# CARESSES ONTOLOGY

## A Practical Guide for Building and Modifying the CARESSES Ontology

**Carmine Tommaso Recchiuto**
Universitá di Genova

**Antonio Sgorbissa**
Universitá di Genova

# CONTENTS

# INTRODUCTION

This tutorial explains how the hierarchy of the *CARESSES Cultural Knowledge Base (CKB)* is organized, and how the robot is able to talk with a person about concepts that are represented in the CKB. In CARESSES, conversations between the robot and a person have three main purposes:

- Entertain the person by talking about topics she/he is familiar with
- Offer to perform activities to help or entertain the person
- Acquire new knowledge about the person's attitudes, preferences, habits, values, and beliefs. This ultimately helps the robot to i) talk about topics the person is familiar with, and ii) offer to perform appropriate activities.

*Remark 1.* This part of the conversation module does not directly address explicit person's requests (goals), but only general chitchatting. In other words, if the person directly asks the robot to perform some actions, this request will directly generate a goal for the planner and the robot will execute the corresponding action. Otherwise, the person's sentence will be taken as input by the "chitchatting" module, trying to achieve the three main purposes listed above.

This course includes a number of exercises. When doing the exercises, keep in mind how the basic elements of standard ontologies

- Classes
- Instances
- Chitchatting
- Object properties
- Data properties

are used in the CARESSES CKB .

These basic elements are described in Chapter 1. Chapters 2-5 describe additional elements of the Ontology.

To create and modify OWL ontologies, this guide refers to Protégé 5. Class, property and individual names are written in an Arial font like this.

Attached to this tutorial you will find a .jar file that you can use to chat with the virtual robot using the Ontology structure. To run the software, open a shell, move to the Tutorial directory and digit:

java -jar CKB_tutorial.jar [optional argument:culture (Indian is the default option)]

The software will take as input an OWL file named CKB.owl that should be in the same directory.

For each of the exercises (except for the Exercise 1.1, that does not require to create a new ontology), you will find a corresponding OWL file with the "solution". All these files are in the folder Solutions. The folder Ontologies contains the ontologies that are used as starting point for the exercises.
Finally, video tutorial generically related to this guide, and describing more specifically all the exercises will be soon available.

**CHAPTER 1**

# THE STRUCTURE OF THE CARESSES ONTOLOGY

In this chapter we will analyze the main components of the Ontology, focusing specifically on the CARESSES structure.

## 1.1 Classes

In the CARESSES ontology, classes are used to represent information that is culture-independent: therefore, this part of the ontology is similar to any other ontology you may be familiar with (e.g., the Pizza ontology[1]). The only difference is that all of the CARESSES classes are derived from a superclass called

Topic

By deriving subclasses from Topic we represent all of the concepts that may be relevant in the application domain (in principle, including all of the cultures in the world; more realistically, including the cultures we want to represent in the system). According to this rationale, in the same ontology we can have classes that represent Christian and Hindu holidays, French and Chinese food, Japanese and Indian clothes, etc. For example, the Diwali Festival of Lights, one of the most popular festivals among Hindu Indians, is also found in the ontology that the robot uses when interacting with a Japanese person, even if it is unlikely (but not impossible!) that a Japanese person is familiar with this festival. Since we want to avoid stereotypical representations, the system is able to talk with the person about virtually anything, even about topics that the person is unlikely to be familiar with.

As usual, classes are properly organized in a hierarchy to take into account that holidays and types of food are different concepts, and are therefore represented in different places of the hierarchy.

[1] https://protege.stanford.edu/ontologies/pizza/pizza.owl

*Remark 1.* In the CARESSES CKB, classes are written in lowercase letters, without spaces. Words are capitalized if the class name is composed of more than one word (e.g., Habit, WatchingMovies).

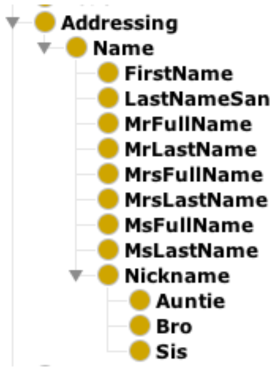



Figure 1.1: The hierarchy of classes that is necessary for the robot to talk with a person about different forms of address. The class Addressing has subclasses FirstName, LastNameSan, Nickname, and so on.

## 1.2 Instances

In the CARESSES ontology, instances may be of two kinds

- culture-specific instances
- person-specific instances.

Here CARESSES makes a distinction that is not standard in other ontologies.

### 1.2.1 Culture-specific instances and Person-specific instances

Culture-specific instances describe the knowledge, attitudes, preferences, habits, values, beliefs, etc., that are generally expected to characterize a cultural group. Person-specific instances describe the knowledge, attitudes, preferences, habits, values, beliefs, etc. that belong to the specific person the robot is interacting with. Once again, this distinction is drawn in order to avoid stereotypes: although we can make some reasonable assumptions about the habits of a person if we know the cultural group she belongs to, these assumptions need to be interpreted in a probabilistic sense. For instance, we cannot make any strong assumption that an Italian person loves soccer - although there is a good chance that this is true - or that a Japanese person has Miso soup for breakfast - especially considering that having a Western-style breakfast is increasingly popular in Japan.

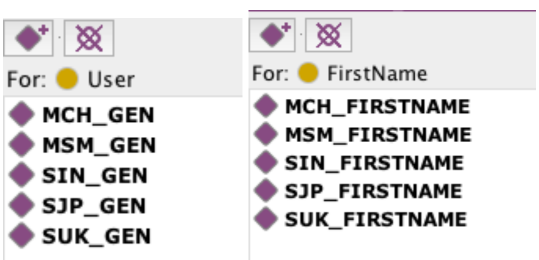



Figure 1.2: Instances of the classes User and FirstName, which represent the generic Indian, English and Japanese user and the attitude of a generic Indian, English, Japanese user towards being addressed by his/her first name (we expect that different cultural

groups may like or not like this possibility). Also shown are the instances MSM_FIRSTNAME and MCH_FIRSTNAME, which describe - respectively - what Mrs Smith and Mrs Chaterjee think about being addressed by their first name; this information may be acquired by the robot through interaction.

*Remark 1.* In the CARESSES CKB, instances are written in capital letters, without spaces. Each instance has a prefix that distinguishes different culture-specific instances (e.g., SIN for the Indian culture, SJP for the Japanese culture) and person-specific instances (e.g., MMS for Mrs Margaret Smith). Also, the first part of the instance's name is identical to the class it belongs to, whereas the remaining part of the name depends on factors that will be explained further on. Therefore, SIN_WATCHINGMOVIES is the culture-specific instance of the class WatchingMovies for the Indian culture, whereas MMS_WATCHINGMOVIES is the person-specific instance of the same class for Mrs Margaret Smith.

*Remark 2.* Culture-specific instances and Person-specific instances should always be linked by the Object Property hasSpecific.

*Remark 3.* A very important culture-specific instance exists which represents a generic person belonging to a given cultural group: generic Indian, English, Japanese persons are referred to, respectively, as SIN_GEN, SEN_GEN, and SJP_ GEN.

### 1.3 Chitchatting

As mentioned earlier, the main purposes of the CARESSES CKB are (i) to talk with the person about topics he/she is familiar with and (ii) to offer to achieve goals and/or perform activities to help or entertain the person (this will be not discussed in this Section, please see Paragraph 3.1.1 and 3.1.2). However, in order to meet the person's expectations, (iii) the system must acquire knowledge about their attitudes, preferences, habits, values, beliefs, and so on. In the simplest case, we can imagine that the system starts from an initial situation in which it has knowledge about the cultural group the person belongs to, but no specific knowledge about the person herself; in other words, only culture-specific instances are initially present. The purpose of chitchatting is to acquire knowledge about the person (i.e., person-specific instances in the CKB) starting from *a priori* assumptions about the cultural group (i.e., culture-specific instances in the CKB). While doing this, the robot shall always be entertaining and helpful for the person. For instance, let us suppose that it is breakfast time: the robot may decide that it is reasonable to start talking about breakfast options, but which breakfast options? Since the robot knows nothing about the person's preferences, it will start to explore the most probable options: if the person is Italian, the robot will suggest having coffee and biscuits, but if it discovers that these are not the person's preferred options, it will revise its suggestions.

*Remark 1.* As explained further on, the robot may have some initial knowledge about the person. Initial knowledge can be encoded into the system during the setup phase, creating proper person-specific instances along with culture-specific instances. This will be shown in one of the exercises below.

### 1.4 Object Properties

As is commonly the case in Ontologies, Object properties may be defined in the CARESSES CKB and represent links between different classes; likewise, instances of Object properties represent links between different instances in the ontology. The predefined Object Properties in the CARESSES ontology are:

- hasTopic
- hasCorrelation
- hasCondition

The most important Object Property is hasTopic, from which almost all of the Object Properties in the CARESSES ontology are derived. hasTopic is described below, while hasCorrelation and hasCondition will be discussed later.

### 1.4.1 hasTopic

hasTopic is an Object Property that describes the fact that a certain class (derived from Topic) may be a conversation topic for the robot. The Object Property hasTopic is used to link the instance that represents a generic person belonging to a cultural group (e.g., SIN_GEN, SEN_GEN, SJP_GEN) to all of the instances that the robot can talk about. For instance, linking the generic Indian person SIN_GEN to SIN_WATCHINGMOVIES, enables the robot to talk about the person's preferences concerning watching movies; linking the prototypical English person SEN_GEN to SEN_COFFEE, enables the robot to talk about the person's preferences about coffee.

*Remark 1.* Culture-specific instances corresponding to different cultures are never connected to each other: for instance, SIN_GEN is only connected to instances with prefix SIN, and SEN_GEN only to instances with prefix SEN.

*Remark 2.* The Object Property hasTopic is only used to connect culture-specific instances, i.e., instances that represent the generic person belonging to a cultural group and her/his attitudes, preferences, habits, values, beliefs, etc. Person-specific instances are not connected this way: their role and how they represent person-specific attitudes, preferences, habits, values, and beliefs will be clarified in one of the exercises below.

*Remark 3.* To take advantage of the hierarchical structure of the ontology, the hasTopic property is never used to connect culture-specific instances directly to each other in the CARESSES CKB. Conversely, a hierarchy of sub-properties is derived from hasTopic, each with well-defined semantics. For example, SIN_GEN is likely to be connected to SIN_WATCHINGMOVIES by Object Property hasHobby and SEN_GEN is likely to be connected to SEN_COFFEE by Object Property hasBeverage, where both hasHobby and hasBeverage are derived from hasTopic.

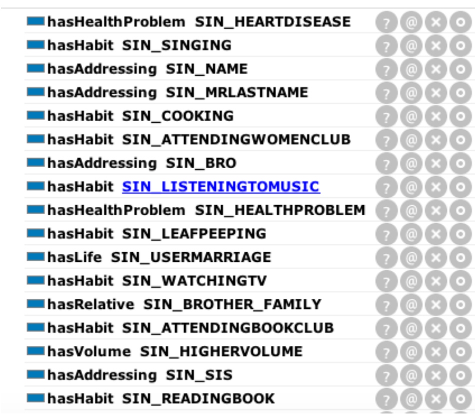



Figure 1.3: A generic Indian person and some Object properties (all of which are sub-properties of hasTopic

## 1.5 Data Properties

All of the classes of the CARESSES CKB that are relevant to conversations between the robot and the person have data properties that enable the system to talk about the corresponding concept, while - at the same time - acquiring new knowledge about the person's attitudes, preferences, habits, values, beliefs, etc. The most important data properties are the following (other data properties will be introduced in the exercises below):

- hasLikeliness
- hasQuestion
- hasPositiveSentence
- hasNegativeSentence
- hasPositiveAndWait

### 1.5.1 hasLikeliness

The link between culture-specific instances and person-specific instances is provided by a data property called hasLikeliness. All of the culture-specific instances in the ontology must have a likeliness value ranging between 0 and 1 (i.e., interpreted as a probability). If a culture-specific instance has a high likeliness value, then it is very probable that a person belonging to that cultural group has a positive attitude towards the corresponding concept. For instance, even if we cannot be sure that an Italian person loves soccer, it is definitely more probable that he or she likes soccer rather than badminton, which is not very popular in Italy. This can be represented by connecting the culture-specific instance SIT_GEN (representing the prototypical Italian person) to the culture-specific instance SIT_SOCCER (representing soccer) through the property hasSport (which is derived from hasTopic), and by assigning a high likeliness value to the instance SIT_SOCCER. For instance, we might decide that SIT_SOCCER has value 0.9 for the Data Property hasLikeliness.

As mentioned earlier, we may wish to represent the fact that an Italian person may also like badminton, even if this is less probable: therefore, the robot must also be able to explore this possibility. To achieve this, it is sufficient to create a culture-specific in- stance SIT_BADMINTON (representing badminton) and to connect SIT_GEN with SIT _BADMINTON. Presumably, we will decide that SIT_BADMINTON has a lower value than SIT_SOCCER for the Data Property hasLikeliness.

Similarly, we may want to consider that an Indian person may or not may like soccer and badminton: to do so, we will connect SIN_GEN with SIN_SOCCER and SIN_BADMINTON through the Object Property hasSport, just as we did for the prototypical Italian person, and we will set likeliness values accordingly. At the end, we obtain that the instances SIT_GEN and SIN_GEN are connected to all instances describing sports, but the likeliness value is different for each sport. It may be argued that knowing the probability for each sport is not very easy: how can we establish the probability that an Indian person likes Sumo? All instances, for which the information is not easily found, may be assigned a low uniform probability value.

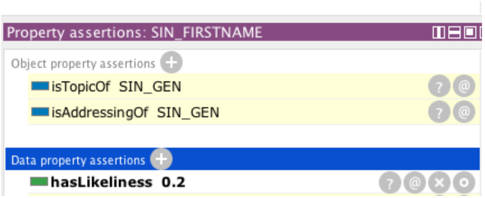



Figure 1.4: A low likeliness value for the instance SIN_FIRSTNAME describing the attitude of a generic Indian person towards being addressed by their first name.

### 1.5.2 hasQuestion

This data property is expected to contain a question that the robot may ask to explore a person's attitude toward corresponding concept: in other words, to ultimately acquire person-specific knowledge starting from culture-specific knowledge. For example, a typical question for the instance SIN_SOCCER may be *Do you watch soccer on TV?*, and the expected answers are *yes*, *no*, *sometimes*, *always*, *of course*. These answers will be interpreted in different ways by the reasoning system that updates the person-specific knowledge.

> *Remark 1.* The Data Property hasQuestion can only be used to encode yes/no or frequency questions: it cannot be used for open questions such as *Can you please tell me more about your family?* For this kind of questions, please refer to the Data Type hasPositiveAndWait.

> *Remark 2.* Instances must have at least one question, otherwise it is not possible for the robot to ask the person about his/her knowledge, attitudes, preferences, habits, values and beliefs. Instances can have more than one question: during chitchatting, one of the available questions will be selected in run-time to increase variability and entertainment value. Notice however that all of the questions that are associated to the same instance must have the same semantics: we cannot have, for the same instance, the two questions *Do you like watching soccer?* and *Do you like playing soccer?* because these are different questions and the answers will be interpreted in different ways. However, we can have the two questions *Do you ever watch soccer on TV?* and *Is watching soccer on TV one of your hobbies?*

### 1.5.3 hasPositiveSentence

This data property is expected to contain a sentence that mirrors the person's positive attitude towards the corresponding concept: that is, it enables the robot to say things that the person may appreciate. A typical positive sentence, if the person previously answered that he/she likes soccer, may be: *Soccer is a very interesting sport, as it requires many different skills* or *I think that soccer matches can be really entertaining to watch, especially if your team is playing* and so on.

> *Remark 1.* Instances must have at least one positive sentence. However, we encourage you to encode more than one positive sentence for each instance. This will increase variability and, ultimately, entertainment value for users: we do not want the robot to repeat the same sentence over and over. During chitchatting, one of the available sentences will be selected in run-time.

### 1.5.4 hasNegativeSentence

This data property is expected to contain a sentence that the robot says if the person answered negatively to the question. A typical negative sentence, if the person previously answered that he/she does not like soccer, may be: *I will remember that*. A stronger version may be *I see. Lots of people like soccer, but I think other sports are more entertaining* or even, more emphatically, *I perfectly understand you, soccer is very boring. I hate soccer*.

*Remark 1.* Using this data property requires careful attention, as we do not want to overemphasize negative responses, as in the last example (*I perfectly understand you, soccer is very boring. I hate soccer)*: however, the robot needs to be able to comment the person's negative answers.

### 1.5.5 hasPositiveAndWait

This data property is the same as hasPositiveSentence, but it allows a person to speak freely: the robot will wait for the person to stop speaking, but it will not understand what the person is saying. A typical positive sentence, if the person previously answered that he/she likes soccer, may be: *I agree that soccer is a beautiful sport: please tell me more about your favorite team*.

*Remark 1.* This Data Property is key, since we want to avoid a pattern where the robot keeps on asking questions after each of the person's answers.

*Remark 2.* All of the Data Properties described here are organized hierarchically, i.e. Data Property hasQuestion has subproperty hasQuestion_0, which in turn has subproperty Question_1, and so on. The reason for this hierarchy will be explained further on, when we describe inherited sentences.

You are now ready for the first exercise!!

## 1.6 Exercise 1. The Cultural Knowledge Base hierarchy

### 1.6.1 Exercise 1.1: Testing the Cultural Knowledge Base hierarchy

In this exercise you will test and modify the hierarchical structure of the ontology, analyzing how the class hierarchy influences the dialogue. Start by testing a limited section of the Cultural Knowledge Base, i.e. *Addressing.owl*.

1. Copy the *Addressing.owl* ontology (from the folder Ontologies) into the root folder and rename it *CKB.owl*

2. Start the textual chitchatting software (please find the instructions for running it in the Introduction); when asked for Conditions, just press Enter

3. Write a sentence and talk with the virtual Pepper robot using the terminal

4. Remember that all direct questions of pepper expect a “yes”-“no” answer (or synonyms, such as “sometimes”, “ok”, “not at all”, ...)

Now you will modify the CKB to obtain different interaction patterns with the virtual robot.

### 1.6.2 Exercise 1.2: Modifying the Cultural Knowledge Base hierarchy

1. Open Protégé

2. Load the *Addressing.owl* ontology

3. Find the individual SIN_BRO (either using the tab *Individuals by class* or *Entities - Individuals*)

4. Modify the value of the DataProperty hasLikeliness (decrease its value)

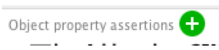
5. Repeat steps 3 and 4 for the individuals SIN_MRLASTNAME and SIN_FIRSTNAME

6. Repeat steps 3 and 4 for the individual SIN_AUNTIE, increasing the value of the DataProperty hasLikeliness

7. From the Tab *Entities - Classes* add the Class MrsLastName as SubClass of Addressing (use the ’Add’ icon )

8. From the Tab *Entities - Individuals* add the individual SIN_MRSLASTNAME, belonging to the class MrsLastName (use the ’Add’ individual icon )

9. Add the DataProperties (hasLikeliness, hasPositiveSentence, hasQuestion)

10. Find the individual SIN_GEN and add an Object Property (use the Add Object Property button from the Property Assertion tab 

)

11. In the new window, fill the fields with the Object Property name hasAddressing and the individual name SIN_MRSLASTNAME

12. Save the modified ontology as *CKB.owl*

13. Copy the ontology from the folder Ontologies into the root folder

14. Start the textual chitchatting software

15. Write a sentence and talk with the virtual Pepper robot using the terminal
16. When finished, open the modified ontology and check the presence of new instances (the user-specific ones).

The basic concept is that the dialogue with Pepper will follow the CKB hierarchy (and the Classes / Individuals structure), and that modifying the Data Property likeliness of some individuals will modify the priorities of the conversation topics.

Please refer to the solution ex1_2.owl for checking your work.

## 1.7 Exercise 2. Adding Classes and Individuals

### 1.7.1 Exercise 2.1: Adding new Classes and Individuals

1. Load the ontology modified in exercise 1.2 (or load ex1_2.owl from the folder Solution)
2. Change the hasLikeliness of the SIN_NAME individual to 0.0
3. Add the class Pizza, as a subclass of Topics
4. Add three subclasses of Pizza, Margherita, Pepperoni and Hawaiian
5. Add the individual SIN_PIZZA (instance of the class Pizza), and the individuals SIN_MARGHERITA, SIN_PEPPERONI and SIN_HAWAIIAN, instances of the three subclasses of Pizza
6. For the individual SIN_PIZZA, add the DataProperty hasKeyword1 with the value "*". Remember to set the tag *Language* to *en* for any Keyword that you add! Keywords will be explained in Chapter 3, here just follow this step for testing purposes
7. For each of the four individuals, add the DataProperty hasLikeliness (in a range between 0.0 and 1.0)
8. For each of the four individuals, add one or more DataProperties hasPositiveSentence (at least one sentence that the virtual robot will say in case of positive feedback from the person). Remember to set the tag *Language* to *en* for any Positive Sentence that you add!
9. For each of the four individuals, add one or more DataProperties hasNegativeSentence (at least one sentence that the virtual robot will say in case of negative feedback from the person). Remember to set the tag *Language* to *en* for any Negative Sentence that you add!
10. For each of the four individuals, add the DataProperty hasQuestion (the question that the virtual robot will ask the person). Remember to set the tag *Language* to *en* for any Question Sentence that you add!
11. For each of the four individuals, add the DataProperty hasPositiveAndWait (the open question that the virtual robot will ask the person). Remember to set the tag *Language* to *en* for any PositiveAndWait Sentence that you add!
12. Link the new instances to the generic Indian person SIN_GEN
13. Save the modified ontology as *CKB.owl*
14. Start the textual chitchatting software
17. Write a sentence and talk with the virtual Pepper robot using the terminal

18. When finished, open the modified ontology and check the presence of new instances (the user-specific ones).

In this exercise, you will test the chitchatting software with a brand new class, Pizza, and some individuals. By changing the values of DataProperties hasLikeliness, hasPositiveSentence, hasNegativeSentence, hasPositiveAndWait and hasQuestion, it is possible to change the dialogue patterns implemented by the virtual robot. Remember to set the tag *Language* to *en* for any sentence that you add!

Please refer to the ex2_1.owl for checking your work.

Didn't do as it is diff

### 1.7.2 Exercise 2.2: Different Individuals for Different Cultures

1. Load the ontology modified in exercise 1.2 (or load ex1_2.owl from the folder Solutions)
2. For each individual in the ontology create a corresponding individual in a different culture (e.g. SJP)
3. For each new instance, add the necessary Data Properties: you may use the same data properties hasQuestion, hasPositiveSentence, hasNegativeSentence (you may change them if you wish), you may use different values for hasLikeliness. Remember to set the DataProperty hasKeyword1 with the value "*" for any instance of the class Name and to set the tag *Language* to *en* for any Keyword and Sentence that you add!
4. Link all new instances to the generic Japanese user, by using Obejct Properties of type hasAddressing
5. Save the modified ontology as CKB.owl
6. Start the textual chitchatting software adding culture as the topic (Indian is the default choice)
7. Write a sentence and talk with the virtual Pepper robot using the terminal
8. When finished, open the modified ontology and check the presence of new instances (the user-specific ones).

You have now tested the possibilities of adding multiple culture-specific instances in the same ontology. Please refer to the solution ex2_2.owl for checking your work.

# CHAPTER 2

# INHERITED SENTENCES

## 2.1 Inherited Data Properties

In this exercise you will focus on two very important concepts in ontologies:

- All of the instances belonging to a class can be specified as having a specific value for a Data Property
- Data Properties are inherited from superclasses to subclasses.

This principle makes it possible to avoid specifying the value of a given data property for each instance of a given class, since we already know that all of those instances share that value. For example, consider the fact that all of the classes in the CARESSES CKB are derived from Topic; this means that all of the instances of those classes are ultimately instances of Topic. Now, suppose we assign a default negative sentence *I will remember that* to the class Topic: that default sentence will be inherited by all of the instances. This makes perfect sense, as the robot's response *I will remember that* is appropriate after any answer the person may give to the robot.

> *Remark 1.* Having a default sentence that is inherited by a superclass does not prevent us from adding a specific sentence that is more appropriate for a given situation. For instance, consider the instance SIT_SOCCER, which inherits the negative sentence *I will remember that*. We can manually add the additional negative answer *I see. Lots of people like soccer, but I think other sports are more entertaining* which works specifically for soccer. When the system has an inherited sentence as well as a manually added sentence, it uses the latter; the inherited sentence is only used if it there is no specific sentence available.

*Remark 2.* Inheriting Data property values adds no functionalities to the system, it is just a faster way of building the ontology. You can ignore this option when filling the ontology with new classes, instances, or Data Properties; however, it is important to understand this principle to interpret correctly what is written in the ontology.

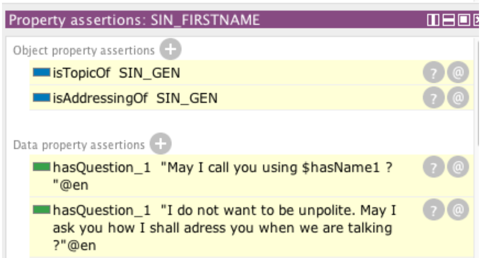



Figure 2.1: Inherited sentences of the instance SIN_FIRSTNAME.

## 2.2 Composing sentences: the Data Property hasName

Using inherited values for Data Properties can make the whole process of filling the ontology faster, but it only works in very few cases. It is actually difficult to find a sentence that is appropriate for all subclasses and their instances. The negative sentence *I will remember that* is very generic, while we would like the robot to be able to say things that are specific and appropriate.

Think about the question *How shall I address you ?* which the robot may ask to learn how the person prefers to be addressed. This question will appear in many different instances, e.g.; *Shall I address you by your first name?* or *Shall I address you by your last name?* or *Shall I address you as Auntie?*. The sentence is made of two parts:

- an invariant part
- a variable part

In the example above, the invariant part is *shall I address you?* whereas the variant part is *by your first name*, or *by your last name*, or *as Auntie*. The variant part is encoded in a new Data Property, hasName. All instances will have the Data Property hasName filled with some words, so the instance SIN_FIRSTNAME can be assigned the value *by your first name*; the instance SIN_LASTNAME can be assigned the value *by your last name*; the instance SIN_AUNTIE can be assigned the value *as Auntie*.

Once we define the variable part hasName, the system can build sentences automatically. Suppose that the instance SIN_FIRSTNAME is assigned the following question for the Data Property hasQuestion: *Shall I address you $hasName?*. Because it includes the symbol $, the expression *$hasName* is interpreted by the system as a variable. During chitchatting, the robot will automatically replace the variable *$hasName* with the corresponding value of the hasName Data Property, finally yielding the question *Shall I address you by your first name?*

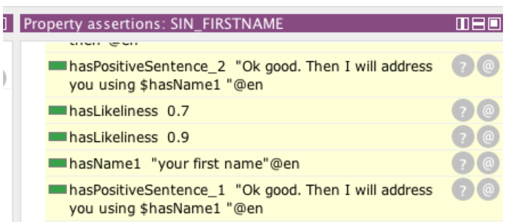



Figure 2.2: Inherited sentences of the instance SIN_FIRSTNAME and Data Property hasName

Consider how powerful this mechanism is. SIN_FIRSTNAME is an instance of the class FirstName, which is derived from the superclass Addressing. Likewise, SIN_LASTNAME is an instance of LastName and SIN_AUNTIE is an instance of Auntie, where both LastName and Auntie are derived from the same superclass Addressing. Suppose that SIN_LASTNAME has been assigned the value *by your last name* for the Data Property hasName and SIN_AUNTIE has been assigned the value *as auntie*. Instead of assigning the question *Shall I address you $hasName ?* to each instance, all we need to do is assign that question to the superclass Addressing: the assigned value of hasQuestion is inherited by all subclasses and therefore by all instances, and substituted in run-time - i.e., during chitchatting - with the corresponding values of the Data Property hasName.

> *Remark 1.* In actual fact, the principle is a little more complex, as the same instance may have different values of the Data Property hasName, which can be used in different situations; this feature will be partially shown through the next exercises, but it is not necessary to understand the basic functioning of the system. In general terms, different Data Properties hasName1, hasName2, hasName3 can be used, depending on which is more appropriate to the situation. For example, instances of the class Dog may have *A dog* as the value of the Data Property hasName1, *the dog* as the value of the Data Property hasName2, and *Your dog* as the value of the Data Property hasName3. Data Properties hasName1, hasName2 and hasName3 are all subproperties of hasName.

> *Remark 2.* All of the Data Properties that are subproperties of hasName are organized hierarchically, i.e. Data Property hasName1 has subproperty hasName1_0, which in turn has subproperty hasName1_1, and so on. This hierarchical structure is necessary because, in principle, hasName properties can be assigned to classes, and thus inherited by instances. However, instances will inherit all of the properties of the superclasses: for example, instances of the class Fridge may inherit the Data Property hasName1 *a fridge* from their class, but they may also inherit the Data Property hasName1 *an appliance* from the class Home Appliance, which is a superclass of Fridge. While this is correct from the point of view of the ontology (a fridge is an appliance) it may generate ambiguities for automatically generated sentences. Thus, in this case, instances of the class Fridge will inherit the Data Property hasName1_1 *a fridge* from the class Fridge and the Data Property hasName1_0 *an appliance* from the class Home Appliance. Basically, the principle is that Data Property follows the hierarchy of the ontology. The system will always consider the Data Property with the higher index when automatically generating sentences. Properties related to sentences are organized the same way.

> *Remark 3* Be careful when using the inheritance mechanism: if you assign the question *Shall I address you $hasName1 ?* to the superclass Addressing, you need to make sure that the instances of the class Addressing do not have a hasName1 property that may lead to non-sense sentences like *Shall I address you to address?*

## 2.3 Exercise 3. Inherited Sentences and hasName

1. Open Protégé
2. Load the ontology *Addressing2.owl*
3. Add Data Properties hasQuestion_0, hasPositiveSentence_0 and hasNegativeSentence_0 to the class Name. In order to add a Data Property to the whole class, you have to add a *restriction* to that class. To this aim, use the AddRestriction button (the + close to *Subclass of*) and in the tab *Class expression editor* write the name of the Data Property that you want to add, the tag *value* and the actual value of the Data Property. Please refer to Fig. 2.3 for an example. Remember to set the tag *Language* to *en* for any Sentence that you add!

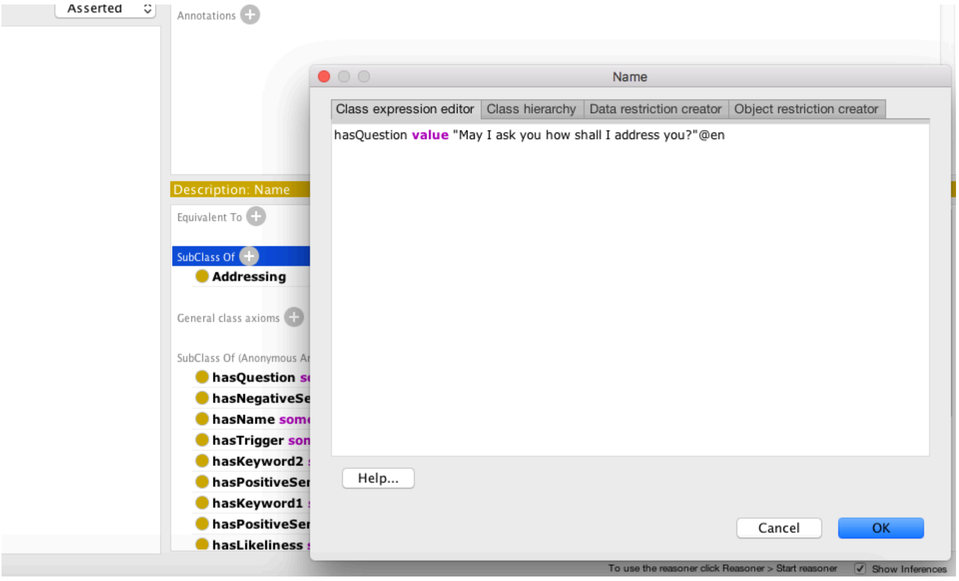



Figure 2.3: Adding a Data Property hasQuestion to the class Name.

4. Add Data Properties hasQuestion_1 and hasPositiveSentence_1 to the class Name using the variable hasName1. The result should be something similar to the one shown in Figure 2.4


Figure 2.4: Data Properties of the class Name.

5. Add the Data Property hasName1 in all subclasses of Name. Remember to set the tag *Language* to *en* for any hasName that you add!
6. Add the class MrsLastName with culture-specific and user-specific instances. Add the Data Property hasName1 to the class, and the likeliness to the culture-specific instance. Connect the culture-specific instance with the user-specific instance using the Object Property hasSpecific. Then add the Data Property hasName1 to the user-specific instance
7. Save the Ontology as *CKB.owl*
8. Start the textual chitchatting software
9. Write a sentence and talk with the virtual Pepper robot using the terminal.

Exercise 3 shows how it is possible to talk with Pepper using sentences that are inherited by the class, and taking advantage of the Data Property hasName, both for culture-specific and user-specific instances. You can practice this some more using the Class Pizza created earlier, and adding more inherited Question, Positive, PositiveAndWait and Negative Sentences.

Please refer to the solution ex3.owl for checking your work.

**CHAPTER 3**

# OTHER DATA PROPERTIES

In order to have a complete idea of the CARESSES Ontology structure, we need to describe the other Data Properties that are required for chitchatting: hasQuestionContextual, hasQuestionGoal, hasKeyword1 and hasKeyword2.

## 3.1 hasQuestionContextual and hasQuestionGoal

These two Data Properties are similar, in the sense that they are both used after the Question corresponding to the Data Property hasQuestion has been asked by the robot and positive feedback has been received from the person.

### 3.1.1 hasQuestionContextual

While the data property hasQuestion is expected to contain a question that the robot can ask to explore the person's general attitude towards a given concept, the data property hasQuestionContextual asks specific questions about a concept. In other words, while the data property hasQuestion refers to a general preference (as expressed using time markers such as *usually* or *sometimes)*, the hasQuestionContextual always refers to a preference that is true at the time of speaking. For example, the instance SIN_SOCCER_WATCHINGSPORT can have *Do you usually watch sports on TV?* as Data Property hasQuestion and *Are you planning to watch soccer on TV today?* as hasQuestionContextual.

### 3.1.2 hasQuestionGoal

This data property has the same meaning as hasQuestionContextual, but can also be linked to a Goal that the robot can achieve if the feedback from the person is positive. In the example above, *Do you want to watch soccer on TV now?* can be a property of type hasQuestionGoal and in that case the Dialogue Manager will generate a goal for the planner (i.e. switch on the TV or accompany the person to the TV).

## 3.2 hasKeyword1 and hasKeyword2

These two Data Properties contain keywords that may trigger a specific dialogue between the robot and the person. Indeed, the chitchat action is usually activated when the robot is in the *AcceptRequest* state. In this context, the robot asks the person if it may be of help, and then it waits for the person to say something. When a sentence is detected, the robot checks if it contains any keywords that are connected to goals (e.g., *play music*, or *show weather*), and then it checks if the sentence contains any keywords related to conversation topics. These keywords are described with the hasKeyword1 and hasKeyword2 data properties.

*Remark 1.* Keywords are usually assigned to classes and then inherited by instances.

*Remark 2.* If multiple instances are triggered by the same keywords (which is likely to occur when keywords are assigned to classes), the keyword with the highest likeliness is chosen for the dialogue. Along the same lines, if the robot detects no keyword, it starts to talk about the conversation topic with the highest likeliness. If only one keyword is detected (hasKeyword1 or hasKeyword2), the topic is chosen by the robot only if there are no other topics that are triggered by a keyword contained in hasKeyword1 or a keyword contained in hasKeyword2.

*Remark 3.* Instances and classes can have more than one hasKeyword1 and hasKeyword2 Data Property. For example, instances of the class ItalianFood may have the Data Property hasKeyword1 *Italian* and the Data Properties hasKeyword2 *Food*,*Restaurant*, *Cuisine*.

## 3.3 Exercise 4

1. Open Protégé
2. Load the ontology created in the Exercise 2.1 (or load ex2_1.owl from the folder Solutions). Change the likeliness of SIN_NAME to 0.5.
3. Add person-specific likeliness about Pizzas (i.e., add first person-specific instances, link them to the corresponding culture-generic instance by the Object property hasSpecific, and finally add the property hasLikeliness to the user-specific instance)
4. Add person-specific likeliness towards forms of address (see point 3), and person-specific sentences, both for pizzas and form of address.
5. Add properties hasQuestionContextual and hasQuestionGoal (indifferently to culture-specific instances or user-specific instances of the class Pizza)
6. Remove the properties hasKeyword1 for the instance SIN_PIZZA and SIN_NAME and add different trigger keywords to different culture-generic instances
7. Test the modified Ontology

We have added questions for suggesting actions and goals to the person. Please refer to the solution ex4.owl for checking your work.

## CHAPTER 4

# TOPIC TREE AND ONTOLOGY HIERARCHY

### 4.1 Topic Tree

In the previous chapters we analyzed the structure of the CARESSES ontology, assuming that the hierarchy of the dialogue will always follow the hierarchy of the ontology (as shown, for instance, in Exercise 3). However, this is not always true. Instances belonging to classes that are not in a hierarchical structure from the point of view of the ontology may still be organized hierarchically from the point of view of the dialogue. For example, the robot may want to ask the person if they have biscuits for breakfast. Clearly, the robot should ask the person if they have breakfast in the morning first. Thus, we may have instance SIN_HAVINGBREAKFAST, belonging to the class HavingBreakfast, which is directly connected to generic Indian person SIN_GEN. The class HavingBreakfast will probably be a subclass of the class DailyRoutine and more generally of the class Habit. Of course the instance related to the person's attitude towards having biscuits for breakfast might not be a subclass of HavingBreakfast because it is related to the attitude of the person towards an object, not a habit. Therefore, it will be an instance of the class Biscuit, a subclass of Food and more generally of the class Object.

In order to express the connection between the habit of having breakfast and the object *biscuit*, an object property of type hasTopic (hasFood) is used once again. However, in this case it will directly link the instance SIN_HAVINGBREAKFAST to the instance SIN_BISCUIT_HAVINGBREAKFAST of the class Biscuit.

> *Remark 1.* The name of the instance is SIN_BISCUIT_HAVINGBREAKFAST because it is related to the person's attitude towards having biscuits for breakfast. In principle different instances of biscuits (e.g., SIN_BISCUIT_HAVINGLUNCH, SIN_BISCUIT_BUYING, ...) may exist in the same class.

> *Remark 2.* The hasTopic construction and the Ontology hierarchy will both be used when creating the Topic Tree. For example, to express the habits of an Indian person who has Italian food for lunch and pizza for lunch, the SIN_GEN instance

will be linked by the Object Property hasHabit (subproperty of hasTopic) to the instance SIN_HAVINGLUNCH. In turn, SIN_HAVINGLUNCH will be linked to the instances SIN ITALIANFOOD HAVINGLUNCH and SIN_PIZZA HAVINGLUNCH by the Object Property hasFood (subproperty of hasTopic). However, SIN_PIZZA_HAVINGLUNCH will be an instance of the class Pizza, subclass of ItalianFood (class of SIN_ITALIANFOOD_HAVINGLUNCH). Thus, when talking with the person, the robot will first ask if the person usually has lunch, then it will investigate the persons's attitude towards having Italian food for lunch and (if the feedback is positive) at the end the robot will ask the person if they sometimes have pizza for lunch.

*Remark 3.* It is possible to build automatically generated sentences using the Object Property hasTopic in addition to the ones directly generated by the class inheritance. In this case, the variable to be substituted is expressed in the form $[ObjectProperty]*[DataProperty] (e.g. *$hasFood*hasName1*).

### 4.2 Exercise 5

1. Open Protégé
2. Load the Addressing.owl ontology
3. Create the Classes Habit (subclass of Topic), DailyRoutine (subclass of Habit), HavingMeal, HavingLunch (subclasses of DailyRoutine)
4. Create the Classes Food (subclass of Topic), Pizza (subclass of Food), PizzaMargherita, PizzaPepperoni (subclasses of Pizza)
5. Create a culture-specific instance of the class HavingLunch, link SIN_GEN to the new instance with the hasHabit Object property, and add the necessary Data Properties (likeliness, sentences). Add the DataProperty hasKeyword1 with the value "*". Remember to set the tag *Language* to *en* for any sentence and keyword that you add!
6. Create culture-specific instances related to the use's attitude towards having pizza for lunch (i.e. SIN_PIZZA_HAVINGLUNCH, SIN_PIZZAMARGHERITA_HAVINGLUNCH, etc.).
7. Link the instance of the class HavingLunch with the three instances of the class Pizza (and its subclasses), with an Object Property of type hasFood
8. Add Data properties hasLikeliness, hasQuestion, hasPositiveSentence and hasNegativeSentence to the new instances SIN_PIZZA_HAVINGLUNCH, SIN_PIZZAMARGHERITA_HAVINGLUNCH and SIN_PIZZAPEPPERONI_HAVINGLUNCH
9. Save and test the modified Ontology

You have learned to use the object properties for connecting, in the same branch of the topic tree, individuals belonging to different classes. Please refer to ex5.owl in the folder Solutions for checking your work.

### 4.3 Exercise 6

1. Open Protégé
2. Load the Environment.owl ontology
3. Create some user-specific instances with hasLikeliness equal to 1. You may add some user-specific sentences, enabling the robot to talk about specific aspects of the person's

environment. As usual, remember to set the tag *Language* to *en* for any Sentence that you add

4. Save and test the modified Ontology

Now we have used the topic tree structure together with some user-specific instances.
Please refer to ex6.owl in the Folder Solutions for checking your work.

**CHAPTER 5**

# OBJECT PROPERTY **HASCONDITION**

This chapter describes an additional Object Property that is relevant to chitchatting: hasCondition.

## 5.1 hasCondition

The system we have described so far enables the robot to choose any conversation topic described in the ontology, using the ontology structure, the hasTopic properties, likeliness and triggering keywords to pick a conversation topic that is appropriate to the person. However, some topics would clearly be strange or unlikely at certain times of the day (e.g. talking about breakfast at 5 p.m.), while other topics would be almost certainly appropriate in given situations (e.g. asking about forms of address when first meeting a person).

In order to take similar situations into account, two additional Object Properties have been added, which are both subproperties of hasCondition: hasNecessaryCondition and hasTriggeringCondition.

### 5.1.1 hasNecessaryCondition

This Object Property links a generic topic of conversation with a condition that has to be verified in order for the robot to choose it. Such conditions can have to do with the time of the day, location, or general situations. In the example above, the instance SIN_HAVINGBREAKFAST will be linked to the instance SIN_MORNING (of the Class Time) by an Object Property of type hasNecessaryCondition.

### 5.1.2 hasTriggeringCondition

This Object Property links a generic topic of conversation with a condition that, if verified, immediately triggers the related conversation topic. In the example above, the instance SIN_NAME will be linked to the instance SIN_FIRSTROBOTENCOUNTER (of the Class Event) by an Object Property of type hasTriggeringCondition.

> *Remark 1.* Instances that represent conditions should have a Data Property hasValue, which specifies the related conditions.

### 5.2 Exercise 7

1. Open Protégé
2. Load the ontology modified in Exercise 5 (or load ex5.owl from the folder Solutions)
3. Add a triggering condition of the topic *HavingLunch*: link the instance SIN_HAVINGLUNCH with a certain condition (i.e. SIN_1PM, that could be an instance of the class 1PM, subclass of HourOfTheDay, subclass of Time) with the ObjectProperty hasTriggeringCondition. In the condition instance, add the DataProperty hasValue, with value equal to 1pm
4. Following the same procedure, add a necessary condition (i.e. the location dining room) for the instance SIN_HAVINGLUNCH
5. Test the modified Ontology: when asked for Condition, you may put "1 pm" and "dining room", and later on you may try to change them (for example you may just press Enter, without adding conditions, or just add the condition "dining room")

   Finally we have tested the usage of necessary and triggering conditions for a context-aware chitchatting. Please refer to the file ex7.owl for checking your work.

Now you are ready to work on the complete CARESSES Ontology (the file Caresses.owl). You may at first try to talk with the virtual robot, keeping the ontology as it is (for testing it with the enclosed software, please rename it as CKB.owl and move it in the same folder of the software), then you may try to add properties, instances and classes, and test the result with the software enclosed.

## Appendix

### A.1 Creating a new dialogue topic (culture-specific)

1. Select the class to which the topic belongs, or create a new class
2. Add the instance (you can use the tab *Individuals by class*)
3. Add DataProperties (at least hasLikeliness, hasQuestion, hasPositiveSentence, hasNegativeSentence)
4. Link the instance with an Object Property of type hasTopic to SIN_GEN or to a different instance
5. Save the Ontology

### A.2 Creating new information (user-specific)

1. Add the user specific instance (You can use the tab *Individuals by class*)
2. Add DataProperties (at least hasLikeliness)
3. Link the instance with an Object Property of type hasSpecific to the corresponding culture-specific instance
4. Save the Ontology