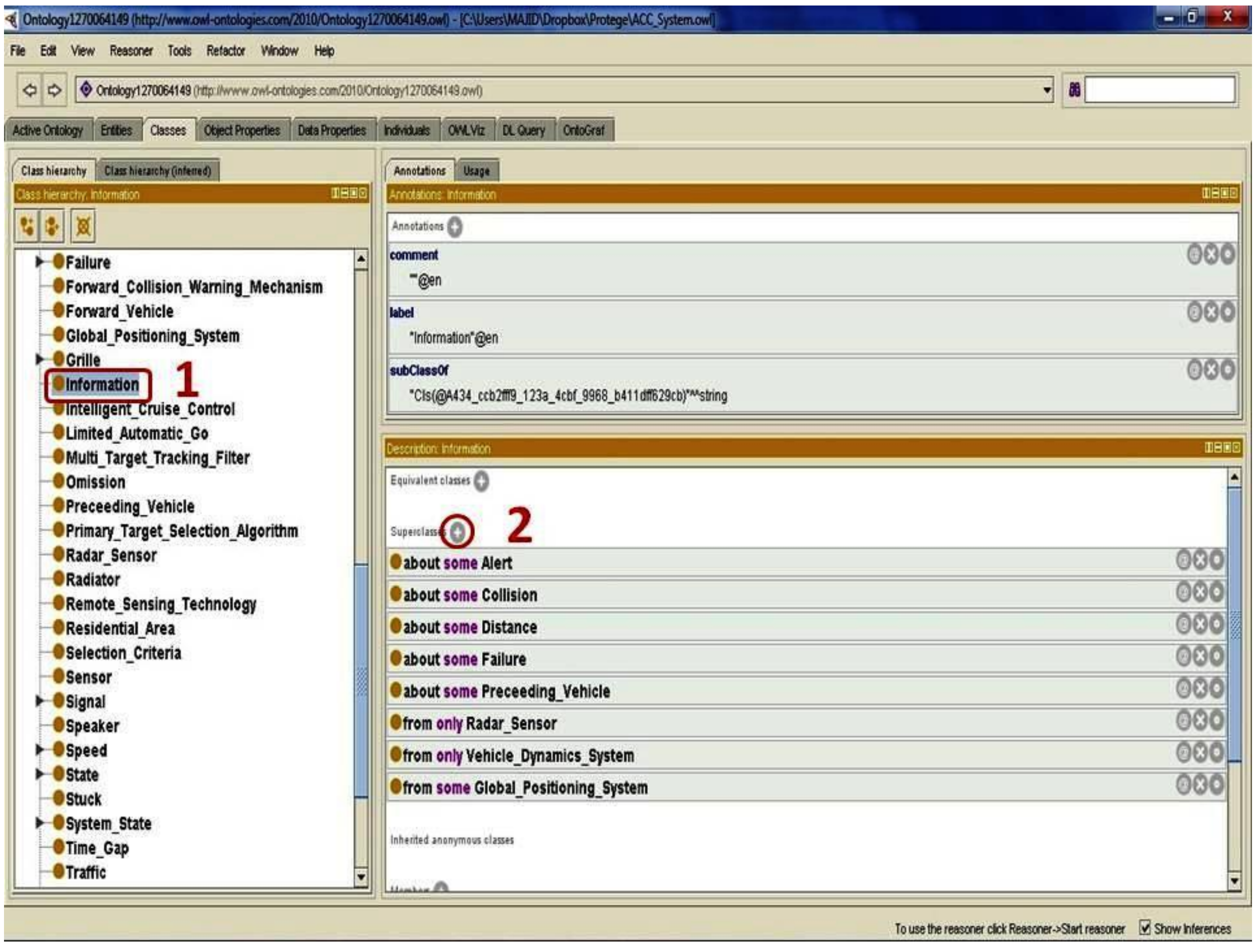
3. Type about as the property to be restricted. **(Figure 6.10).**

4. Type ‘only’ to create the universal restriction. **(Figure 6.10).**

5. Open brackets and type 'Alert or Collision or Distance or Failure or Preceding_Vehicle' and then close bracket. **(Figure 6.10).**

6. Press 'OK' to create the restriction and add it to the class Information. **(Figure 6.10).**

7. The result on description view should be like picture in **Figure 6.11**.

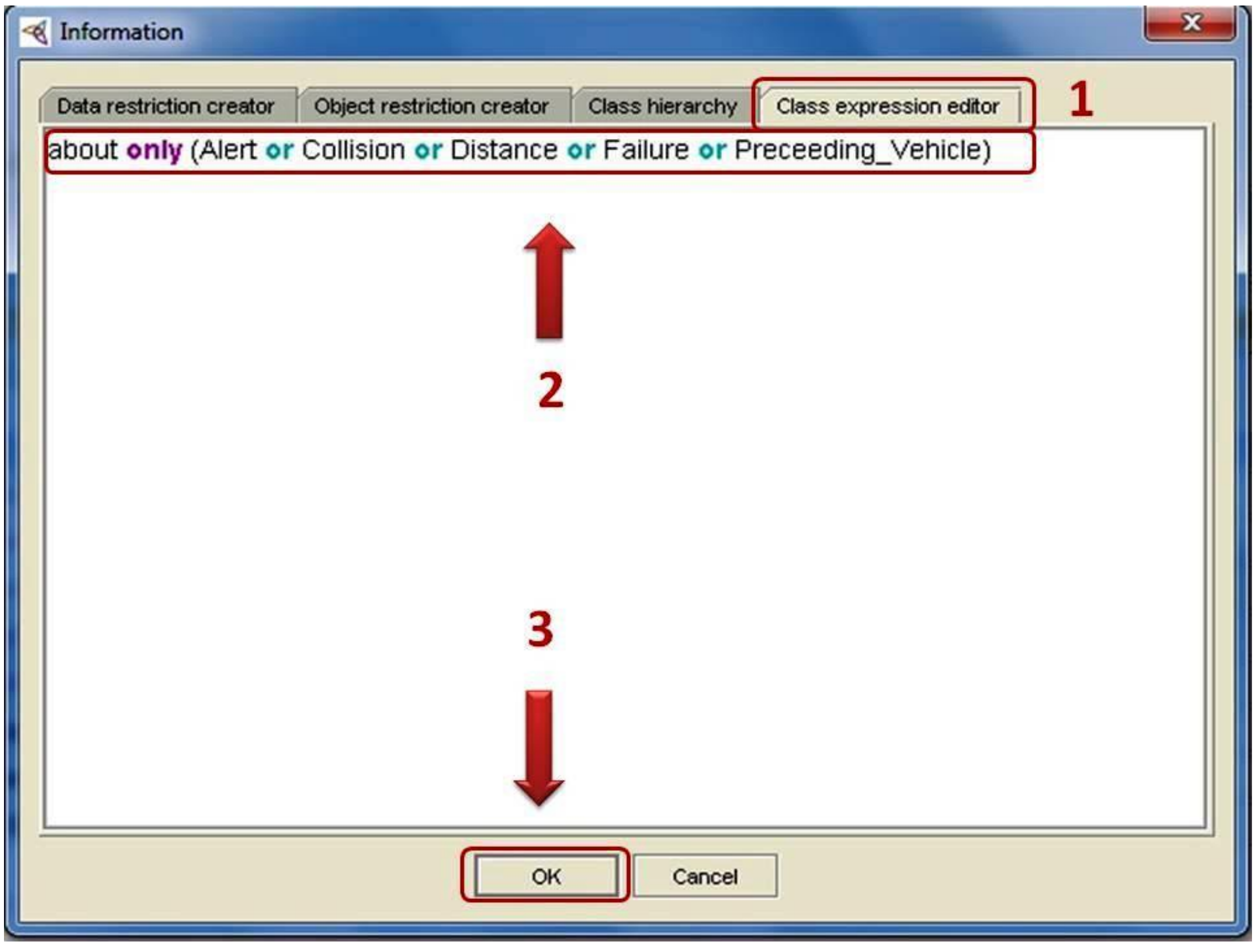

Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\MAJID\Dropbox\Protege\ACC_System.owl]
File Edit View Reasoner Tools Refactor Window Help
Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Class hierarchy Class hierarchy (inferred)
Class hierarchy: Information
Annotations Usage
Annotations: Information
Description: Information
1
2
To use the reasoner click Reasoner->Start reasoner
Show Inferences

Failure
Forward_Collision_Warning_Mechanism
Forward_Vehicle
Global_Positioning_System
Grille
Information
Intelligent_Cruise_Control
Limited_Automatic_Go
Multi_Target_Tracking_Filter
Omission
Preceeding_Vehicle
Primary_Target_Selection_Algorithm
Radar_Sensor
Radiator
Remote_Sensing_Technology
Residential_Area
Selection_Criteria
Sensor
Signal
Speaker
Speed
State
Stuck
System_State
Time_Gap
Traffic

Annotations
comment
""@en
label
"Information"@en
subClassOf
"Cls(@A434_ccb2fff9_123a_4cbf_9968_b411dff629cb)"^^string

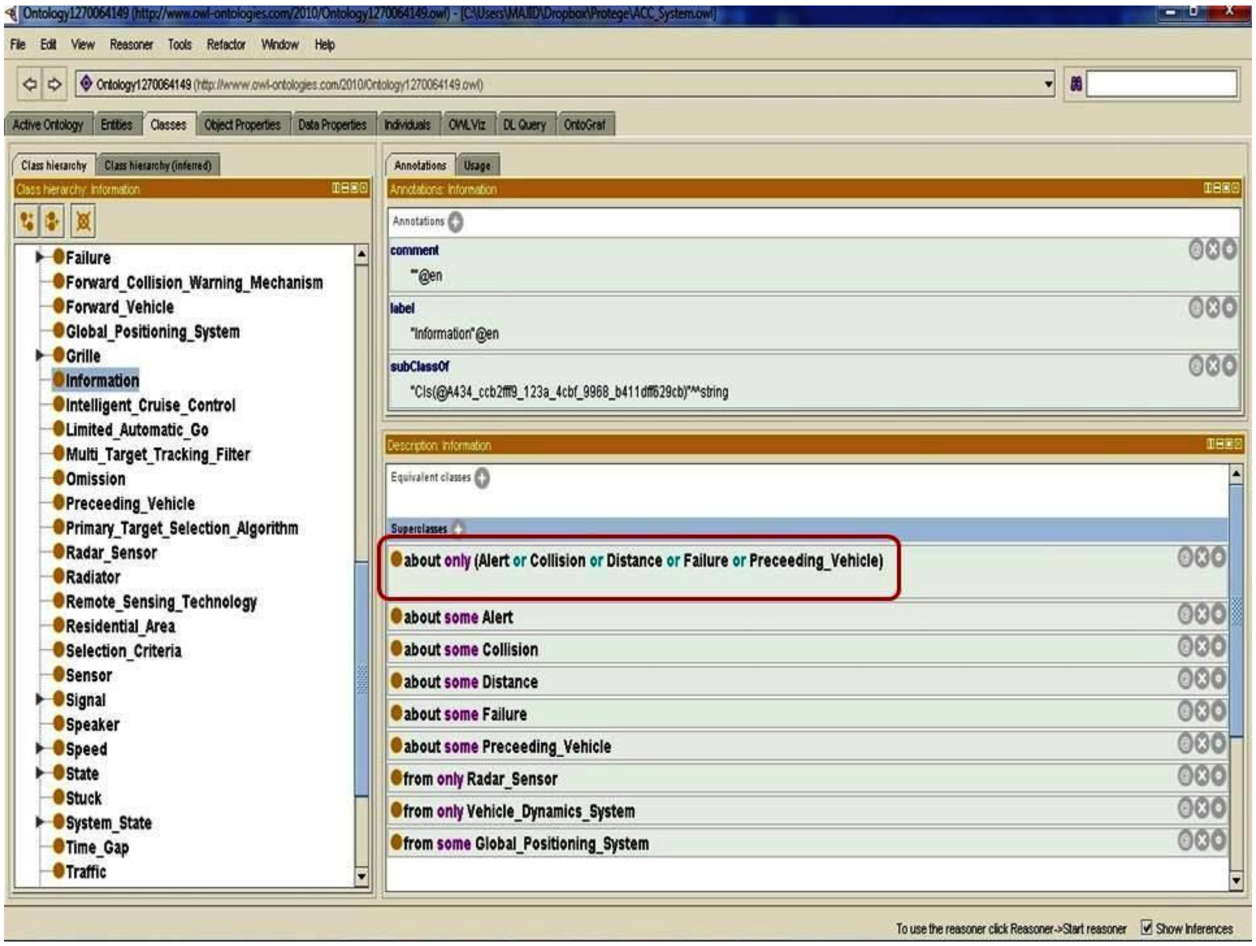
Equivalent classes
Superclasses
about some Alert
about some Collision
about some Distance
about some Failure
about some Preceeding_Vehicle
from only Radar_Sensor
from only Vehicle_Dynamics_System
from some Global_Positioning_System
Inherited anonymous classes

Figure (6.9)



```
about only (Alert or Collision or Distance or Failure or Preceeding_Vehicle)
```

Figure (6.10)


Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\MAJID\Dropbox\Protege\ACC_System.owl]
File Edit View Reasoner Tools Refactor Window Help
Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Class hierarchy Class hierarchy (inferred)
Class hierarchy: Information
Annotations Usage
Annotations: Information
Description: Information
To use the reasoner click Reasoner->Start reasoner Show Inferences

Failure
Forward_Collision_Warning_Mechanism
Forward_Vehicle
Global_Positioning_System
Grille
Information
Intelligent_Cruise_Control
Limited_Automatic_Go
Multi_Target_Tracking_Filter
Omission
Preceeding_Vehicle
Primary_Target_Selection_Algorithm
Radar_Sensor
Radiator
Remote_Sensing_Technology
Residential_Area
Selection_Criteria
Sensor
Signal
Speaker
Speed
State
Stuck
System_State
Time_Gap
Traffic

Annotations
comment
""@en
label
"Information"@en
subClassOf
"Cls(@A434_ccb2ff9_123a_4cbf_9968_b411dff629cb)"^^string

Equivalent classes

Superclasses
about only (Alert or Collision or Distance or Failure or Preceeding_Vehicle)
about some Alert
about some Collision
about some Distance
about some Failure
about some Preceeding_Vehicle
from only Radar_Sensor
from only Vehicle_Dynamics_System
from some Global_Positioning_System

Figure (6.11)

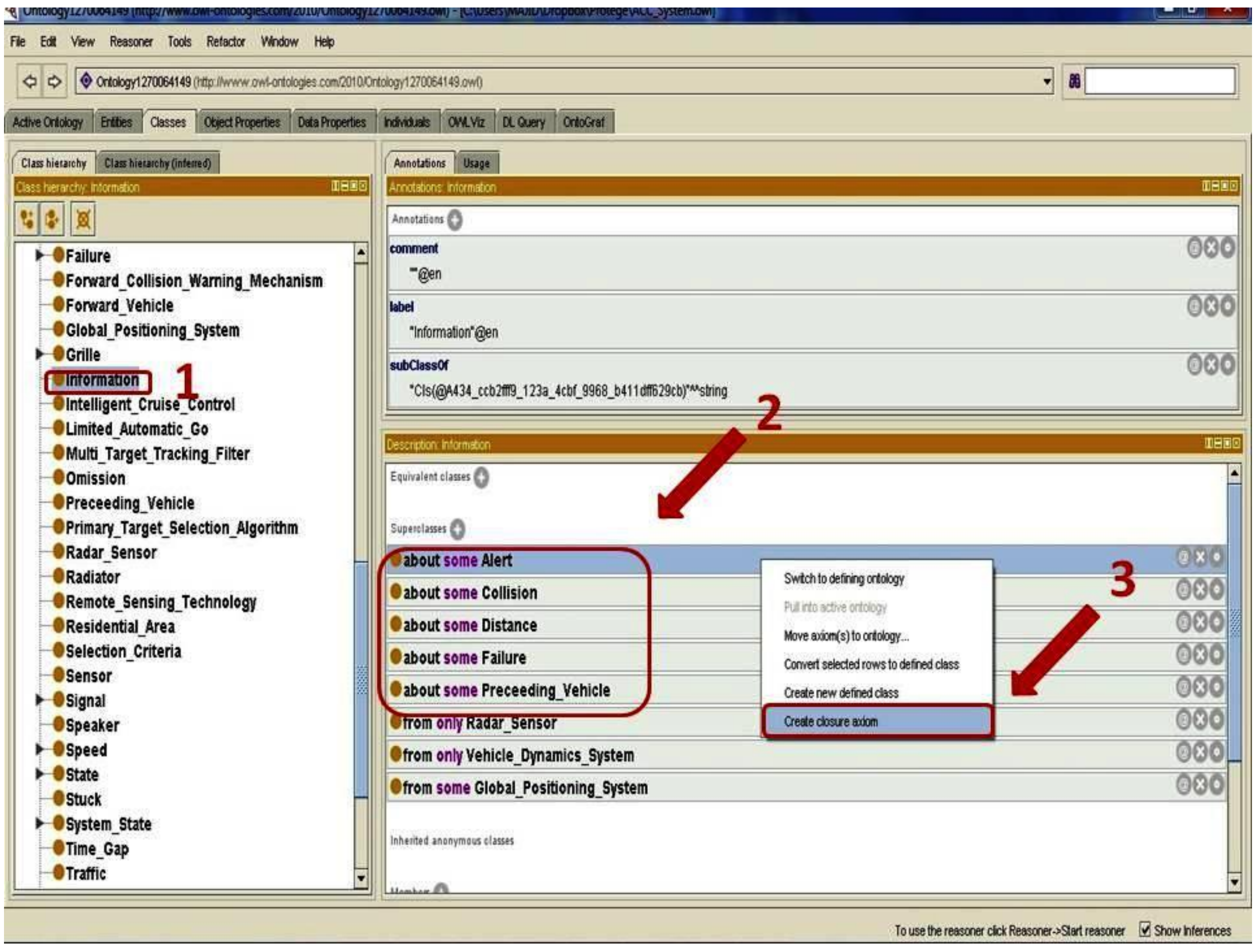
There is another way to add a closure axiom on a property for a class that is easier and faster to do. In following steps you can find how to do it on 'about' property for 'Information':

1. Make sure that you have selected Information in class hierarchy. **(Figure 6.12).**

2. Select one of the about property restrictions on the property description view. **(Figure 6.12).**

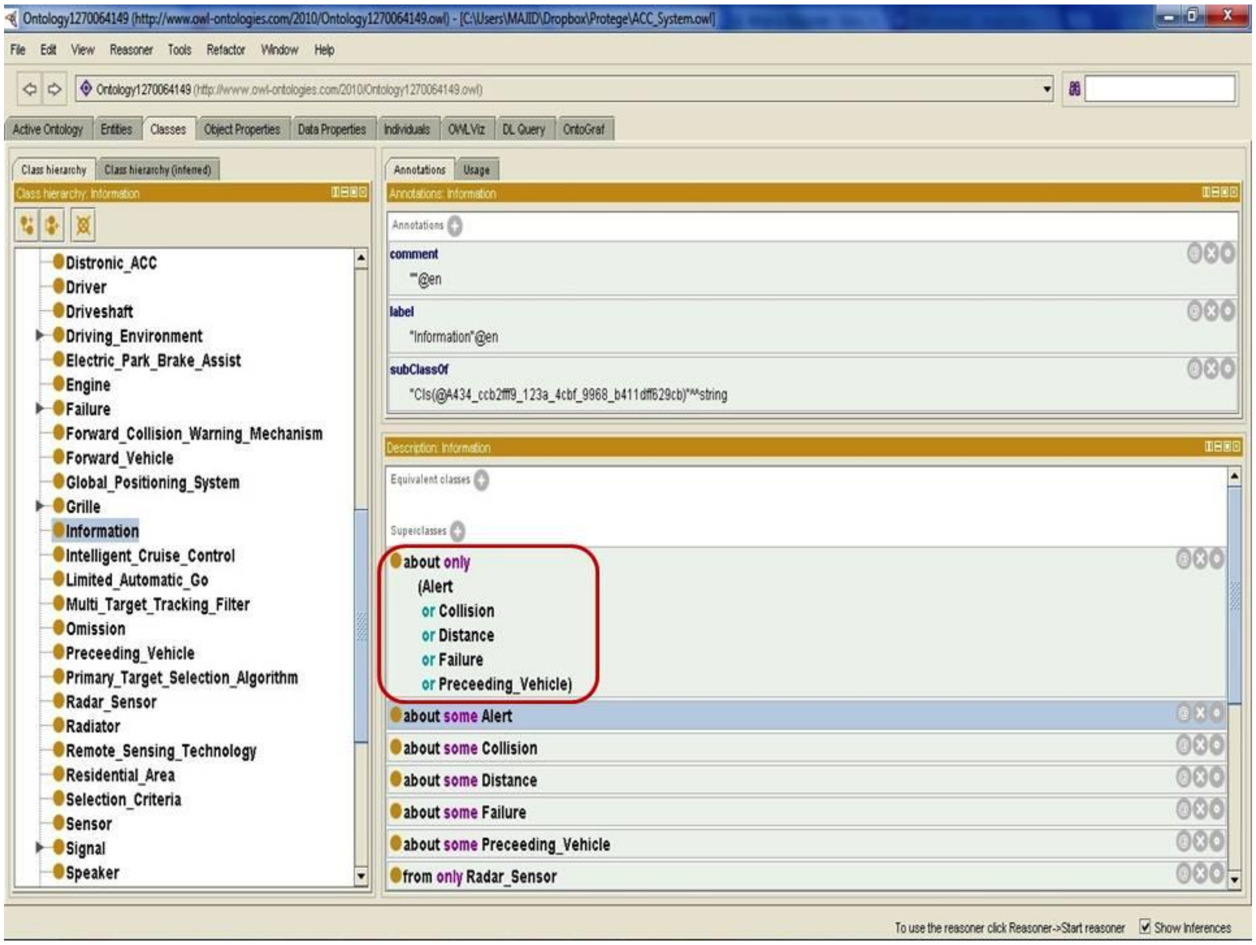
3. Right click on the selected property restriction and then choose Create closure axiom.(**Figure 6.12).**

4. The result on description view should be like picture in **(Figure 6.13).**



File Edit View Reasoner Tools Refactor Window Help
Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl)
Active Ontology
Entities
Classes
Object Properties
Data Properties
Individuals
OWLViz
DL Query
OntoGraf
Class hierarchy
Class hierarchy (inferred)
Class hierarchy: Information
Failure
Forward_Collision_Warning_Mechanism
Forward_Vehicle
Global_Positioning_System
Grille
Information
Intelligent_Cruise_Control
Limited_Automatic_Go
Multi_Target_Tracking_Filter
Omission
Preceeding_Vehicle
Primary_Target_Selection_Algorithm
Radar_Sensor
Radiator
Remote_Sensing_Technology
Residential_Area
Selection_Criteria
Sensor
Signal
Speaker
Speed
State
Stuck
System_State
Time_Gap
Traffic
1
Annotations
Usage
Annotations: Information
Annotations
comment
""@en
label
"Information"@en
subClassOf
"Cls(@A434_ccb2ff9_123a_4cbf_9968_b411dff629cb)"^^string
2
Description: Information
Equivalent classes
Superclasses
about some Alert
about some Collision
about some Distance
about some Failure
about some Preceeding_Vehicle
from only Radar_Sensor
from only Vehicle_Dynamics_System
from some Global_Positioning_System
Inherited anonymous classes
Switch to defining ontology
Pull into active ontology
Move axiom(s) to ontology...
Convert selected rows to defined class
Create new defined class
Create closure axiom
3
To use the reasoner click Reasoner->Start reasoner
Show Inferences

Figure (6.12)


Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\MAJID\Dropbox\Protege\ACC_System.owl]
File Edit View Reasoner Tools Refactor Window Help
Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl)
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Class hierarchy Class hierarchy (inferred)
Class hierarchy: Information
Annotations Usage
Annotations: Information
Description: Information
To use the reasoner click Reasoner->Start reasoner
Show Inferences

Distronic_ACC
Driver
Driveshaft
Driving_Environment
Electric_Park_Brake_Assist
Engine
Failure
Forward_Collision_Warning_Mechanism
Forward_Vehicle
Global_Positioning_System
Grille
Information
Intelligent_Cruise_Control
Limited_Automatic_Go
Multi_Target_Tracking_Filter
Omission
Preceeding_Vehicle
Primary_Target_Selection_Algorithm
Radar_Sensor
Radiator
Remote_Sensing_Technology
Residential_Area
Selection_Criteria
Sensor
Signal
Speaker

Annotations
comment
""@en
label
"Information"@en
subClassOf
"Cls(@A434_ccb2fff9_123a_4cbf_9968_b411dff629cb)"^^string

Equivalent classes
Superclasses
about only
(Alert
or Collision
or Distance
or Failure
or Preceeding_Vehicle)
about some Alert
about some Collision
about some Distance
about some Failure
about some Preceeding_Vehicle
from only Radar_Sensor

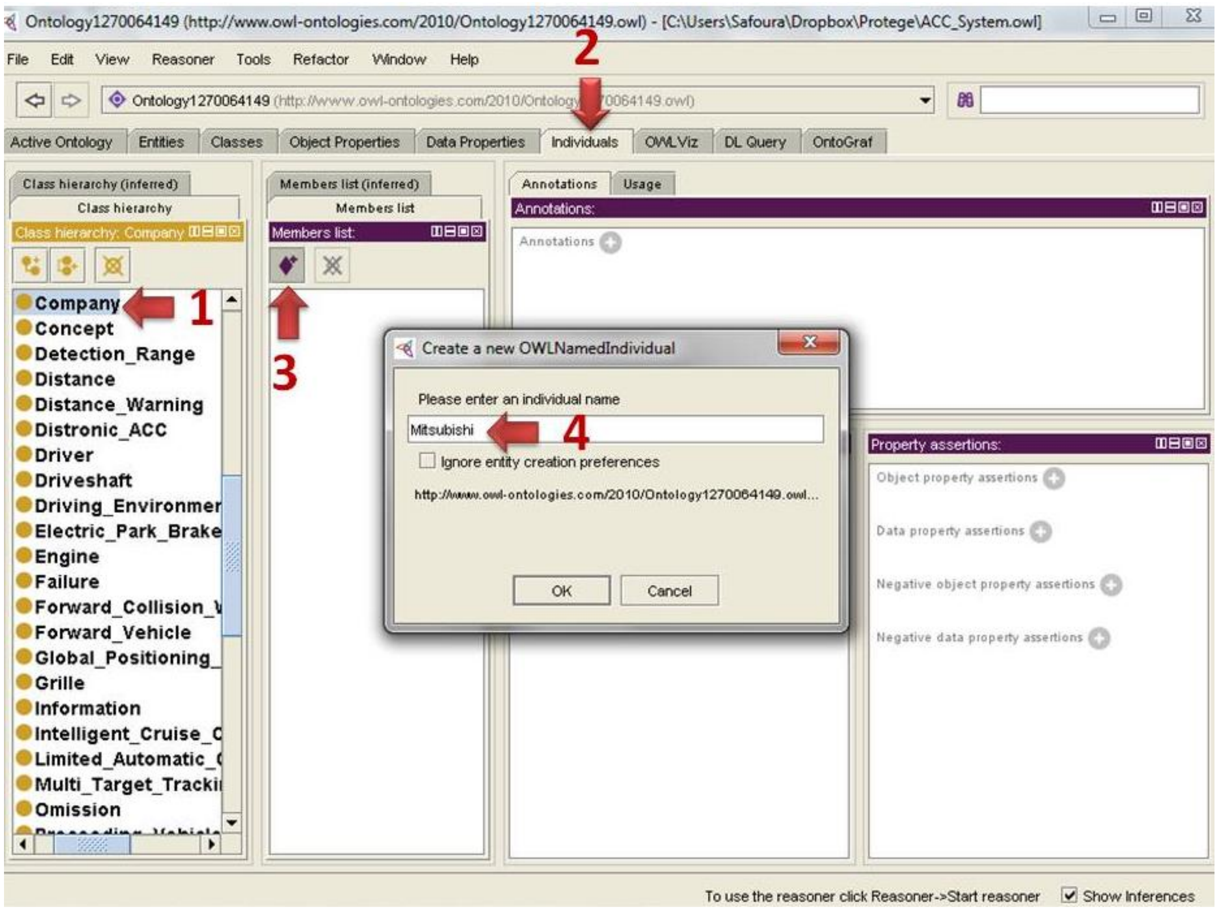
Figure (6.13)

7. Creating Individuals

In OWL we can define individuals. Consider we wanted to define different companies which have been offered ACC systems. In order to do that, first we have to create a class namely 'Company', then we can define different company instances (individuals) for class 'Company' such as 'Mitsubishi', 'Toyota', ' Lexus', 'Jaguar', Audi'' and … . Notice that instances are different from subclasses. To create individuals in Protégé 4 the 'Individuals Tab' is used.

Now we will create a class called 'Company' and populate it with some individuals:

1. Create Company as a subclass of Thing. Shown in **(Figure 7.1)**.


Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\Safoura\Dropbox\Protege\ACC_System.owl]

Create a new OWLNamedIndividual

Please enter an individual name

Mitsubishi

Ignore entity creation preferences

http://www.owl-ontologies.com/2010/Ontology1270064149.owl...

To use the reasoner click Reasoner->Start reasoner

Figure (7.1)

2. Switch to the ‘Individuals Tab’.

3. Press the ‘Add individual’ button.

4. Name the new individual ‘Mitsubishi’, press ‘Ok’.

5. In the individual ‘Description’ view located at the centre of the Individual tab, make sure that the class ‘Company’ to be the ‘Type’ for this individual. If it’s ‘Type’ is not Company, select the ‘Add’ icon next to the ‘Types’ header from the individual ‘Description’ view, choose class ‘Company’ from the class hierarchy, this will make ‘Mitsubishi’ an individual of the class Company **(Figure 7.2)**.

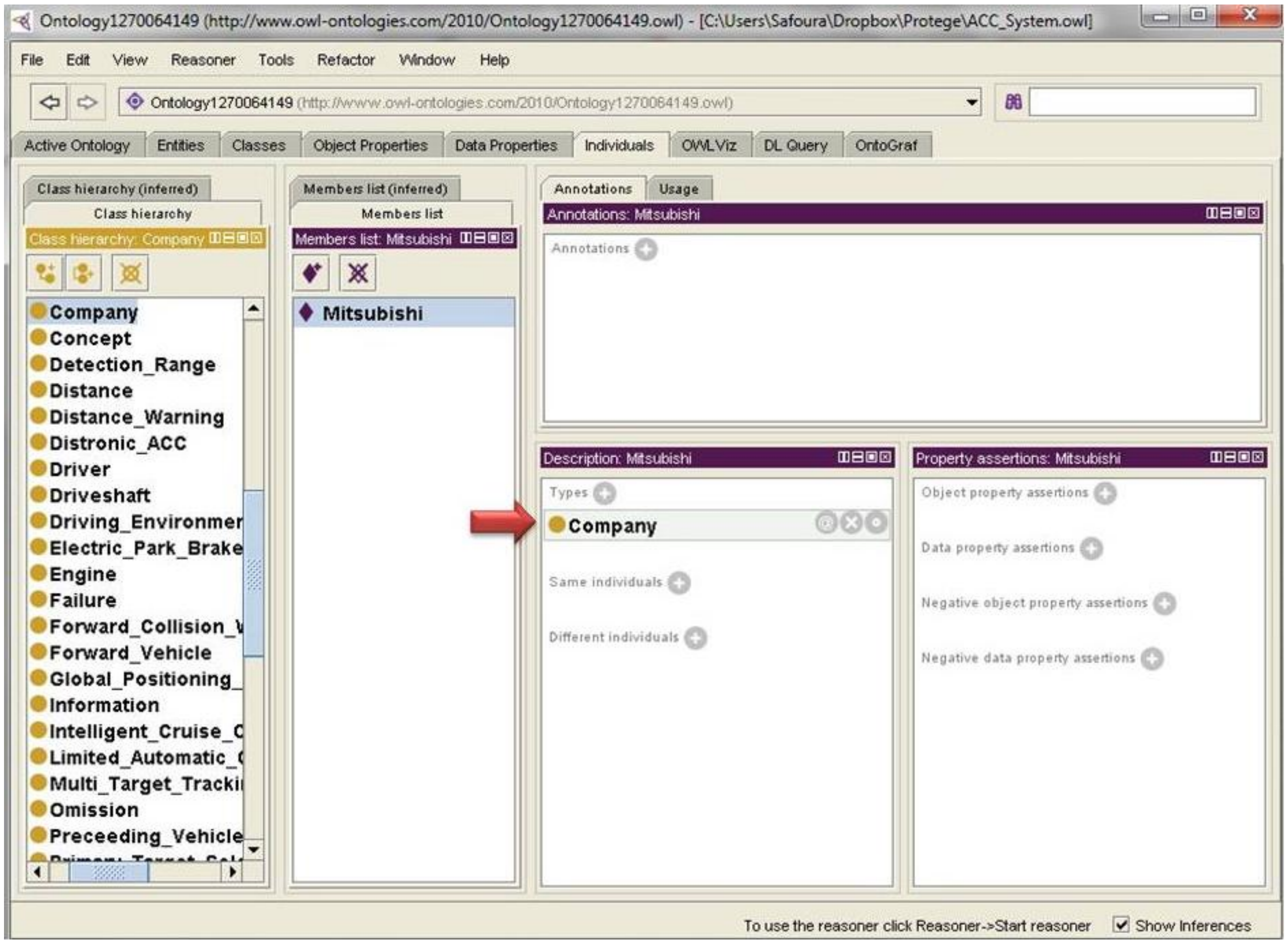
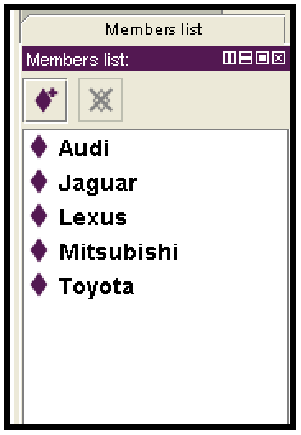

Ontology1270064149 (http://www.owl-ontologies.com/2010/Ontology1270064149.owl) - [C:\Users\Safoura\Dropbox\Protege\ACC_System.owl]
File Edit View Reasoner Tools Refactor Window Help
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query OntoGraf
Class hierarchy (inferred) Class hierarchy
Class hierarchy: Company
Company
Concept
Detection_Range
Distance
Distance_Warning
Distronic_ACC
Driver
Driveshaft
Driving_Environmer
Electric_Park_Brake
Engine
Failure
Forward_Collision_W
Forward_Vehicle
Global_Positioning_
Information
Intelligent_Cruise_C
Limited_Automatic_(
Multi_Target_Tracki
Omission
Preceeding_Vehicle
Members list (inferred) Members list
Members list: Mitsubishi
Mitsubishi
Annotations Usage
Annotations: Mitsubishi
Annotations
Description: Mitsubishi
Types
Company
Same individuals
Different individuals
Property assertions: Mitsubishi
Object property assertions
Data property assertions
Negative object property assertions
Negative data property assertions
To use the reasoner click Reasoner->Start reasoner
Show Inferences

Figure (7.2)

Repeat the above steps to create some more individuals that are members of the class Company. After adding individuals, the member list should be like **(Figure 7.3)**. Using ‘Delete individuals’ button next to ‘Add individuals’ button, you can delete previously created individuals.



- Audi
- Jaguar
- Lexus
- Mitsubishi
- Toyota

Figure (7.3)

Reference:

Protégé web site: http://Protégé.stanford.edu/