

# **CARESSES ONTOLOGY**



---

# **CARESSES ONTOLOGY**

## **Tutorial for building and modifying the CARESSES Ontology**

---

**Carminé Tommaso Recchiuto**  
Università di Genova

**Antonio Sgorbissa**  
Università di Genova

# CONTENTS

---

Introduction	vi
<b>1 The structure of the ontology</b>	<b>1</b>
1.1 Classes	1
1.2 Instances	2
1.2.1 Culture-specific instances and Person-specific instances	2
1.3 Chit-chatting	3
1.4 Object Properties	4
1.4.1 hasTopic	4
1.5 Data Properties	5
1.5.1 hasLikeliness	5
1.5.2 hasQuestion	6
1.5.3 hasPositiveSentence	7
1.5.4 hasNegativeSentence	7
1.5.5 hasPositiveAndWait	7
1.6 Exercise 1. The CKB hierarchy	8
1.6.1 Exercise 1.1: Testing the CKB hierarchy	8
1.6.2 Exercise 1.2: Modifying the CKB hierarchy	8
1.7 Exercise 2. Adding Classes and Individuals	9
1.7.1 Exercise 2.1: Adding new Classes and Individuals	9
1.7.2 Exercise 2.2: Different Individuals for Different Cultures	10

<b>2</b>	<b>Inherited Sentences</b>	<b>11</b>
2.1	Inherited Data Properties	11
2.2	Composing sentences: the Data Property <code>hasName</code>	12
2.3	Exercise 3. Inherited Sentences and <code>hasName</code>	14
<b>3</b>	<b>Other Data Properties</b>	<b>15</b>
3.1	<code>hasQuestionContextual</code> and <code>hasQuestionGoal</code>	15
3.1.1	<code>hasQuestionContextual</code>	15
3.1.2	<code>hasQuestionGoal</code>	16
3.2	<code>hasKeyword1</code> and <code>hasKeyword2</code>	16
3.3	Exercise 4	16
<b>4</b>	<b>Dialog Tree</b>	<b>17</b>
4.1	Topic Tree	17
4.2	Exercise 5	18
4.3	Exercise 6	19
<b>5</b>	<b>Condition</b>	<b>20</b>
5.1	<code>hasCondition</code>	20
5.1.1	<code>hasNecessaryCondition</code>	20
5.1.2	<code>hasTriggeringCondition</code>	21
5.2	Exercise 7	21
	Chapter Appendix	22
A.1	General Steps for creating a new dialogue topic (culture-generic)	22
A.2	General Steps for creating a new user-specific information	22

# INTRODUCTION

---

This Section of the course allows you to learn how the hierarchy of the *CARESSES CKB* is organized, and how it is possible for the robot to talk with the person about concepts that are represented in the CKB. Talking with the person has three main purposes:

- Entertain the person by talking about topics she/he is familiar with;
- Suggest to the person activities to be performed, i.e., what the robot may do in order to help or entertain her/him;
- Acquire new knowledge about the person attitudes, preferences, habits, values, beliefs. This ultimately helps the robot to i) talk about topics the person is familiar with, and ii) suggest to the person what the robot may do for her/him.

In order to perform the following exercises, it is necessary to keep in mind how the basic elements of standard ontologies are used in the in the CARESSES CKB:

- Classes
- Instances
- Chit-chatting
- Object properties
- Data properties

All these basic elements will be described in the Chapter 1. The other Chapters will describe additional elements of the Ontology and will introduce the related exercises.

This guide will refer to Protégé 5 for creating and modifying OWL ontologies. Class, property and individual names are written in a sans serif font like this.

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

```
java -jar CKB_tutorial.jar [optional argument:culture (Indian as default)]
```

## CHAPTER 1

---

# THE STRUCTURE OF THE CARESSES ONTOLOGY

---

In the following, the main components of the Ontology, with particular reference to the CARESSES structure, will be analysed.

### 1.1 Classes

The classes of the CARESSES ontology are used to represent information that is culture-independent: therefore, this part of the ontology is not different from any other ontology you may know (e.g., the Pizza ontology). The only important thing that the reader shall remember is that all CARESSES classes are derived from a superclass called

#### Topic

By deriving subclasses from **Topic** we represent all concepts that may be relevant in the application domain (in principle, by considering all cultures of the world; more realistically, by considering 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. As an example, please notice the concept representing the Diwali festival of light, which is very important in the Hindu culture, will be present also in the ontology that the robot uses when interacting with a Japanese person, even if it is unlikely that the person knows about this festival (but not impossible!). Since we want to avoid stereotyped representations, we allow the system to virtually talk with the person about everything, 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 therefore shall be represented in different places of the hierarchy.

*Remark 1* In the CARESSES CKB, classes are written in small letters, without spaces, and using capitol letter to outline the first letter of each word, if the class name is composed of more than one word (e.g., Habit, WatchingMovies).

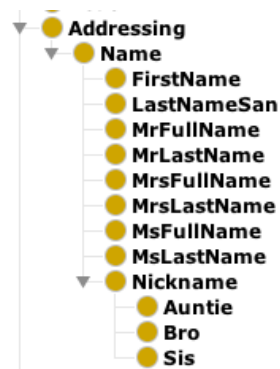


Figure 1.1: The Figure shows the hierarchy of classes that are necessary for the robot to talk with the person about different ways of addressing her/him. The class **Addressing** has the subclasses **FirstName**, **LastNameSan**, **Nickname**, and so on.

## 1.2 Instances

Instances of the CARESSES ontology may be of two kinds:

- culture-specific instances
- person-specific instances.

In this case, CARESSES makes a distinction that is not standard in the ontologies you may know (e.g., the Pizza ontology).

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

Culture-specific instances describe the knowledge, attitudes, preferences, habits, values, beliefs, etc., that are 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 aims at avoiding stereotypes: even if we can make some reasonable assumptions about the habits of a person by knowing the cultural group she belongs to, these assumptions should necessarily be interpreted in a probabilistic sense. For instance, we cannot make any strong assumption that an Italian person loves soccer (even if there is a high chance for this to be true), or that a Japanese person has Miso soup for breakfast (as an aside, it is well known that a western style breakfast is becoming more and more popular in Japan).

## 2 CARESSES Ontology.

By Carmine Recchiuto, Antonio Sgorbissa

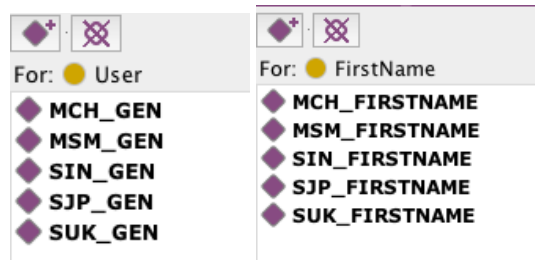


Figure 1.3: The Figure shows the instances of the class `User` and `FirstName`, that represent the generic Indian, English and Japanese user and the attitude of a generic Indian, English, Japanese person towards being addressed using the first name (we expect that different cultural groups may like or not like this possibility). The Figure shows also the instances `MSM_FIRSTNAME` and `MCH_FIRSTNAME`, that describe - respectively - what Mrs Smith and Mrs Chaterjee think about being addressed using the 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 allows for making a distinction between 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 in the following. Then, the culture-specific instance of the class `WatchingMovies` for the Indian culture will be `SIN_WATCHINGMOVIES...`, whereas the person-specific instances of the same class for Mrs Margaret Smith may look like `MMS_WATCHINGMOVIES`.

*Remark 2* Culture-specific instances and Person-specific instances should be always 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: the generic Indian, English, Japanese users are referred to, respectively, as `SIN_GEN`, `SUK_GEN`, `SJP_GEN`.

### 1.3 Chit-chatting

As anticipated, the main purposes of the CARESSES CKB are (i) to talk with the user about topics he/she is familiar with and (ii) to suggest goals to be achieved and/or activities to be performed (this second objective is not discussed here). However, in order to meet the person's expectations, (iii) the system must acquire knowledge about the person's attitudes, preferences, habits, values, beliefs, etc. In the simplest case, we can imagine that the system starts from an initial situation in which it has only knowledge about the cultural group the person belongs to, but no specific knowledge about the person herself (i.e., only culture-specific instance are initially present). The purpose of chit-chatting is to acquire knowledge about the person (i.e., person-specific instances in the CKB) starting from the a priori assumption 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 preferences of the person, it will start by exploring the most probable options: if the person is Italian, suggesting coffee and biscuits may be good options, that the robot is ready to revise if it discovers that these options are not welcome by the person.

*Remark* As we will see in the following, it is possible that we have some initial knowledge about the person. If this is true, we can encode initial knowledge in the system in the setup phase, by creating proper person-specific instances together with culture-specific instances. This will be shown in one of the following exercises.

## 1.4 Object Properties

As usual in Ontologies, Object properties may be defined in the CARESSES CKB and represent links between different classes; analogously, instances of Object properties represents links between different instances in the ontology. There are a few pre-defined Object Properties in the CARESSES ontology:

- `hasTopic`
- `hasCorrelation`
- `hasCondition`

The most important Object Property is `hasTopic`, from which almost all Object Properties in the CARESSES ontology are derived. In the following, we will describe only `hasTopic`, whereas `hasCorrelation` and `hasCondition` will be discussed later.

### 1.4.1 `hasTopic`

This Object Property basically describes the fact that a class (derived from `Topic`) may be a possible topic of discussion for the robot. The Object Property `hasTopic` is used to link the instance that represent the generic user belonging to a cultural group (e.g., `SIN_GEN`, `SUK_GEN`, `SJP_GEN`) to all instances that the robot is allowed to talk about. For instance, by linking the generic Indian person `SIN_GEN` to `SIN_WATCHINGMOVIES`, the robot is allowed to talk about the person's preference concerning watching movies; by linking the prototypical English person `SUK_GEN` to `SUK_COFFEE`, the robot is allowed to talk about the person's preferences concerning coffee.

*Remark 1* Culture-specific instances corresponding to different cultures are never connected to each other: for instance, `SIN_GEN` shall be connected exclusively to instances with prefix `SIN`, and `SUK_GEN` only to instances with prefix `SUK`.

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

*Remark 3* To take advantages of the hierarchical structure of the ontology, the `hasTopic` property is never used to directly connect culture-specific instances to each other in the

CARESSES CKB. On the opposite, a hierarchy of subproperties are derived by `hasTopic`, each with a well-defined semantics. As an example, `SIN_GEN` is likely to be connected to `SIN_WATCHINGMOVIES` by an Object Property `hasHobby` and `SUK_GEN` is likely to be connected to `SUK_COFFEE` by an Object Property `hasBeverage`, where both `hasHobby` and `hasBeverage` are derived from `hasTopic`.

hasHealthProblem	SIN_HEARTDISEASE	?	@	x	o
hasHabit	SIN_SINGING	?	@	x	o
hasAddressing	SIN_NAME	?	@	x	o
hasAddressing	SIN_MRLASTNAME	?	@	x	o
hasHabit	SIN_COOKING	?	@	x	o
hasHabit	SIN_ATTENDINGWOMENCLUB	?	@	x	o
hasAddressing	SIN_BRO	?	@	x	o
hasHabit	SIN_LISTENINGTOMUSIC	?	@	x	o
hasHealthProblem	SIN_HEALTHPROBLEM	?	@	x	o
hasHabit	SIN_LEAFPPEEPING	?	@	x	o
hasLife	SIN_USERMARRIAGE	?	@	x	o
hasHabit	SIN_WATCHINGTV	?	@	x	o
hasRelative	SIN_BROTHER_FAMILY	?	@	x	o
hasHabit	SIN_ATTENDINGBOOKCLUB	?	@	x	o
hasVolume	SIN_HIGHERVOLUME	?	@	x	o
hasAddressing	SIN_SIS	?	@	x	o
hasHabit	SIN_READINGBOOK	?	@	x	o

Figure 1.4: The generic Indian user and some Object properties (all subproperties of `hasTopic`)

## 1.5 Data Properties

All classes of the CARESSES CKB that are relevant for the discussion have data properties that allow 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 following exercises):

- `hasLikeliness`
- `hasQuestion`
- `hasPositiveSentence`
- `hasNegativeSentence`
- `hasPositiveAndWait`

### 1.5.1 hasLikeliness

The link between culture-specific instances and person-specific instances is given by a data property called `hasLikeliness`. All the culture-specific instances in the ontology must have a likeliness value ranging from 0 to 1 (i.e., interpreted as a probability). If a culture-specific instance has a high likeliness value, this means that 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, however it is definitely more probable that he or she likes soccer rather than badminton (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` (that 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 a value 0.9 for the Data Property `hasLikeliness`.

As already mentioned, we may wish to represent the fact that an Italian person may also like badminton, even if this is less probable: then, the robot must be given the possibility to explore this possibility as well. To achieve this, it is sufficient to create a culture-specific instance `SIT_BADMINTON` (representing badminton) and to connect `SIT_GEN` with `SIT_BADMINTON`. Presumably, we might decide that `SIT_BADMINTON` has a lower value than `SIT_SOCCER` for the Data Property `hasLikeliness`.

Similarly, we want to consider that an Indian person may or not may like soccer and badminton: then, we will connect `SIN_GEN` with `SIN_SOCCER` and `SIN_BADMINTON` through the Object Property `hasSport` exactly as we did for the Italian prototypical 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: which is the probability that an Indian person may like Sumo? All instances for which the information is not easily found, may be assigned a low uniform probability value.

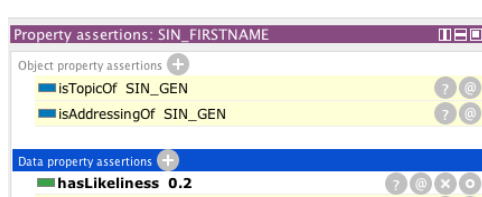


Figure 1.5: A low likeliness value for the instance `SIN_FIRSTNAME` (describing the attitude of a generic Indian person towards being addressed using the first name).

### 1.5.2 hasQuestion

This data property is expected to contain a question that the robot may ask in order to explore the attitude of the person concerning the corresponding concept: that is, to ultimately acquire person-specific knowledge starting from culture-specific knowledge. A typical question for the instance `SIN_SOCCER` may be *Do you watch soccer on TV?*, and the expected answers are *yes, no, maybe, sometimes, always, of course*. These answers will be interpreted in different ways by the reasoning system that updates the person-specific knowledge, but the details are out of the scope of this tutorial.

**Remark 1** It shall be noticed that the Data Property `hasQuestion` can only be used to encode questions whose answers can be positive, negative (or values in-between): it cannot be used for open questions such as *Can you please tell me more about your family?* For this latter kind of questions, please refer to the Data Type `hasPositiveAndWait`.

*Remark 2* Each instance shall 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, beliefs. However, an instance can also have more than one question: during chit-chatting, one of the available questions will be selected in run-time to increase variability (and entertainment). Notice however that all questions associated to the same instance must have the same semantics: I cannot have, for the same instance, the two questions *Do you like watching soccer?* and *Do you like playing soccer?* since these are different questions and the answers shall be interpreted in different ways. However, I can have the two questions *Do you sometimes watch soccer on TV?* and *Is watching soccer in TV a hobby for you?*

### 1.5.3 hasPositiveSentence

This data property is expected to contain a sentence that enforce the positive attitude of the person towards the corresponding concept: that is, to allow 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 game, as it requires many different skills* or *I think that a soccer match is really entertaining, especially when your team is playing* and so on.

*Remark 1* Each instance shall have at least one positive sentence. However, we encourage the reader to encode more than one positive sentence for each instance. This will increase variability and, ultimately, entertainment: we do not want the robot to repeat the same sentence over and over. During chit-chatting, one of the available sentence will be selected in run-time.

### 1.5.4 hasNegativeSentence

This data property is expected to contain a sentence that is told by the robot if the person answered negatively to the question. A typical negative sentence, if the person previously answered that he/she do not like soccer, may be: *I will remember that.* A stronger version may be *I see. Many persons like soccer, but there are more interesting sports* or even the very strong *I perfectly understand you, soccer is very boring. I hate soccer.*

*Remark 1* We need to be very careful about this, as we do not want to enforce negative feelings: however, the capability to reply in case of a negative answer is fundamental and shall be considered.

### 1.5.5 hasPositiveAndWait

This data property is identical to `hasPositiveSentence`, but it leaves the person the opportunity 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: if you wish, please tell me more about your favourite team and the matches that you have seen.*

*Remark 1* The usage of this Data Property is key, since we want to avoid the situation that the robot asks questions, gets answers, asks another questions, over and over again.

*Remark 2* All Data Properties described here are organized in a hierarchical way, i.e. the Data Property `hasQuestion` will have the subproperty `hasQuestion_0`, that in turn will have the subproperty `Question_1`, and so on. The motivation of this hierarchy will be explained in the following, when described inherited sentences.

You are now ready for the first exercise!!

## 1.6 Exercise 1. The CKB hierarchy



### 1.6.1 Exercise 1.1: Testing the CKB hierarchy

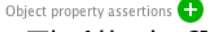
The first exercise is mainly focused on testing and modifying the hierarchical structure of the ontology, analysing how the class hierarchy influences the dialogue. At first, you may try testing a limited section of the Cultural Knowledge Base, i.e. *Addressing.owl*.

1. Copy the *Addressing.owl* ontology into the root folder, and rename it as *CKB.owl*;
2. Start the chitchatting textual software;
3. Write a sentence and talk with the virtual Pepper robot by using the terminal.

The idea now is to modify the CKB in order to obtain different interaction patterns with the virtual robot.

### 1.6.2 Exercise 1.2: Modifying the CKB hierarchy

1. Open Protégé
2. Load the *Addressing.owl* ontology
3. Find the individual `SIN.BRO` (either by 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 into the root folder.
- 14. Start the chitchatting textual software;
- 15. Write a sentence and talk with the virtual Pepper robot by using the terminal.

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

## 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 the previous exercise.
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_PEPERONI` and `SIN_HAWAIIAN`, instances of the three subclasses of `Pizza`.
6. For each of the four individuals, add the DataProperty `hasLikeliness` (in a range between 0.0 and 1.0)
7. For each of the four individuals, add one or more DataProperties `hasPositiveSentence` (at least one sentence that the virtual robot would say in case of a positive feedback from the user)
8. For each of the four individuals, add one or more DataProperties `hasNegativeSentence` (at least one sentence that the virtual robot would say in case of a negative feedback from the user)
9. For each of the four individuals, add the DataProperty `hasQuestion` (the question that the virtual robot will ask to the user)
10. For each of the four individuals, add the DataProperty `hasPositiveAndWait` (the open question that the virtual robot will ask to the user)
11. Link the new instances to the generic indian user `SIN_GEN`;



12. Save the modified ontology as *CKB.owl*;
13. Start the chitchatting textual software;
14. Write a sentence and talk with the virtual Pepper robot by using the terminal.

With this exercise, the user is required to test the chitchatting software with a brand new class, *Pizza*, and some individuals. By changing the values of DataProperties *hasLikelihood*, *hasPositiveSentence*, *hasNegativeSentence*, *hasPositiveAndWait* and *hasQuestion*, the user can change the dialogue patterns implemented by the virtual robot.

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

1. Load the ontology modified in the previous exercise;
2. Change the *hasLikelihood* of the *SIN\_NAME* individual to 0.7
3. For all individuals existing in the ontology create the corresponding individual related to a different culture (e.g. *SJP*).
4. Save the modified ontology as *CKB.owl*
5. Start the chitchatting textual software adding as argument the culture (by default, the Indian culture is considered).
6. Write a sentence and talk with the virtual Pepper robot by using the terminal.

## CHAPTER 2

---

# INHERITED SENTENCES

---

### 2.1 Inherited Data Properties

The second exercise requires to consider two very important concepts in ontologies:

- It is possible to specify that all instances belonging to a class have a specific value for a Data Property
- Data Properties are inherited from superclasses to subclasses.

As the reader can imagine, it may be interesting to use this principle to avoid manually specifying the value of a given data property in all instances of a same class, given that we know in advance that all instances share this value. As an example, consider the fact that all classes in the CARESSES CKB are derived from Topic, and therefore all instances are ultimately instances of Topic. We may decide to assign a default negative sentence *I will remember that* to the class Topic, which then will be inherited by all instances. This is perfectly reasonable, as the robot's answer *I will remember that* works everywhere, whenever the person answers to a question posed by the robot.

*Remark 1* Adding a default sentence that is inherited by a superclass do not prevent us to add a more specific sentence that is more appropriate for a given situation. For instance, consider again the instance SIT\_SOCCER, which has inherited the negative sentence *I will remember that*. We may manually add an additional negative answer *I see. Many persons like soccer; but there are more interesting sports* which works specifically for

soccer. When the system has both an inherited sentence and a manually added sentence, the latter is preferred by the system: the inherited sentence is used only if there are no specific sentences available.

*Remark 2* Inheriting values of Data property does not add additional functionalities to the system: it is only a way of building the ontology faster. The reader may ignore this possibility 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 itself.

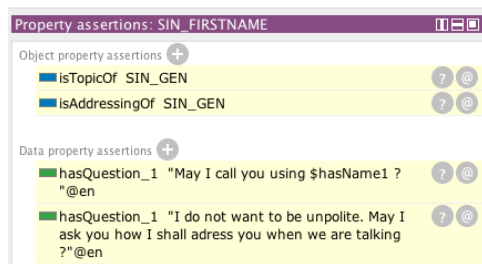


Figure 2.1: Inherited sentences of the instance SIN\_FIRSTNAME.

## 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 works in very limited case. In fact, it is very hard to find a sentence that works for all subclasses and the corresponding instances. The example of the negative sentence *I will remember that* is very generic, and we would like the robot to be much more specific. Think about the question *Shall I address you ??* that the robot may ask to understand the way in which the person prefers to be addressed. This question shall appear in many different instances, e.g.; *Shall I address you with your first name?* or *Shall I address you with your last name?* or *Shall I address you as auntie?*. As the reader may observe, the sentence is composed of two parts:

- A invariant part
- A variable part

In the example above, the invariant part is *Shall I address you ??* whereas the variant part is, from case to case, *with your first name*, *with your last name*, *as auntie*. To deal with this situation, the variant part is encoded in a new Data Property that has not been introduced before, *hasName*. All instances shall have the Data Property *hasName* filled with a *chunk of sentence*. According to this rationale, the instance *SIN\_FIRSTNAME* can then be assigned the value *with your first name*; the instance *SIN\_LASTNAME* can be assigned the value *with your last name*; the instance *SIN\_AUNTIE* can be assigned the value *as auntie*.

Once we have defined the variable part *hasName*, it is possible for the system to automatically compose sentences. Suppose that the instance *SIN\_FIRSTNAME* is assigned

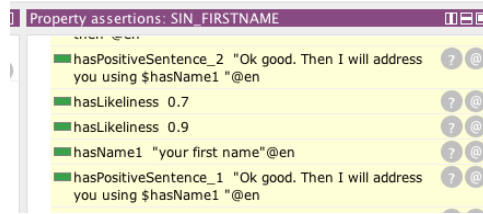


Figure 2.2: Inherited sentences of the instance SIN\_FIRSTNAME and Data Property has-Name

the following question for the Data Property hasQuestion: *Shall I address you \$hasName?*. The expression *\$hasName*, thanks to the symbol \$, is interpreted by the system as a variable: during chit-chatting, the robot will automatically substitute the variable *\$hasName* with the corresponding value of the hasName Data Property, thus finally yielding the question *Shall I address you with your first name?*

Now think how powerful this mechanism is. SIN\_FIRSTNAME is an instance of the class FirstName, which is derived from the superclass Addressing. Similarly, 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 also that SIN\_LASTNAME has been assigned the value *with 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, it is now sufficient to 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 chit-chatting - with the corresponding values of the Data Property hasName.

**Remark 1** The principle adopted is a little more complex, as the same instance may have different values of the Data Property hasName to be adopted in different contexts; this additional feature will be partially shown through experiments, but it is not necessary to understand the basic functioning of the system. The general idea consists in using different Data Properties hasName1, hasName2, hasName3, using in each situation the more appropriate property. As an example, instances of the class Dog may have *A dog* as value of the Data Property hasName1, *the dog* as value of the Data Property hasName2, *Your dog* as value of the Data Property hasName3. Data Properties hasName1, hasName2 and hasName3 are all subproperties of hasName

**Remark 2** All Data Properties that are subproperties of hasName are organized in a hierarchical way, i.e. the Data Property hasName1 will have the subproperty hasName1\_0, that in turn will have the 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 the instances. However, instances will inherit all properties of the superclasses: for example, instances of the class Fridge may inherit the Data Property hasName1 *a fridge* by their class, but they may also inherit the Data Property hasName1 *an appliance* by the class Home Appliance, superclass of Fridge. While this is correct

from an ontological point of view (i.e. a fridge is an appliance) it may lead to a ambiguities for the creation of automatic 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 Ontology hierarchy. The system will always consider the Data Property with the higher index when creating automatic generated sentences. In the same way, and with the same rationale, also properties related to sentences are organized in the same way.

*Remark 3* Be careful while applying the inheritance mechanism: if we assign that question *Shall I address you \$hasName1 ?* to the superclass **Addressing**, we have to be sure that instances of the class **Addressing** does not have a **hasName1** property that may lead to non-sense sentences (i.e., *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 the Data Properties **hasQuestion**, **hasPositiveSentence** and **hasNegativeSentence** to the class **Name**;
4. Add additionally Data Properties **hasQuestion\_0** and **hasPositiveSentence\_0** to the class **Name**. using the variable **hasName1**;
5. Add the Data Property **hasName1** in all subclasses of **Name**;
6. Add the class **MrsLastName** with culture-specific and user-specific instances. In the user-specific instance, add the Data Property **hasName1\_0** and eventually the likeliness. Connect the culture-generic instance with the user-specific one by 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 chitchatting textual software;
9. Write a sentence and talk with the virtual Pepper robot by using the terminal.

The Exercise shows the possibility of talking with Pepper by using sentences that are inherited by the class and by exploiting the possibility of the Data Property **hasName**, both for culture-specific and user-specific instances.

The exercise may be extended by working with the Class **Pizza** created before, and adding more inherited Question, Positive, PositiveAndWait and Negative Sentences.

## 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 relevant for chit-chatting purposes: `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 that the Question corresponding to the Data Property `hasQuestion` has been asked by the robot and a positive feedback has been received by the user.

#### 3.1.1 `hasQuestionContextual`

While the data property `hasQuestion` is expected to contain a question that the robot may ask in order to explore the general attitude of the person concerning the corresponding concept, the data property `hasQuestionContextual` poses specific questions related to the corresponding concept. In other words, while the data property `hasQuestion` refers to *usually* or *sometimes*, the `hasQuestionContextual` is always referred to *now*. As an example, the instance `SIN_SOCCER_WATCHINGSPORT` can have *Do you usually watch sport on TV?* as Data Property `hasQuestion` and *Do you want to watch soccer on TV now?* as `hasQuestionContextual`.

### 3.1.2 hasQuestionGoal

This data property has the same meaning of `hasQuestionContextual`, but additionally can be linked to a Goal that the robot may achieve if the feedback of the user is positive. In the example before, *Do you want to watch soccer on TV now?* could be a property of type `hasQuestionGoal` and in this case the Dialogue Manager will generate a goal for the planner (i.e. switching on TV or accompany the user to the TV).

## 3.2 hasKeyword1 and hasKeyword2

These two Data Properties contains the keywords that may trigger a particular dialogue between the robot and the user. Indeed, the chit-chat 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 some help, and it waits for the user to talk. When a sentence is detected, the robot check if it contains keywords that are connected to some goals (e.g., *play music*, or *show weather*) and in a second steps check if it contains keywords related to conversation topics. These keywords are indeed described with the `hasKeyword1` and `hasKeyword2` data properties.

*Remark 1* Keywords usually may be assigned to classes and then inherited by the instances.

*Remark 2* In case of multiple instances triggered by the same keywords (this is very likely to happen when keywords are assigned to classes), the one with the highest likeliness is chosen for the dialogue. With the same rationale, if no keywords are detected, the robot starts talking about the topic of conversation 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` and a keyword contained in `hasKeyword2`.

*Remark 3* Instances and classes may have more than one `hasKeyword1` and `hasKeyword2` Data Properties. 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 previous exercise;
3. Add person-specific likeliness about Pizzas;
4. Add person-specific attitudes towards ways of addressing and person-specific sentences;
5. Add properties `QuestionContextual` and `QuestionGoal`
6. Add triggering keywords
7. Test the modified Ontology

## CHAPTER 4

---

# TOPIC TREE AND ONTOLOGY HIERARCHY

---

### 4.1 Topic Tree

In the previous chapters we analysed the structure of the CARESSES ontology by assuming that the hierarchy of the dialogue will always follow the hierarchy of the ontology, as you have seen in the exercises related to the classes *Addressing* and *Pizza*. However, this is not true in general. Instances belonging to classes that are not in an ontological hierarchical structure may however be hierarchical from the point of view of the dialogue. As an example, the robot may want to ask if the user has biscuits for breakfast. It is obvious that it should at first ask if the user has breakfast in the morning. Thus, there can be an instance *SIN\_HAVINGBREAKFAST*, of the class *HavingBreakfast* that is directly connected to the generic Indian user (*SIN\_GEN*). The class *HavingBreakfast* will be probably a subclass of the class *DailyRoutine* and more generally of the class *Habit*. Of course the instance related to the attitude of the user of having biscuits during breakfast may not be a subclass of *HavingBreakfast* because it is related to the attitude of the user towards an object, not an habit. Therefore it will be an instance of the class *Biscuit*, a subclass of *Food* and more generally speaking 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 again used. 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 attitude of the user of having biscuits during breakfast. In principle different instances of biscuits (e.g., `SIN_BISCUIT_HAVINGLUNCH`, `SIN_BISCUIT_BUYING`, ...) may be present in the same class.

*Remark 2* The `hasTopic` construction and the Ontology hierarchy will be both used when creating the Topic Tree. For example, to express the habits of the Indian user of having 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`. On 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 user, the robot will first ask if the user usually has lunch, then it will investigate the user's attitude towards italian food during lunch and (in case of positive feedback) in the end it will ask if the user sometimes eat pizzas during lunch.

*Remark 3* It is possible to build automatic 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`, `DailyRoutine`, `HavingMeal`, `HavingLunch`;
4. Create the Classes `Pizza`, `PizzaMargherita`, `PizzaPepperoni`;
5. Create a culture-generic instance of the class `HavingLunch`, and add the corresponding likeliness;
6. Create culture-generic instances related to the attitude of the person to have pizza at lunch (i.e. `SIN_PIZZA_HAVINGLUNCH`, `SIN_PIZZAMARGHERITA_HAVINGLUNCH`, etc. and add the corresponding likeliness;
7. Link the instance of the class `HavingLunch` with instances of the class `Pizza` (and its subclasses).
8. Link the `SIN_GEN` with the instance of the class `HavingLunch` (`hasTopic`).
9. Add Question, Positive and Negative sentences in all the new instances.
10. Save and test the modified Ontology

### 4.3 Exercise 6

1. Open Protégé;
2. Load the Environment.owl ontology;
3. Create some user-specific instances with `hasLikeliness` equal to 1
4. Save and test the modified Ontology

## CHAPTER 5

---

# OBJECT PROPERTY HASCONDITION

---

This final chapter describes an additional Object Property relevant for chit-chatting purposes: `hasCondition`.

### 5.1 `hasCondition`

Until now, the robot has the possibility to choose about all conversation topics described in the ontology, using the ontology structure, the `hasTopic` properties, likeliness and triggering keywords in order to choose the best conversation topic for the user. However, it can be quite obvious that there are some topics that could be weird in certain moment of the day (e.g. talking about breakfast at 5 p.m.), while on the other side there are some situations in which some topics should be absolutely chosen by the robot (e.g. talking about way of addressing during the first encounter).

In order to consider these two situations, two additional Object Properties have been added, both subproperties of `hasCondition`: `hasNecessaryCondition` and `hasTriggeringCondition`.

#### 5.1.1 `hasNecessaryCondition`

This Object Property links a generic topic of conversation with a condition that should be absolutely verified or the topic of conversation will be never chosen by the robot. Conditions can be related to time, location, or general situations. In the example before, the

instance `SIN_HAVINGBREAKFAST` will be linked to the instance `SIN_MORNING_HAVINGBREAKFAST` (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 topic of conversation. In the example before, the instance `SIN_NAME` will be linked to the instance `SIN_FIRSTROBOTENCOUNTER_NAME` (of the Class `Event`) by an Object Property of type `hasTriggeringCondition`.

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

## 5.2 Exercise 7

1. Open Protégé;
2. Load the ontology modified in the Exercise 5;
3. Add a triggering condition of the topic *HavingLunch*: link the instance `SIN_HAVINGLUNCH` with a certain condition (i.e. `SIN_1PM_HAVINGLUNCH`) with the ObjectProperty `hasTriggeringCondition`. In the condition instance, add the DataProperty `hasValue`, with value equal to 1pm.
4. With the same procedure, add now a necessary condition (i.e. location) of the class *HavingLunch* (it will be interpreted as AND);
5. Test the modified Ontology.

Now you are ready to work on the complete CARESSES Ontology. Try to add properties, instances and eventually classes and test the result with the attached software.

## Appendix

### A.1 General Steps for creating a new dialogue topic (culture-generic)

1. Select the class to which the topic belong, or eventually create a new class;
2. Add the instance (You may 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 eventually to a different instance;
5. Save the Ontology.

### A.2 General Steps for creating a new user-specific information

1. Add the user specific instance (You may 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-generic instance;
4. Save the Ontology.