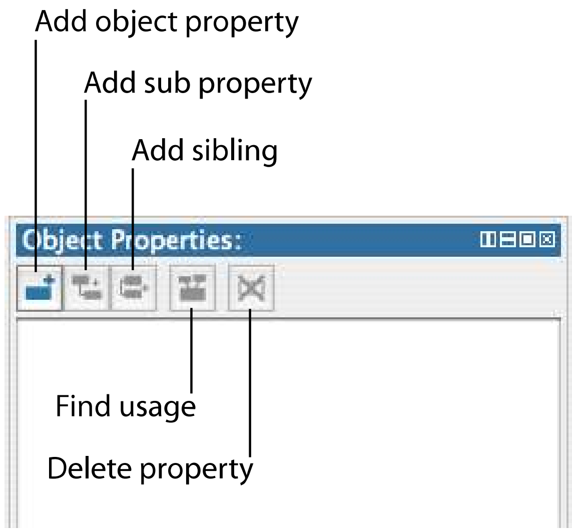

Figure 4.14: Property Creation Buttons — located on the Properties Tab above the property list/tree

Figure 4.15: Property Name Dialog

TIP

> Although there is no strict naming convention for properties, we recommend that property names start with a lower case letter, have no spaces and have the remaining words capitalised. We also recommend that properties are prefixed with the word 'has', or the word 'is', for example hasPart, isPartOf, hasManufacturer, isProducerOf. Not only does this convention help make the intent of the property clearer to humans, it is also taken advantage of by the 'English Prose Tooltip Generator'[a], which uses this naming convention where possible to generate more human readable expressions for class descriptions.
>
> [a]The English Prose Tooltip Generator displays the description of classes etc. in a more natural form of English, making is easy to understand a class description. The tooltips pop up when the mouse pointer is made to hover over a class description in the user interface.

Having added the hasIngredient property, we will now add two more properties — hasTopping, and hasBase. In OWL, properties may have sub properties, so that it is possible to form hierarchies of properties. Sub properties specialise their super properties (in the same way that subclasses specialise their superclasses). For example, the property hasMother might specialise the more general property of

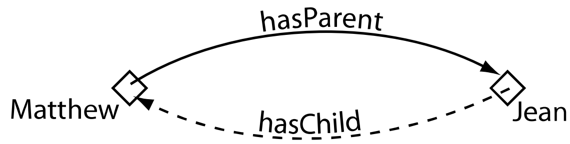

Figure 4.16: An Example Of An Inverse Property: hasParent has an inverse property that is hasChild

hasParent. In the case of our pizza ontology the properties hasTopping and hasBase should be created as sub properties of hasIngredient. If the hasTopping property (or the hasBase property) links two individuals this implies that the two individuals are related by the hasIngredient property.

Exercise 9: Create hasTopping **and** hasBase **as sub-properties of** hasIngredient

1. To create the hasTopping property as a sub property of the hasIngredient property, select the hasIngredient property in the property hierarchy on the '**Object Properties**' tab.
2. Press the '**Add subproperty**' button. A new object property will be created as a sub property of the hasIngredient property.
3. Name the new property to hasTopping.
4. Repeat the above steps but name the property hasBase.

Note that it is also possible to create sub properties of datatype properties. However, it is not possible to mix and match object properties and datatype properties with regards to sub properties. For example, it is not possible to create an object property that is the sub property of a datatype property and vice-versa.

4.5 Inverse Properties

Each object property may 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. For example, Figure 4.16 shows the property hasParent and its inverse property hasChild — if Matthew hasParent Jean, then because of the inverse property we can infer that Jean hasChild Matthew.

Inverse properties can be created/specified using the inverse property view shown in Figure 4.17. For

Figure 4.17: The Inv erse Property View

completeness we will specify inverse properties for our existing properties in the Pizza Ontology.

Exercise 10: Create some inverse properties

1. Use the '**Add object property**' button on the '**Object Properties**' tab to create a new Object property called isIngredientOf (this will become the inverse property of hasIngredient).
2. Press the '**Add**' icon ( ) next to '**Inverse properties**' button on the '**Property Description**' view shown in Figure 4.17. This will display a dialog from which properties may be selected. Select the hasIngredient property and press '**OK**'. The property hasIngredient should now be displayed in the '**Inverse Property**' view.
3. Select the hasBase property.
4. Press the '**Add**' icon ( ) next to '**Inverse properties**' on the '**Property Description**' view. Create a new property in this dialog called isBaseOf. Select this property and click '**OK**'. Notice that hasBase now has a inverse property assigned called isBaseOf. You can optionally place the new isBaseOf property as a sub-property of isIngredientOf (N.B This will get inferred later anyway when you use the reasoner).
5. Select the hasTopping property.
6. Press the '**Add**' icon ( ) next to '**Inverse properties**' on the '**Property Description**' view. Use the property dialog that pops up to create the property isToppingOf and press '**OK**'.

4.6 OWL Object Property Characteristics

OWL allows the meaning of properties to be enriched through the use of *property characteristics*. The following sections discuss the various characteristics that properties may have:

4.6.1 Functional Properties

If a property is functional, for a given individual, there can be at most one individual that is related to the individual via the property. Figure 4.18 shows an example of a functional property hasBirthMother — something can only have *one* birth mother. If we say that the individual Jean hasBirthMother Peggy and we also say that the individual Jean hasBirthMother Margaret[4], then because hasBirthMother is a functional property, we can infer that Peggy and Margaret must be the same individual. It should be noted however, that if Peggy and Margaret were explicitly stated to be two different individuals then the above statements would lead to an inconsistency.

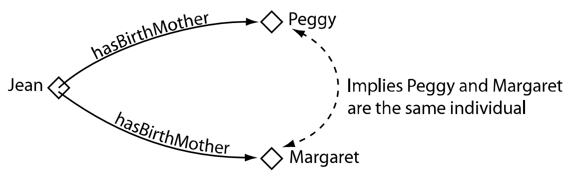

Figure 4.18: An Example Of A Functional Property: hasBirthMother

Vocabulary

Functional properties are also known as *single valued properties* and also *features*.

4.6.2 Inverse Functional Properties

If a property is inverse functional then it means that the *inverse* property is *functional*. For a given individual, there can be at most one individual related to that individual via the property. Figure 4.19 shows an example of an inverse functional property isBirthMotherOf. This is the inverse property of hasBirthMother — since hasBirthMother is functional, isBirthMotherOf is inverse functional. If we state that Peggy is the birth mother of Jean, and we also state that Margaret is the birth mother of Jean, then we can infer that Peggy and Margaret are the same individual.

4.6.3 Transitive Properties

If a property is transitive, and the property relates individual a to individual b, and also individual b to individual c, then we can infer that individual a is related to individual c via property P. For example, Figure 4.20 shows an example of the transitive property hasAncestor. If the individual Matthew has an ancestor that is Peter, and Peter has an ancestor that is William, then we can infer that Matthew has an ancestor that is William – this is indicated by the dashed line in Figure 4.20.

[4]The name Peggy is a diminutive form for the name Margaret

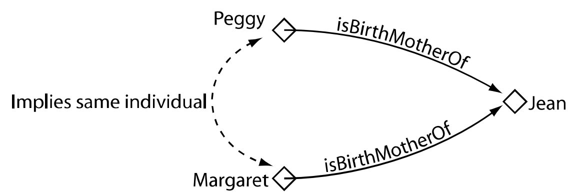

Figure 4.19: An Example Of An Inverse Functional Property: isBirthMotherOf

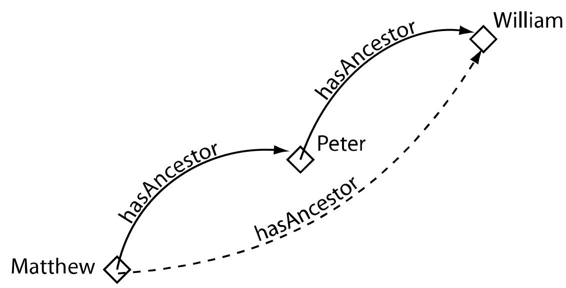

Figure 4.20: An Example Of A Transitive Property: hasAncestor

4.6.4 Symmetric Properties

If a property P is symmetric, and the property relates individual a to individual b then individual b is also related to individual a via property P. Figure 4.21 shows an example of a symmetric property. If the individual Matthew is related to the individual Gemma via the hasSibling property, then we can infer that Gemma must also be related to Matthew via the hasSibling property. In other words, if Matthew has a sibling that is Gemma, then Gemma must have a sibling that is Matthew. Put another way, the property is its own inverse property.

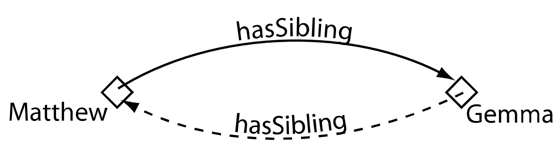

Figure 4.21: An Example Of A Symmetric Property: hasSibling

We want to make the hasIngredient property transitive, so that for example if a pizza topping has an ingredient, then the pizza itself also has that ingredient. To set the property characteristics of a property the property characteristics view shown in Figure 4.22 which is located in the lower right hand corner of

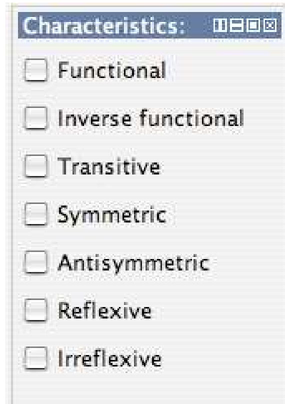

Figure 4.22: Property Characteristics Views

the properties tab is used.

Exercise 11: Make the hasIngredient **property transitive**

1. Select the hasIngredient property in the property hierarchy on the '**Object Properties**' tab.
2. Tick the '**Transitive**' tick box on the '**Property Characteristics View**'.
3. Select the isIngredientOf property, which is the inverse of hasIngredient. Ensure that the transitive tick box is ticked.

NOTE

If a property is transitive then its inverse property should also be transitive.[a]

[a] At the time of writing this must be done manually in Protégé 4. However, the reasoner *will* assume that if a property is transitive, its inverse property is also a transitive.

Note that if a property is transitive then it cannot be functional.[a]

[a] The reason for this is that transitive properties, by their nature, may form 'chains' of individuals. Making a transitive property functional would therefore not make sense.

We now want to say that our pizza can only have one base. There are numerous ways that this could be accomplished. However, to do this we will make the hasBase property *functional*, so that it may have

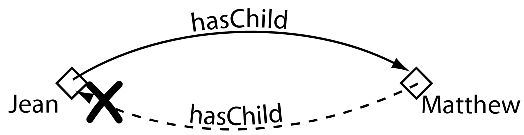

Figure 4.23: An example of the asymmetric property hasChildOf

only one value for a given individual.

Exercise 12: Make the hasBase **property functional**

1. Select the hasBase property.
2. Click the '**Functional**' tick box on the '**Property Characteristics View**' so that it is ticked.

NOTE

> If a datatype property is selected, the property characteristics view will be reduced so that only options for '**Allows multiple values**' and '**Inverse Functional**' will be displayed. This is because OWL-DL does not allow datatype properties to be transitive, symmetric or have inverse properties.

4.6.5 Asymmetric properties

If a property P is asymmetric, and the property relates individual a to individual b then individual b cannot be related to individual a via property P. Figure 4.23 shows an example of a asymmetric property. If the individual Robert is related to the individual David via the isChildOf property, then it can be inferred that David is not related to Robert via the isChildOf property. It is, however, reasonable to state that David could be related to another individual Bill via the isChildOf property. In other words, if Robert is a child of David, then David cannot be a child of Robert, but David can be a child of Bill.

4.6.6 Reflexive properties

A property P is said to be reflexive when the property must relate individual a to itself. In Figure 4.24 we can see an example of this: using the property knows, an individual George must have a relationship to itself using the property knows. In other words, George must know herself. However, in addition, it is possible for George to know other people; therefore the individual George can have a relationship with individual Simon along the property knows.

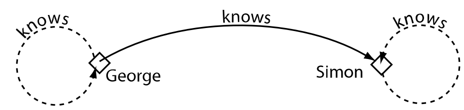

Figure 4.24: An example of a Reflexive Property: knows

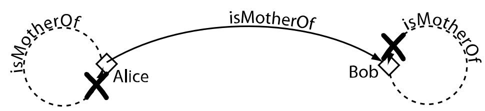

Figure 4.25: An example of an Irreflexive Property: isMotherOf

4.6.7 Irreflexive properties

If a property P is *irreflexive*, it can be described as a property that relates an individual a to individualb, where individual a and individualb are not the same. An example of this would be the property motherOf: an individual Alice can be related to individual Bob along the property motherOf, but Alice cannot be motherOf herself (Figure 4.25).

4.7 Property Domains and Ranges

Properties may have a *domain* and a *range* specified. Properties link individuals from the *domain* to individuals from the *range*. For example, in our pizza ontology, the property hasTopping would probably link individuals belonging to the class Pizza to individuals belonging to the class of PizzaTopping. In this case the *domain* of the hasTopping property is Pizza and the *range* is PizzaTopping — this is depicted in Figure 4.26.

Property Domains And Ranges In OWL — It is important to realise that in OWL domains and ranges should *not* be viewed as constraints to be checked. They are used as 'axioms' in reasoning. For example if the property hasTopping has the domain set as Pizza and we then applied the hasTopping property to IceCream (individuals that are members of the class IceCream), this would generally not result in an error. It would be used to infer that the class IceCream must be a subclass of Pizza! [a].

[a] An error will only be generated (by a reasoner) if Pizza is disjoint to IceCream

We now want to specify that the hasTopping property has a *range* of PizzaTopping. To do this the range

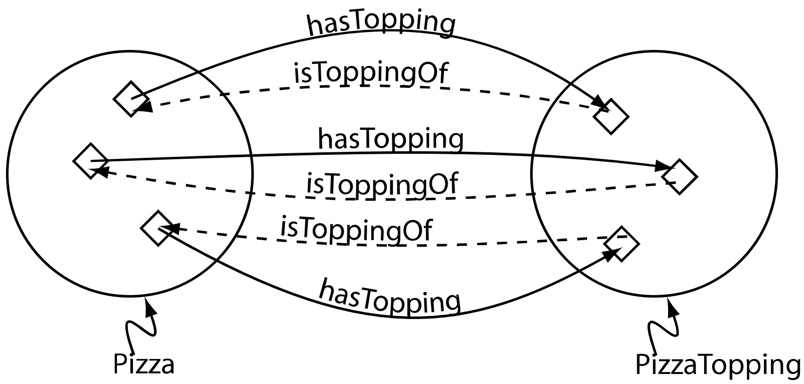

Figure 4.26: The domain and range for the hasTopping property and its inverse property isToppingOf. The domain of hasTopping is Pizza the range of hasTopping is PizzaTopping — the domain and range for isToppingOf are the domain and range for hasTopping swapped over

Figure 4.27: Property Range View (For Object Properties)

view shown in Figure 4.27 is used.

Exercise 13: Specify the range of hasTopping

1. Make sure that the hasTopping property is selected in the property hierarchy on the '**Object Properties**' tab.
2. Press the '**Add**' icon ( ) next to '**Ranges**' on the '**Property Description**' view (Figure 4.27). A dialog will appear that allows a class to be selected from the ontology class hierarchy.
3. Select PizzaTopping and press the '**OK**' button. PizzaTopping should now be displayed in the range list.

Figure 4.28: Property Domain View

NOTE

It is possible to specify multiple classes as the range for a property. If multiple classes are specified in Protégé 4 the range of the property is interpreted to be the *intersection* of the classes. For example, if the range of a property has the classes Man and Woman listed in the range view, the range of the property will be interpreted as Man *intersection* Woman.

To specify the domain of a property the domain view shown in Figure 4.28 is used.

Exercise 14: Specify Pizza **as the domain of the** hasTopping **property**

1. Make sure that the hasTopping property is selected in the property hierarchy on the '**Object Properties**' tab.
2. Press the '**Add**' icon ( ) next to '**Domains**' on the '**Propert Description**' view . A dialog will appear that allows a class to be selected from the ontology class hierarchy.
3. Select Pizza and press the OK button. Pizza should now be displayed in the domain list.

MEANING

This means that individuals that are used 'on the left hand side' of the hasTopping property will be inferred to be members of the class Pizza. Any individuals that are used 'on the right hand side' of the hasTopping property will be inferred to be members of the class PizzaTopping. For example, if we have individuals a and b and an assertion of the form a hasTopping b then it will be inferred that a is a member of the class Pizza and that b is a member of the class PizzaTopping[a].

[a] This will be the case even if a has not been asserted to be a member of the class Pizza and/or b has not been asserted to be a member of the class PizzaTopping.

NOTE

> Take a look at the isToppingOf property, which is the inverse property of hasTopping. Notice that Protégé has automatically filled in domain and range of the isToppingOf property because the domain and range of the inverse property were specified. The range of isToppingOf is the domain of the inverse property hasTopping, and the domain of isToppingOf is the range of the inverse property hasTopping. This is depicted in Figure 4.26.

Exercise 15: Specify the domain and range for the hasBase **property and its inverse property** isBaseOf

---

1. Select the hasBase property.
2. Specify the domain of the hasBase property as Pizza.
3. Specify the range of the hasBase property as PizzaBase.
4. Select the isBaseOf property. Notice that the domain of isBaseOf is the range of the inverse property hasBase and that the range of isBaseOf is the domain of the inverse property hasBase.
5. Make the domain of the isBaseOf property PizzaBase.
6. Make the range of the isBaseOf property Pizza.

TIP

> In the previous steps we have ensured that the domains and ranges for properties are also set up for inverse properties in a correct manner. In general, domain for a property is the range for its inverse, and the range for a property is the domain for its inverse — Figure 4.26 illustrates this for the hasTopping and isToppingOf.

> Although we have specified the domains and ranges of various properties for the purposes of this tutorial, we generally advise *against* doing this. The fact that domain and range conditions do not behave as constraints and the fact that they can cause 'unexpected' classification results can lead problems and unexpected side effects. These problems and side effects can be particularly difficult to track down in a large ontology.

4.8 Describing And Defining Classes

Having created some properties we can now use these properties to describe and define our Pizza Ontology classes.

4.8.1 Property Restrictions

Recall that in OWL, properties describe binary relationships. Datatype properties describe relationships between individuals and data values. Object properties describe relationships between two individuals. For example, in Figure 3.2 the individual Matthew is related to the individual Gemma via the hasSibling property. Now consider all of the individuals that have a hasSibling relationship to some other individual. We can think of these individuals as belonging *the class of individuals* that have some hasSibling relationship. The key idea is that a class of individuals is described or defined by the relationships that these individuals participate in. In OWL we can define such classes by using *restrictions*.

Vocabulary

> 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.

Restriction Examples

Let's take a look at some examples to help clarify the kinds of classes of individuals that we might want to describe based on their properties.

- The class of individuals that have at least one hasSibling relationship.
- The class of individuals that have at least one hasSibling relationship to members of Man – i.e. things that have at least one sibling that is a man.
- The class of individuals that only have hasSibling relationships to individuals that are Women – i.e. things that only have siblings that are women (sisters).
- The class of individuals that have more than three hasSibling relationships.
- The class of individuals that have at least one hasTopping relationship to individuals that are members of MozzarellaTopping – i.e. the class of things that have at least one kind of mozzarella topping.
- The class of individuals that only have hasTopping relationships to members of VegetableTopping – i.e. the class of individuals that only have toppings that are vegetable toppings.

In OWL we can describe all of the above classes of individuals using *restrictions*. OWL restrictions in OWL fall into three main categories:

- Quantifier Restrictions
- Cardinality Restrictions
- hasValue Restrictions.

We will initially use quantifier restrictions, which can be further categorised into *existential* restrictions and *universal* restrictions. Both types of restrictions will be illustrated with examples throughout the tutorial.

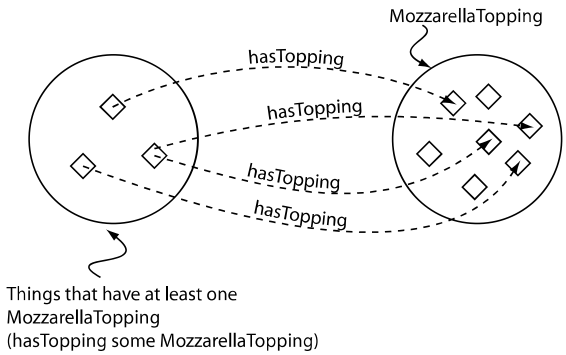

Figure 4.29: The Restriction hasTopping some Mozzarella. This restriction describes the class of individuals that have *at least one* topping that is Mozzarella

Existential and Universal Restrictions

- 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, "the class of individuals that have *at least one* (some) hasTopping relationship to members of MozzarellaTopping". In Protégé 4 the keyword '**some**' is used to denote existential restrictions.[5].
- 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 hasTopping relationships to members of VegetableTopping". In Protégé 4 the keyword '**only**' is used. [6].

Let's take a closer look at the example of an existential restriction. The restriction hasTopping some MozzarellaTopping is an existential restriction (as indicated by the some keyword), which acts along the hasTopping property, and has a *filler* MozzarellaTopping. This restriction describes *the class* of individuals that have *at least one* hasTopping relationship to an individual that is a member of the class MozzarellaTopping. This restriction is depicted in Figure 4.29 — The diamonds in the Figure represent individuals. As can be seen from Figure 4.29, the restriction is a class which contains the individuals that satisfy the restriction.

MEANING

> A restriction describes an *anonymous class* (an unnamed class). The anonymous class contains all of the individuals that satisfy the restriction – i.e. all of the individuals that have the relationships required to be a member of the class.

The restrictions for a class are displayed and edited using the '**Class Description View**' shown in Figure 4.30. The '**Class Description View**' is the 'heart of' the '**Classes**' tab in protege, and holds virtually all of the information used to describe a class. At first glance, the '**Class Description View**' may seem complicated, however, it will become apparent that it is an incredibly powerful way of describing and defining classes.

[5] Existential restrictions may be denoted by the *existential quantifier* (∃). They are also knows as 'someValuesFrom' restrictions in OWL speak.

[6] Universal restrictions may be denoted by the *universal quantifier* (∀), which can be read as *only*. They are also known as 'allValuesFrom' restrictions in OWL speak.

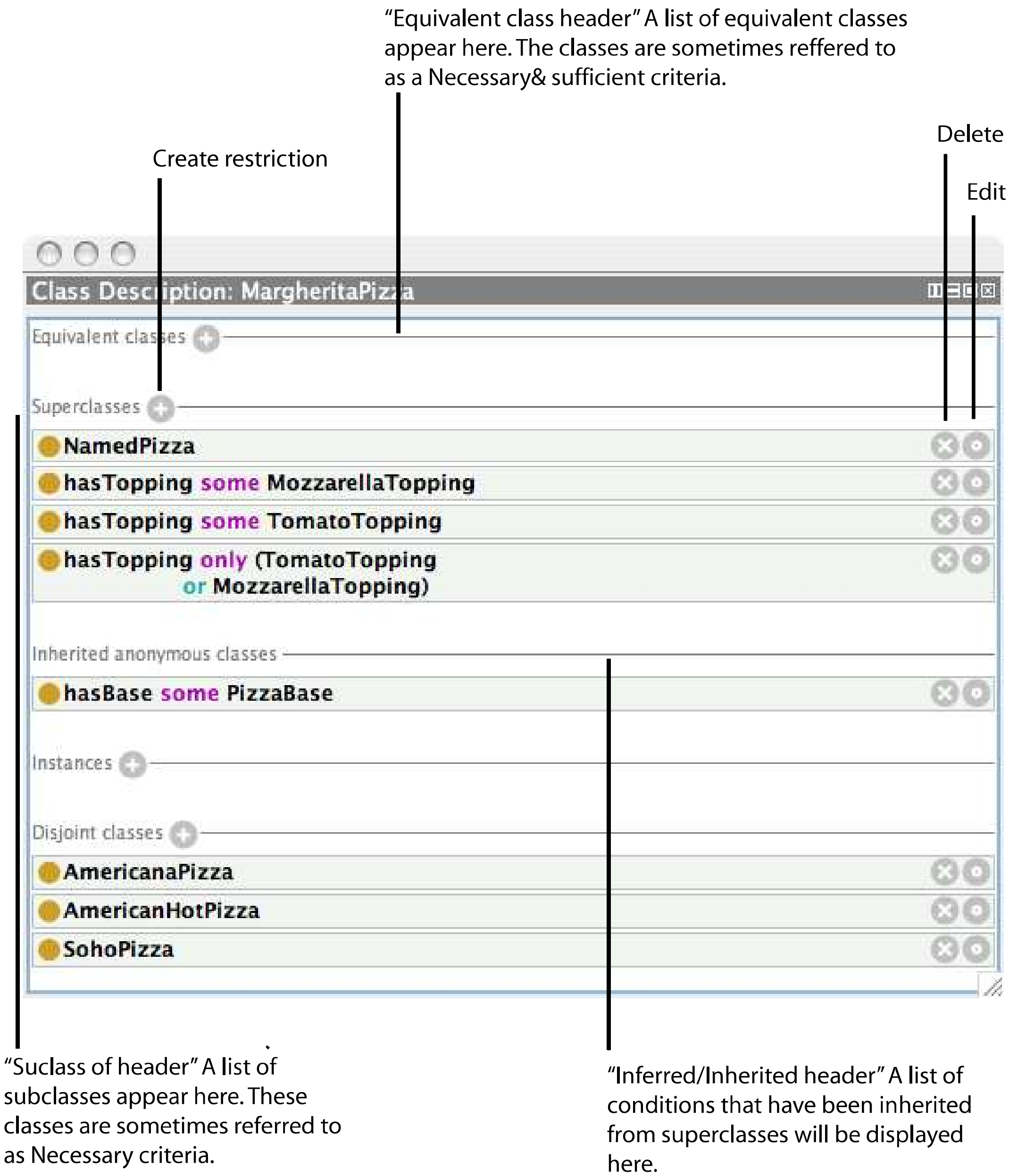

Figure 4.30: The Class Description View

Restrictions are used in OWL class descriptions to specify anonymous superclasses of the class being described.

4.8.2 Existential Restrictions

Existential restrictions are by far the most common type of restrictions in OWL ontologies. An existential restriction describes a class of individuals that have *at least one* (some) relationship along a specified property to an individual that is a member of a specified class. For example, hasBase some PizzaBase describes all of the individuals that have *at least one* relationship along the hasBase property to an individual that is a member of the class PizzaBase — in more natural English, all of the individuals that have at least one pizza base.

Vocabulary

> Existential restrictions are also known as *Some Restrictions*, or as *some values from* restrictions.

TIP

> Other tools, papers and presentations might write the restriction hasBase some PizzaBase as $\exists$ hasBase PizzaBase — this alternative notation is known as DL Syntax (Description Logics Syntax), which is a more formal syntax.

Exercise 16: Add a restriction to Pizza **that specifies a** Pizza **must have a** PizzaBase

1. Select Pizza from the class hierarchy on the '**Classes**' tab.
2. Select the '**Add**' icon ( ) next to '**Superclasses**' header in the '**Class Description View**' shown in Figure 4.31 in order to create a necessary condition.
3. Press the '**Add Class**' button shown in Figure 4.31. This will open a text box in the Class Description view where we can enter our restrictions as shown in Figure 4.32

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

Figure 4.31: Creating a Necessary Restriction

Exercise 17: Add a restriction to Pizza **that specifies a** Pizza **must have a** PizzaBase **(Continued...)**

---

1. You can either drag and drop hasBase from the property list into the create restriction text box, or type it in.
2. Now add the type or restriction, we will use an existential restriction so type '**some**'.
3. Specify that the filler is PizzaBase — to do this either: type `PizzaBase` into the filler edit box, or drag and drop `PizzaBase` into the text box as show in Figure 4.32
4. Press '**Enter**' to create the restriction and close the create restriction text box. If all information was entered correctly the dialog will close and the restriction will be displayed in the '**Class Description View**'. If there were errors they will be underlined in red in the text box, o 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.

TIP

A very useful feature of the expression builder is the ability to 'auto complete' class names, property names and individual names. Auto completion is activated by pressing '**alt tab**' or '**Ctrl-Space**' on the keyboard. In the above example if we had typed `Pi` into the expression editor and pressed the tab key, the choices to complete the word Pi would have poped up in a list as shown in Figure 4.32. The up and down arrow keys could then have been used to select PizzaBase and pressing the Enter key would complete the word for us.

The class description view should now look similar to the picture shown in Figure 4.33.

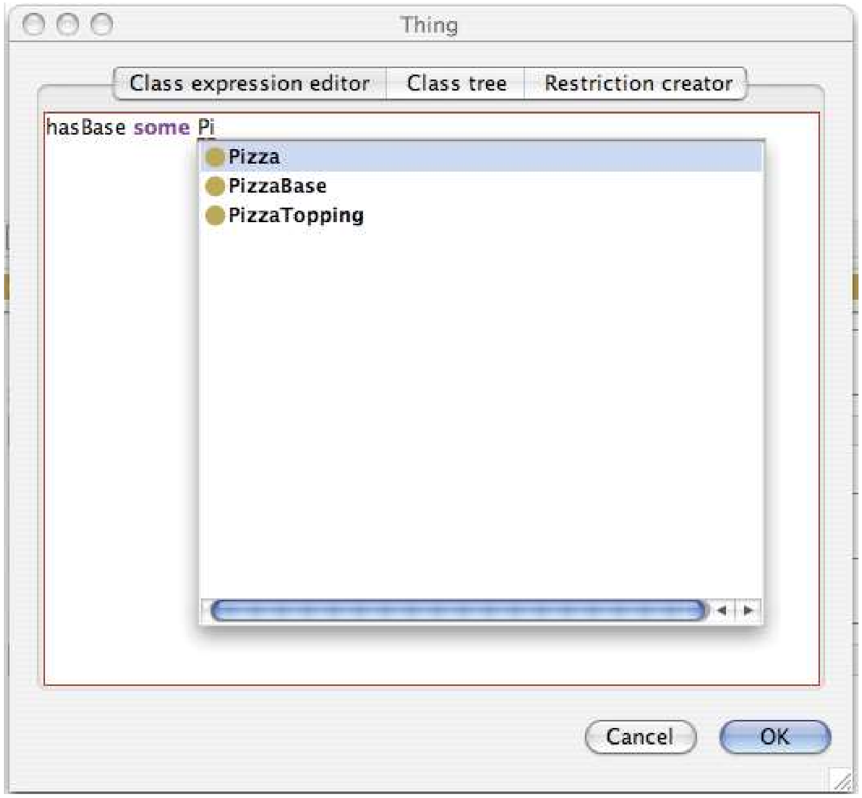

Figure 4.32: Creating a restriction in the text box, with auto-complete

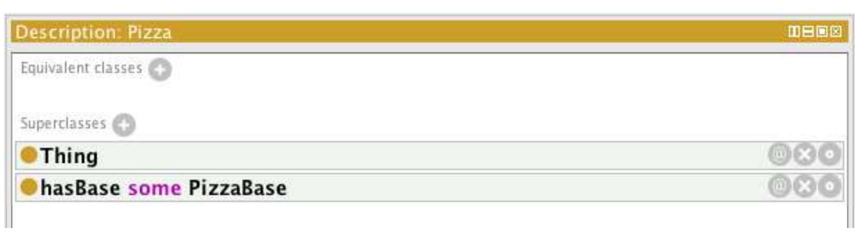

Figure 4.33: class description view: Description of a Pizza

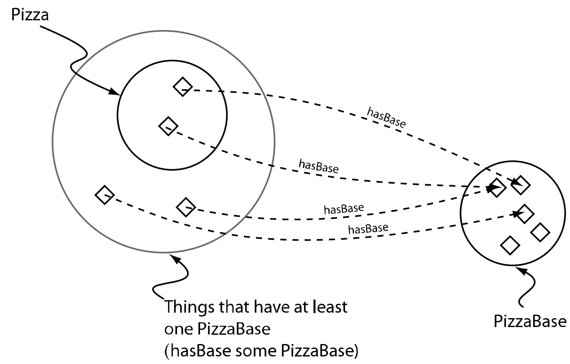

Figure 4.34: A Schematic Description of a Pizza — In order for something to be a Pizza it is *necessary* for it to have a (*at least one*) PizzaBase — A Pizza is a *subclass* of the things that have *at least one* PizzaBase

MEANING

> We have described the class Pizza to be a subclass of Thing and a subclass of the things that have a base which is some kind of PizzaBase.
> Notice that these are *necessary* conditions — if something is a Pizza it is *necessary* for it to be a member of the class Thing (in OWL, everything is a member of the class Thing) and necessary for it to have a kind of PizzaBase.
> More formally, for something to be a Pizza it is *necessary* for it to be in a relationship with an individual that is a member of the class PizzaBase via the property hasBase — This is depicted in Figure 4.34.

MEANING

> When restrictions are used to describe classes, they actually specify anonymous superclasses of the class being described. For example, we could say that MargheritaPizza is a subclass of, amongst other things, Pizza and also a subclass of the things that have *at least one* topping that is MozzarellaTopping.

Creating Some Different Kinds Of Pizzas

It's now time to add some different kinds of pizzas to our ontology. We will start off by adding a 'MargheritaPizza', which is a pizza that has toppings of mozzarella and tomato. In order to keep our ontology tidy, we will group our different pizzas under the class 'NamedPizza':

Exercise 18: Create a subclass of Pizza **called** NamedPizza**, and a subclass of** NamedPizza **called** MargheritaPizza

---

1. Select the class Pizza from the class hierarchy on the '**Classes**' tab.
2. Press the '**Add**' icon ( ) to create a new subclass of Pizza, and name it NamedPizza.
3. Create a new subclass of NamedPizza, and name it MargheritaPizza.
4. Add a comment to the class MargheritaPizza using the '**Annotations**' view that is located next to the class hierarchy view: `A pizza that only has Mozarella and Tomato toppings` – it's always a good idea to document classes, properties etc. during ontology editing sessions in order to communicate intentions to other ontology builders.

Having created the class MargheritaPizza we now need to specify the toppings that it has. To do this we will add two restrictions to say that a MargheritaPizza has the toppings MozzarellaTopping and TomatoTopping.

Exercise 19: Create an existential (some) restriction on MargheritaPizza **that acts along the property** hasTopping **with a filler of** MozzarellaTopping **to specify that a** MargheritaPizza **has at least one** MozzarellaTopping

---

1. Make sure that MargheritaPizza is selected in the class hierarchy.
2. Use the '**Add**' icon ( ) on the '**Superclasses**' section of the '**Class Description view**' (Figure 4.30) to open a text box.
3. Type hasTopping as the property to be restricted in the text box.
4. Type '**some**' to create the existential restriction.
5. Type the class MozzarellaTopping as the filler for the restriction — remember that this can be achieved by typing the class name `MozzarellaTopping` into the filler edit box, or by using drag and drop from the class hierarchy.
6. Press '**Enter**' to create the restriction — if there are any errors, the restriction will not be created, and the error will be highlighted in red.

Figure 4.35: The Class Description View Showing A Description Of A MargheritaPizza

Now specify that MargheritaPizzas also have TomatoTopping.

Exercise 20: Create a existential restriction (some) on MargheritaPizza **that acts along the property** hasTopping **with a filler of** TomatoTopping **to specify that a** MargheritaPizza **has *at least one*** TomatoTopping

---

1. Ensure that MargheritaPizza is selected in the class hierarchy.
2. Use the '**Add**' icon ( ) on the '**Superclasses**' section of the '**Class Description View**' (Figure 4.30) to display open the text box.
3. Type hasTopping as the property to be restricted.
4. Type '**some**' to create the existential restriction.
5. Type the class TomatoTopping as the filler for the restriction.
6. Click '**Enter**' to create restriction dialog to create the restriction.

The '**Class Description View**' should now look similar to the picture shown in Figure 4.35.

MEANING

We have added restrictions to MargeritaPizza to say that a MargheritaPizza is a NamedPizza that has at least one kind of MozzarellaTopping and at least one kind of TomatoTopping.
More formally (reading the class description view line by line), if something is a member of the class MargheritaPizza it is *necessary* for it to be a member of the class NamedPizza *and* it is *necessary* for it to be a member of the anonymous class of things that are linked to at least one member of the class MozzarellaTopping via the property hasTopping, *and* it is *necessary* for it to be a member of the anonymous class of things that are linked to at least one member of the class TomatoTopping via the property hasTopping.

Now create the class to represent an Americana Pizza, which has toppings of pepperoni, mozzarella

Figure 4.36: The Class Description View displaying the description for AmericanaPizza

and tomato. Because the class AmericanaPizza is very similar to the class MargheritaPizza (i.e. an Americana pizza is almost the same as a Margherita pizza but with an extra topping of pepperoni) we will make a *clone* of the MargheritaPizza class and then add an extra restriction to say that it has a topping of pepperoni.

Exercise 21: Create AmericanaPizza **by cloning and modifying the description of** MargheritaPizza

1. Select the class MargheritaPizza in the class hierarchy on the Classes tab.
2. Select '"Duplicate selected class from the '**Edit**' menu. A dialog will appear for you to name the new class, this will be created a with exactly the same conditions (restrictions etc.) as AmericanaPizza.
3. Ensuring that AmericanaPizza is still selected, select the '**Add**' icon ( ) next to the '**Superclasses**' header in the class description view, as we want to add a new restriction to the necessary conditions for AmericanaPizza.
4. Type the property hasTopping as the property to be restricted.
5. Type '**some**' to create the existential restriction.
6. Specify the restriction filler as the class PepperoniTopping by either typing `PepperoniTopping` into the text box, or by using drag and drop from the class hierarchy.
7. Press OK to create the restriction.