Marie Curie Individual Fellowship
FP7 Program

Report 2:

Implementation

Produced by:	Raquel Ros Espinoza

Grant agreement Number:	220368
Project title:	Adapting Robot Behavior based on Interaction
Project acronym:	ARBI
Project starting date:	01 August 2008
Project ending date:	31 July 2008
Project coordinator:	Dr. Rachid Alami

Toulouse, August 2010.

In this document we report on the implementation of the concepts described in report 1 (Concepts and Research Methodology). First we will describe the knowledge representation of the robot based on an ontology approach. Next, we detail the geometric reasoning engine which will in turn feed the symbolic knowledge of the robot. Finally, we present the dialog module in charge of translating written natural language to robot's knowledge.

1 Knowledge Representation

We believe that the knowledge model of a robot should include a comprehensive model of the roles, relationships and context of objects in the environment, as well as beliefs and intentions of other agents. Moreover, this understanding must rely on a formal encoding that requires high expressivity while remaining well suited for machine processing in order to be used by the robot.

1.1 The OpenRobots Ontology

We propose the use of ORO (the "OpenRobots Ontology" server[1]), a central knowledge repository that stores, manages, processes and exposes knowledge for the robot from a symbolic point of view. It internally relies on RDF-derivate OWL Description Logics[2] to formally represent statements on the world as triples `<subject> <predicate> <object>`. For example, to represent that Maria is a human we would include the statement `Raquel rdf:type Human`. It uses two open-source libraries: `Jena`[3] for storage and manipulation of statements and `Pellet`[4] first-order logic reasoner to classify, apply rules and compute inferences on the knowledge base [2].

ORO defines an initial *upper* ontology for human-aware robotics called *OpenRobots Commonsense Ontology*[5]. This initial ontology contains a set of concepts, relationships between concepts and rules and defines the "cultural background" of the robot, i.e. the a priori known concepts. Currently, this commonsense knowledge is focused on the requirement of human-robot interactions in everyday environments, but contains as well generic concepts like `thing`, `object`, `location` and relationships between those. The common-sense ontology design relies heavily on the standard OpenCyc[6] upper ontology for the concepts naming, thus ensuring a good compatibility with other knowledge bases. Figure 1 illustrates a simple example with some concepts.

[1] Project homepage: http://homepages.laas.fr/slemaign/oro-server
[2] http://www.w3.org/TR/owl2-overview
[3] http://jena.sourceforge.net
[4] http://clarkparsia.com/pellet
[5] http://homepages.laas.fr/slemaign/oro-server/oro-ontology.html
[6] http://www.opencyc.org

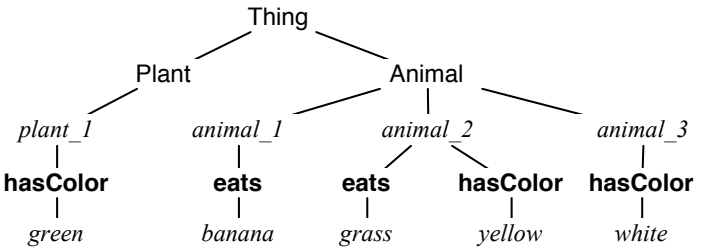


Figure 1: Ontology example. Names with first capital letter correspond to classes; bold names, to properties; and italic names, to instances.

1.2 Finding discriminants

In daily human interactions, where people refer to objects ("Look at the bike"), sometimes the utterance does not contain sufficient information to be understood correctly. That is, ambiguities concerning the referent can occur ("Which of the two bikes visible to me does she mean?"). To establish an efficient exchange of information and thus communicate meaning, these ambiguities have to be resolved. Humans employ several basic strategies in order to clarify such ambiguities, and they do so efficiently and smoothly. First, by applying internal cognitive strategies (such as visual perspective taking, detailed in Section 2.1); and only later, when those proved unsuccessful, verbal inquiries come into play, i.e. asking for additional information in order to clarify the ambiguity.

In the latter case, verbal inquiries, the main question is: what to ask? It is fundamental to provide the robot with a strategy that efficiently searches the "right" question to ask. For instance, in the example above, if the two bikes have different color, querying for this attribute is efficient since the answer will immediately solve the ambiguity.

We have implemented a set of semantic categorization functions in ORO. One of them consists in looking for discriminants, i.e. descriptors that allow a maximum discrimination among a set of individuals. As we describe later on, this functionality is used to ground the referent during interaction.

We distinguish two types of discriminants. *Complete* discriminants are those attributes (or properties) that totally discriminate the set of individuals. In other words, properties whose values can uniquely identify those individuals. However, they are not always available. First, because two or more individuals may share the same value, and second, because not all individuals may share the same properties. Thus, *partial* discriminants are those that "better" split the set of individuals in different subsets based on some criteria.

The algorithm to determine the type of discriminant available (Algorithm 1) has the following steps (to better follow it, we base its description on the ontology example illustrated in Figure 1. We search a discriminant for the following individuals: `plant_1, animal_1, animal_2` and `animal_3`). First we obtain the direct properties for all the individuals, i.e. we do not consider all the hierarchy of properties (line 1). In the example, `plant_1` has two superclasses (`plant` and

`thing`), but we only take the most direct one (the class `plant`). Next, we compute the number of individuals per property (line 4) and the number of different values for that property (line 5). If there is more than one different value for the property (in other words, if not all individuals have the same value), then we consider that property as a potential discriminant (lines 6 and 7). Finally, we sort the list of potential properties following two criteria: the number of individual occurrences (i.e. the most individuals are covered by that property, the better) and the values occurrences (i.e. the more distinct values, the better). The best discriminant corresponds to the first element of the sorted list. In other words, the class with higher number of occurrences and more variety in it. If several properties are equal, return all of them.

In our example, the algorithm would return the class name as the partial discriminant. If we only consider the instances of the class `Animal`, it would return two properties equally discriminant: {`hasColor`,`eats`}. It should be noted that this way of proceeding does not respect the open world assumption. We believe that the robot should only reason bases on his current knowledge.

Algorithm 1 get_discriminant($individuals$)

```
1:  P ← get_properties(individuals)
2:  P̂ ← nil
3:  for all p ∈ P do
4:     n_ind ← nb_ind_with_prop(p)
5:     n_val ← nb_diff_values(p)
6:     if n_val > 1 then
7:        P̂ ← append([p, n_ind, n_val])
8:     end if
9:  end for
10: sort(P̂)
11: return first(first(P̂))
```

1.3 Modeling agent's beliefs

ORO implements separate cognitive models for each agent it knows. Thus, there is at least one cognitive model for the robot itself, and n additional models, one per agent. When the robot interacts with a new agent, a separate RDF triple storage is created to store the robot's knowledge about the agent's perception. For instance, in the case of perspective taking (Section 2.1), we compute the visibility and spatial information about the world from each agent point of view, and store it in their own cognitive models. Having separate cognitive models allows us to store and reason on different models of the world. Thus, allowing us to easily attribute false beliefs to other agents.

1.4 Feeding the ontology

Objects have features (like color, size, shape, texture, etc.) that allow us to distinguish one from another. Besides, we can also categorize objects in different classes and refer to their class as a descriptor. For example, a glass is an object that can be classified based on its purpose in different ways, such as a beer glass, a wine glass, a champagne glass, and others. However, the wine glass can also be subdivided in two categories, white wine glass and red wine glass. Hence, in a scenario with three glasses (a champagne glass, a white wine glass and a red wine glass), simply asking for "the glass" would bring out ambiguities. Asking for "the wine glass", still would produce confusion. The only unambiguous feature description would be asking for "the red wine glass" instead.

In the current approach, the robot cannot perceive these type of features by itself (due to limitations in perception, which is not the focus of our work). Thus, we have to explicitly inform them to the robot. So far, this information is loaded into the ontology during initialization. However, as we explain in the next section, all the information coming from geometric reasoning is automatically computed and sent to the ontology.

2 Geometric Reasoning

This section describes different reasoning mechanisms to provide an abstraction layer to the decisional layer on top of the geometrical description of the environment.

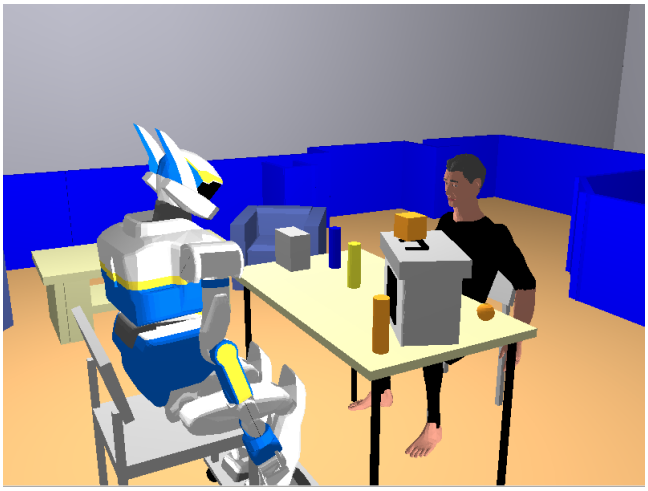
To model the environment we use the software platform Move3D [4]. The kinematic structures of the human and the robot, as well as their positions and objects' positions are integrated into this platform to maintain a coherent model of the real environment. It also allows us to view the visual perspective of the agents in the world by modeling their visual sensors (eyes for humans, cameras for robots). Figure 2 illustrates an example environment with a person, a robot and multiple objects.

The next subsections describe how we compute the different basic skills presented in Report 1. All this information is stored in the ontology, which in turn may infer additional information as we explain next. Moreover, the information concerning specific agents, i.e. perspective taking descriptors, is stored in each agent's cognitive kernel in ORO (Section 1.3) allowing the decisional level to reason about each agents' beliefs about the world. This way we have a centralized source of knowledge, where different components may update or gather the information as required.

2.1 Perspective Taking

We next describe three skill for perspective taking: visibility, reachability and spatial. Each skill is computed for each object (or agent) in the environment with respect to each agent. This information is stored in each agent's cognitive model (Section 1.3).

(a)

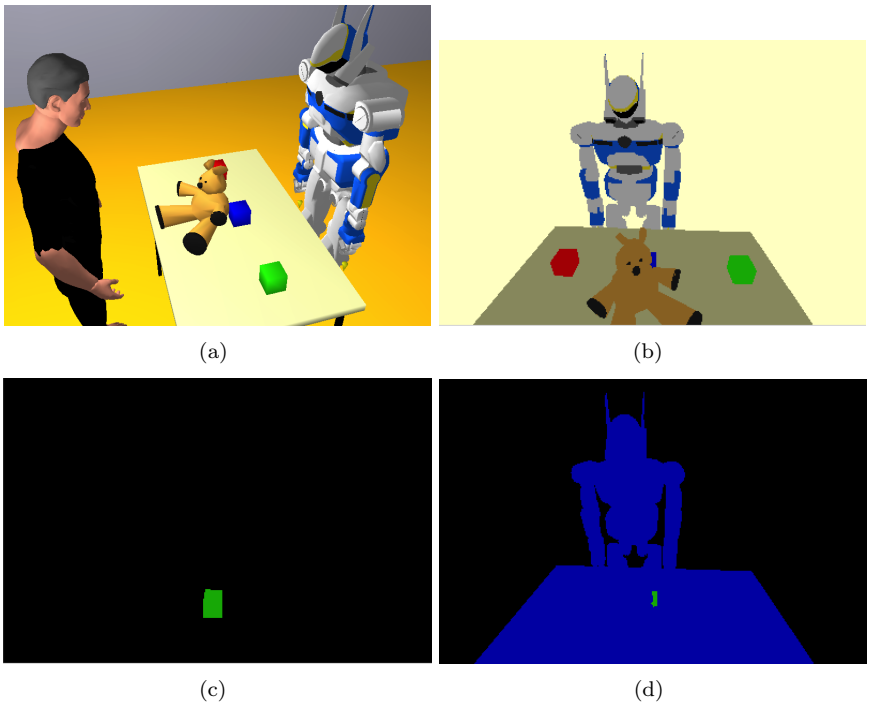


(b)

Figure 2: Scenario: (a) real environment and (b) 3D geometric model of the environment.

2.1.1 Visibility

In [3] we present a model-based approach for implementing visual perspective taking abilities. In this approach, 2D perspective projections of the 3D environment (Figure 3a,b) is used to determine if an object is visible to an agent. We first obtain the projection of the isolated object (Figure 3c, the blue box), and we compare it with the "real" projection of the scene which considers occlusions of the evaluated object (Figure 3d, the teddy bear is partially occluding



(a) (b)

(c) (d)

Figure 3: (a) An example of the environment, (b) human visual perspective, (c) free relative projection and (d) visible relative projection.

the blue box). A visibility ratio of the object is then computed by comparing both images. An object is visible to an agent if the ratio is over a given threshold.

In order to obtain a visual perspective, the actual visibility alone is not enough. We believe that visual perspective taking ability is not restricted to what the other person is seeing in a given moment, but also what he "can" see with a minimal effort (moving the eyes or the head). To model the potential visibility of an object we compute the visibility ratio while turning the head of the agent model towards the object.

Moreover, to enrich the visual perspective model and reason on the human's focus of attention, the placement of the object respect to the human's vision is also computed. According to human's gaze direction and object's position, we compute whether the object is within the human's focus of attention (FOA), field of view (FOV) or out of field of view (OOF). Figure 4 illustrates the different visibility regions around the agent.

Thus, the following information is sent to the ontology for each agent:

if $object_i$ can be seen (directly or moving the head)
$\rightarrow$ $object_i$ `isVisible true`
otherwise, $object_i$ `isVisible false`

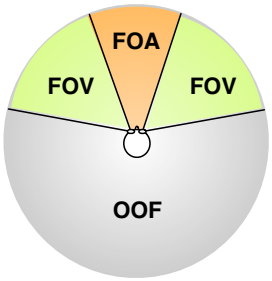
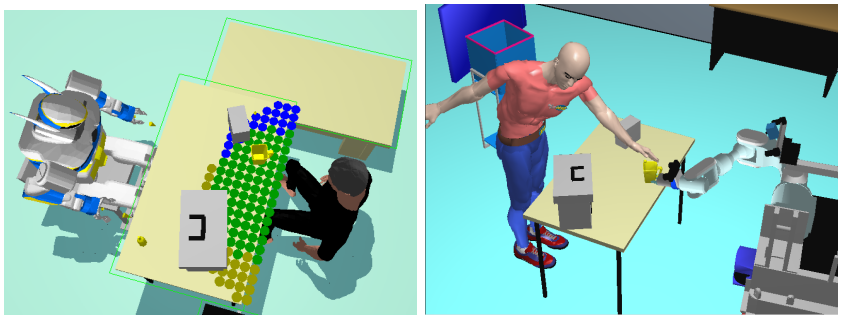


Figure 4: Object visibility placements around the agent.



(a) (b)

Figure 5: (a) Reachable points from the human perspective when bending: yellow, blue and green points correspond to left hand, right hand and both hands respectively. (b) Human and robot posture for reaching the cup.

2.1.2 Reachability

This ability allows the robot to estimate the agent's capacity to reach an object, which is fundamental for task planning. For example, if the human asks the robot to give her an object, the robot must compute a transfer point where the human will be able to get the object.

We say that an object or a region is reachable if there is a collision free posture for the agent where the end-effector is at the center of the object or region with a given tolerance. A valid posture includes moving the upper-body or standing, if possible. Figure 5 illustrates the reasoning results for reaching regions and an object.

The information sent to the ontology is the following (for each agent):

if $object_i$ can be reached (directly or by moving the body)
$\rightarrow$ $object_i$ `isReachable true`
otherwise, $object_i$ `isReachable false`

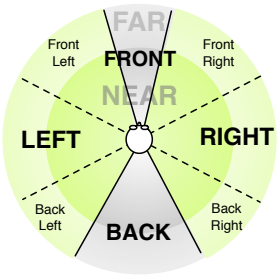


Figure 6: Relative placements around the agent through space discretization.

2.1.3 Spatial

In this work, we use two types of the frames of reference: egocentric (from the robot perspective) and addressee-centered (from the human perspective). Thus, given an object and the referent we divide the space around the referent into four regions: front, left, right and back. The number of these regions are doubled with the distinction of near and far from the referent in the center. These regions are separated by arbitrary angle values relative to the referent orientation. Depending of the task the number of regions can be increased to 16 to include a more precise spatial placement information (e.g. "near front right", "far back left"). Figure 6 illustrates an example.

2.2 Gaze following and Focusing Attention

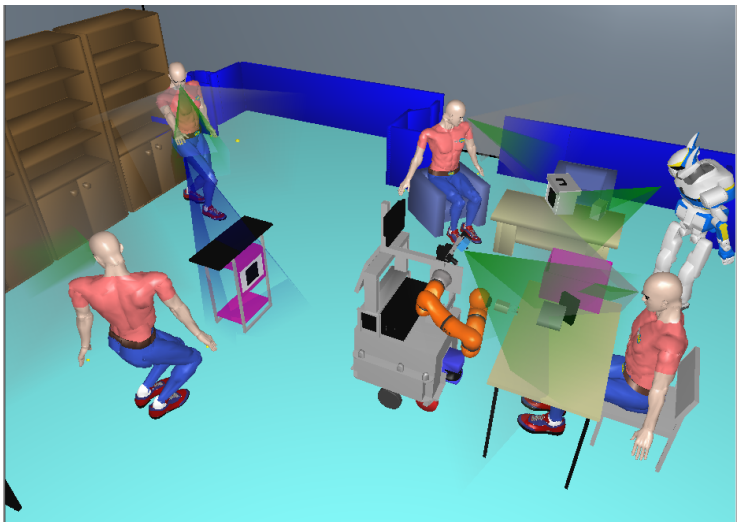
Gaze following is based on the head (for humans) and the cameras (for robots) orientation. We use motion capture to obtain a precise orientation of the human head, and then, based on the human model, we can compute where she/he is looking at. In the previous section we have described how visibility is computed. Since we are interested in knowing if an agent is actually looking or not to an object (instead of knowing if an object is potentially visible when turning its head or torso), we apply the following reasoning:

$$\text{if } object_i \text{ is within FOA } \wedge object_i \texttt{ isVisible } \text{true} \\ \rightarrow agent_i \texttt{ looksAt } object_i$$

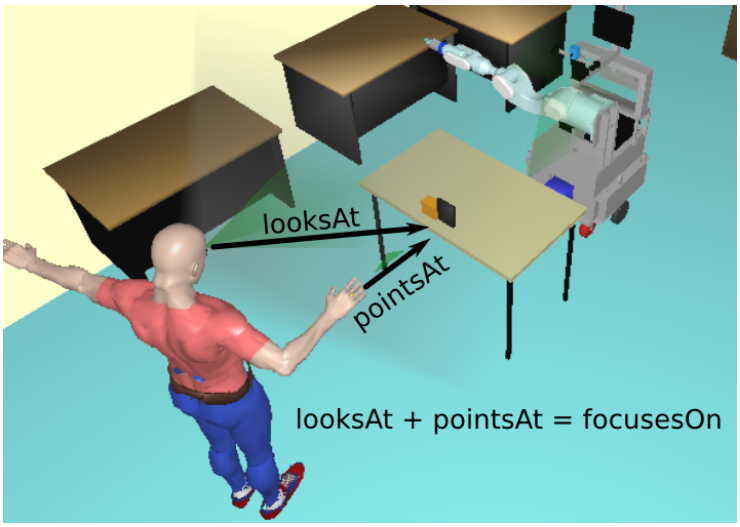
Pointing is computed in a similar way to visibility. The idea is to place the camera at the fingertips of the agent (in the case of the human). Next, we can apply the same mechanism used for computing visibility, but reducing the angle of FOV. This way we obtain a cone containing the set of objects pointed by the agent.

Figure 7 shows different configurations showing a cone in green for both, gaze and pointing. All objects that fall into this region are considered as either being looked at or being pointed at by the agent. The information sent to the ontology is the following:

$$agent_i \texttt{ pointsAt } object_j$$



(a)



(b)

Figure 7: (a) Different agents pointing and looking at objects in the environment. (b) Computation of focuses on.

Based on this information it is easy to infer in ORO through rules the following information:

- focus of attention of an agent:

$$\text{if } agent_i \texttt{ pointsAt } object_k \wedge agent_i \texttt{ looksAt } object_k$$

$$\rightarrow agent_i \texttt{ focusesAt } object_k$$

- joint attention for two or more agents:

$$\text{if } agent_i \texttt{ focusOn } object_k \rightarrow obj_k \texttt{ isFocusOf } agent_i$$
$$\Rightarrow |\{agent_i | obj_k \texttt{ isFocusOf } agent_i\}| > 1$$

2.3 Symbolic Location Descriptors

Symbolic location descriptors allow the robot to compute spatial relations between objects in the environment. The system infers symbolic relations between objects from its 3D geometric world representation. In this work we propose the use of three basic symbolic relations between each pair of objects. However, their inverse relations can be automatically computed at the symbolic level, i.e. through inference based on OpenRobots Commonsense Ontology, enlarging the symbolic descriptions knowledge easily.

- *IsIn*: indicates if an object (or an agent) is inside of another object. Its inverse relation corresponds to *Contains*. Ex. `Bottle IsIn TrashBin`. Its inverse relation corresponds to `TrashBin Contains Bottle`.
- *IsOn*: indicates if an object (or an agent) is placed on top of another object. Its inverse relation is *IsUnder*[7]. Ex. `Red-box IsOn Blue-box`. Its inverse relation corresponds to `Blue-box IsUnder Red-box`.
- *IsNextTo*: tests if an object (or an agent) is next to another object. It has no inverse relation, but symmetric. Ex. `Bottle IsNextTo Cup`. There is no inverse relation, but symmetric, i.e. `Cup IsNextTo Bottle`.

3 Dialog

In this section we describe the dialog module we have developed in order to process natural language utterances. Our goal is to allow the user to communicate with the robot in a more natural way, instead of using a very limited set of utterances. However, our system is still very limited and we mainly focused to solve the tasks we propose in the game scenarios presented in Report 3. Thus, given the user input, it grounds the concepts based on the robot's knowledge and eventually, translates the discourse into a set of declarative RDF statements which are sent to the robot's knowledge, i.e. to ORO.

Figure 8 depicts the components of our dialog system. We have three main components:

1. *parsing*: receives the natural language utterances and performs a syntax analysis. Its output consists on a formal sentence class which is the basic element to be used through the overall module.

[7] We consider that there is a physical contact between both objects, although the English definition of under does not necessarily imply it.

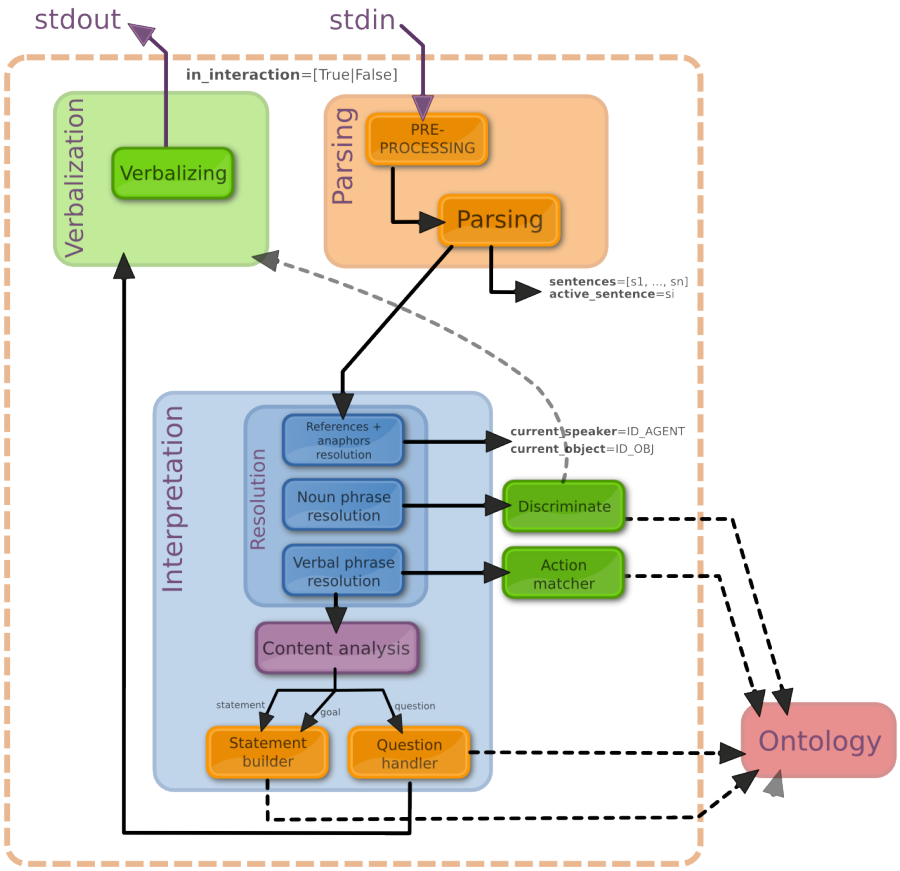


Figure 8: Dialog system.

2. *interpretation*: analyses the concepts in the sentence to align them according to the robot's current knowledge and transforms the sentence into RDF statements.

3. *verbalization*: performs the inverse translation, i.e. translate a class sentence into natural language.

We next describe in more detail the Interpretation module since it is the one of most interest for the project.

3.1 Interpretation

This module is composed of three main components: sentence resolution, content analysis and statement builder.

The sentence resolution consists of the following three steps:

- pronouns (me, you, it) are replaced with speakers' and objects' IDs.
- noun phrase resolution consists in grounding the concepts referred to (see Section 4 for details).
- verbal resolution refers to grounding the verbs and associating the parameters (*thematic roles*) to each.

  Thematic roles allow to semantically link a verb to its complements. There is no general agreement amongst linguists on a comprehensive list of thematic roles. Therefore, the granularity of the roles largely varies in the literature[1]. In this work we only use a small set of roles, which is sufficient enough to represent the verbs we employ.

  We store the list of verbs associated to their thematic roles in a file, along with an optional set of synonyms (helpful to enrich the vocabulary of the robot). The thematic role is composed of: `theme`, `agent` and `recipient/receiver`.

Let us take a look at an example to better understand the process. Suppose that the user asks the robot Jido the next task:

"*Jido, get me the box on the table.*"

The sentence would be transformed at each step as follows:

1. replacing pronoun *me*:

   get **human_01** *the box on the table.*

2. resolving the object id for *box on the table*:

   get human_01 **object_123**.

3. resolving the verb *get* (searching for synonyms and retrieving its thematic role):

   give human_01 object_123.

Once the sentence has be aligned to the robot's knowledge, we are able to analyze its meaning. In other words, what is the user asking for, is it a question?, is it an order?, is it providing information? The *content analysis* is in charge of this task. Based on this information, the system will either answer to the human (after querying the ontology for the response), or will include the new information (statements or goals) into the robot's knowledge.

In any case, the grounded sentence must be finally transformed into ORO predicates. The *statement builder* performs this translation. Thus, from the above example we would obtain the following predicates (i.e. a new goal for the robot):

```
human_01 desires situation_a3f74
situation_a3f74 rdf:type Give
situation_a3f74 performedBy myself
situation_a3f74 actsOnObject object_123
situation_a3f74 receivedBy human_01
```

which can be read as: `human_01` (the user) desires a situation (`situation_a3f74`), where the situation is described with the thematic role `Give`, performed by `myself` (the robot), acting on the the `object_123` and being recieved by `human_01`. This set of statements is sent to the robot's knowledge (ORO) and the supervision is now in charge of executing the action.

At any moment during the interaction, the robot may need to query the user for additional information in order to continue the grounding process. In this case, a history of the dialog must be stored to follow the discourse of the dialog. If at some point, it fails, it will inform the human, and the history of dialog will be removed to start the process from scratch all over again (in order to avoid more confusion or ambiguaty in the discourse).

4 Clarification Algorithm

The ontology is first initialized with the description of the environment represented by object features as defined in Section 1.4 which is considered the robot's initial knowledge about the world (along with the common sense concepts). During interaction, the robot's knowledge is updated with the incoming information from the geometric reasoning, i.e. visual perspective taking, spatial perspective taking and symbolic locations descriptors. Based on all this information, and a given partial (or complete) description of an object (list of attribute-value pairs), the robot is able to identify the referred object the following way (Algorithm 2). First it obtains all objects that fulfill the initial description. Based on the result it either succeeds (obtains one single object), fails (no object with that description could be found) or obtains several objects. In this latter case, a new descriptor is added to the initial description and the process starts over again. Failure occur when the description does not match any object from the robot's knowledge. Either because the robot's knowledge is incomplete (the human refers to an unknown descriptor or descriptor value) or due to inconsistent information (human's and robot's beliefs differ).

Let us take a look at an example to better understand the overall process. Suppose there are two bottles on a table, b_1, a red glass bottle and b_2, a green plastic bottle. The human asks the robot for a bottle: "Give me the bottle". Thus, the initial description corresponds to $[(\texttt{type}, \texttt{bottle})]$. Since both objects fulfill this description, a new descriptor is required. Suppose we add the color information. In this case, the new description corresponds to $[(\texttt{type}, \texttt{bottle}), (\texttt{color}, \texttt{red})]$. The algorithm ends now indicating that the object is identified as b_1, the red glass bottle.

In order to add a new descriptor (attribute-value pair) two alternatives are available: directly asking the human for a new descriptor, or automatically

Algorithm 2 clarify(*description*)

```
1: objectL ← get_obj_with_desc(description)
2: if length(objectL) == 1 then
3:    return first(objectL)
4: else if length(objectL) == 0 then
5:    return no_object_found
6: else
7:    description ← add_descriptor(description)
8:    return clarify(description)
9: end if
```

searching a new attribute and ask the human for its value. In the latter case, we need to automatically find the best discriminant for the current list of objects being evaluated (*objectL* in the algorithm). To this end, we use the categorization functionalities provided by ORO (Section 1.2).

References

[1] C. J. Fillmore. The case for case. In Bach and Harms, editors, *Universals in Linguistic Theory*, 1968.

[2] S. Lemaignan, R. Ros, L. Mösenlechner, R. Alami, and M. Beetz. Oro, a knowledge management module for cognitive architectures in robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010. To appear.

[3] L. F. Marin-Urias, E. A. Sisbot, and R. Alami. Geometric tools for perspective taking for human-robot interaction. In *7th International Conference on Artificial Intelligence*, 2008. hri, psp.

[4] T. Siméon, J.-P. Laumond, and F. Lamiraux. Move3d: A generic platform for path planning. In *IEEE International Symposium on Assembly and Task Planning, ISATP*, pages 25–30, Fukuoka, Japan, May 2001.