Journal of Intelligent & Fuzzy Systems 36 (2019) 5105–5114
DOI:10.3233/JIFS-179055
IOS Press

Reasoning with preferences in service robots

Ivan Torres[b], Noé Hernández[a], Arturo Rodríguez[b], Gibrán Fuentes[a] and Luis A. Pineda[a,*]
[a]*Departamento de Ciencias de la Computación, Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS), Universidad Nacional Autónoma de México (UNAM), Mexico*
[b]*Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México (UNAM), Mexico*

Abstract. Service Robots should be able to reason about preferences when assisting people in common daily tasks. This functionality is useful, for instance, to respond to action directives that conflict with the user's interest or wellbeing or when commands are underspecified. Preferences are defeasible knowledge as they can change with time or context, and should be stored in a non-monotonic knowledge-base system, capable of expressing incomplete knowledge, updating defaults and exceptions dynamically, and handling multiple extensions. In this paper a knowledge-base system with such an expressive power is presented. Non-monotonicity is handled using a generalization of the Principle of Specificity, which states that in case of knowledge conflict the most specific proposition should be preferred. Reasoning about preferences is used on demand through conversational protocols that are generic and domain independent. We describe the general principles underlying such protocols and their implementation through the SitLog programming language. We also show a demonstration scenario in which the robot Golem-III assists human users using such protocols and preferences stored in its non-monotonic knowledege-base service.

Keywords: Reasoning with preferences, non-monotonic knowledge-base, robust behavior in service robots, robotics cognitive architecture, robot golem-III

1. Introduction

Service robots need to reason about user preferences in the course of service task. However, the inclusion of preferences in the knowledge-base (KB) can lead to inconsistencies. For instance, if the robot needs to go to the place where the user is in the course of a service task, and the KB includes the rules that (1) if a person is tired he is in the living room and (2) if a person is hungry then he is in the kitchen, and the robot knows that the user is tired and hungry too, it can conclude that the user is in the living room but also that he/she is in the kitchen, and a mechanism to resolve the inconsistency and move to the appropriate place is required. More generally, in this kind of situations multiple and incompatible conclusions can be derived from a KB at a given state –known as multiple extensions– and a general mechanism to deal with this kind of inconsistency needs to be defined.

Preferences involve a large and diverse type of knowledge, like the things the user likes to have or do, safety or healthy guidelines, social patterns of behavior, and more generally common sense knowledge. Preferences are defeasible knowledge and can change dynamically during the execution of a task and, in order to represent and use this kind of knowledge, a non-monotonic knowledge-base service is required.

In previous work we presented a non-monotonic knowledge-base management system for practical applications in service robots. The KB was defined as a conceptual hierarchy with inheritance that

*Corresponding author. Luis A. Pineda, Departamento de Ciencias de la Computación, Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS), Universidad Nacional Autónoma de México (UNAM), Mexico. E-mail: lpineda@unam.mx.

supports the expression of strong negation, defaults and exceptions [16]. Non-monotonicity is handled using the Principle of Specificity that states that in case of knowledge conflicts the more specific piece of knowledge has precedence or is preferred over the more general one. In this article we extend the KB's expressive power of the representational system with the association of weights to properties and relations, where the knowledge objects with lower weights are preferred over objects with higher ones. The extension also includes the expression of preference rules over classes and individuals that can be stated at any level in the hierarchy – and hence inherited to the subsumed classes. The conclusions of such rules can be thought of as conditional defaults that depend on their antecedents. These rules can also be thought of as abductive rules such that their conclusions are explained or justified by their antecedents.

Reasoning with preferences in a practical robotics task is illustrated in the storyboard in Fig. 1. In frame 1 Susan arrives home after work and is received by the service robot Golem-III. Susan tells the robot that she had a bad a day and asks it to bring her a coke and something to eat. In frame 2 the robot checks its KB and according to the current preferences – the healthy guideline stored in its KB– concludes that malz is a healthier beverage. To solve the conflict between the overt request (the coke) and the preference, the robot kindly suggests Susan to change her decision. She remembers how bad the coke is for her health and accepts the suggestion. Susan's original command included an underspecified object too (something to eat), so Golem-III chooses noodles because their expiration date is due earlier than other options. This decision is also based on the preferences defined in its KB. In frame 3 the robot visits the preferred locations where the items can be found. First, Golem-III goes to the drinks table and takes the malz; then it goes to the food table but doesn't find the noodles as depicted in frame 4. In frame 5 Golem-III checks its KB and notices that the second preferred location for food items is the snacks table, so it moves to that table and takes the noodles. Finally, in frame 6 the robot goes to the room where the user prefers to be and delivers the objects there. It also informs that the noodles were in the wrong location and asks Susan if she wants to update the preferred location for this item in its KB.

In this paper we show a framework for reasoning about preferences and multiple extensions with a non-monotonic KB in service robots. First, a review of related work is presented in Section 2. The machinery of the non-monotonic KB-System capable of handling multiple extensions is presented in Section 3. Section 4 presents the preferences management strategy and how it is used in Dialogue Models and Section 5 includes a description of the Interaction-Oriented Cognitive Architecture (IOCA) and the robot Golem-III. Finally, the conclusions are presented in Section 6.

2. Related work

There has been a large body of research about strategies to handle preferences among properties and relations in non-monotonic knowledge representation and reasoning systems. A common approach has been the definition of prioritized versions of Default Logic, which are extensions containing explicit priorities for defaults (e.g. [3, 4]). In contrast, Delgrande and Schaub [8] proposed an approach to express preferences within Reiter's Default Logic [17] using an ordered default theory. A related approach was later introduced by Delgrande et al. [9] to express preferences in logic programming under the answer set semantics. Logic Programming has also been extended to handle preferences by adding priorities to rules [5].

Both monotonic and non-monotonic knowledge representation and reasoning systems have been integrated into different service robots. A prominent instance is KnowRob [20, 21], a monotonic system based on Description Logic (DL) which has been used in different complex service robot tasks such as helping set a breakfast table [12], manipulating tools [2], performing actions cooperatively [10] and semantic mapping [22]. Answer Set Programming (ASP) is probably the most used non-monotonic system in service robotics (e.g. [6, 23]). ASP has allowed the Ke Jia's robot task planning system to handle open-knowledge rules [7] and to perform the tasks of the RoboCup@Home competition [6].

However, reasoning with preferences in service robotics has been very scarcely explored. One of the few instances is the work by Ferreti et al. [11], which uses Defeasible Logic Programming to represent the knowledge about the environment and the robot's preferences, and to reason with them for decision making. Similarly, a high-level task planner that takes into account human preferences was proposed by Alami et al. [1]. Preferences have also been incorporated in spatial reasoning systems for motion planning and manipulation [18].

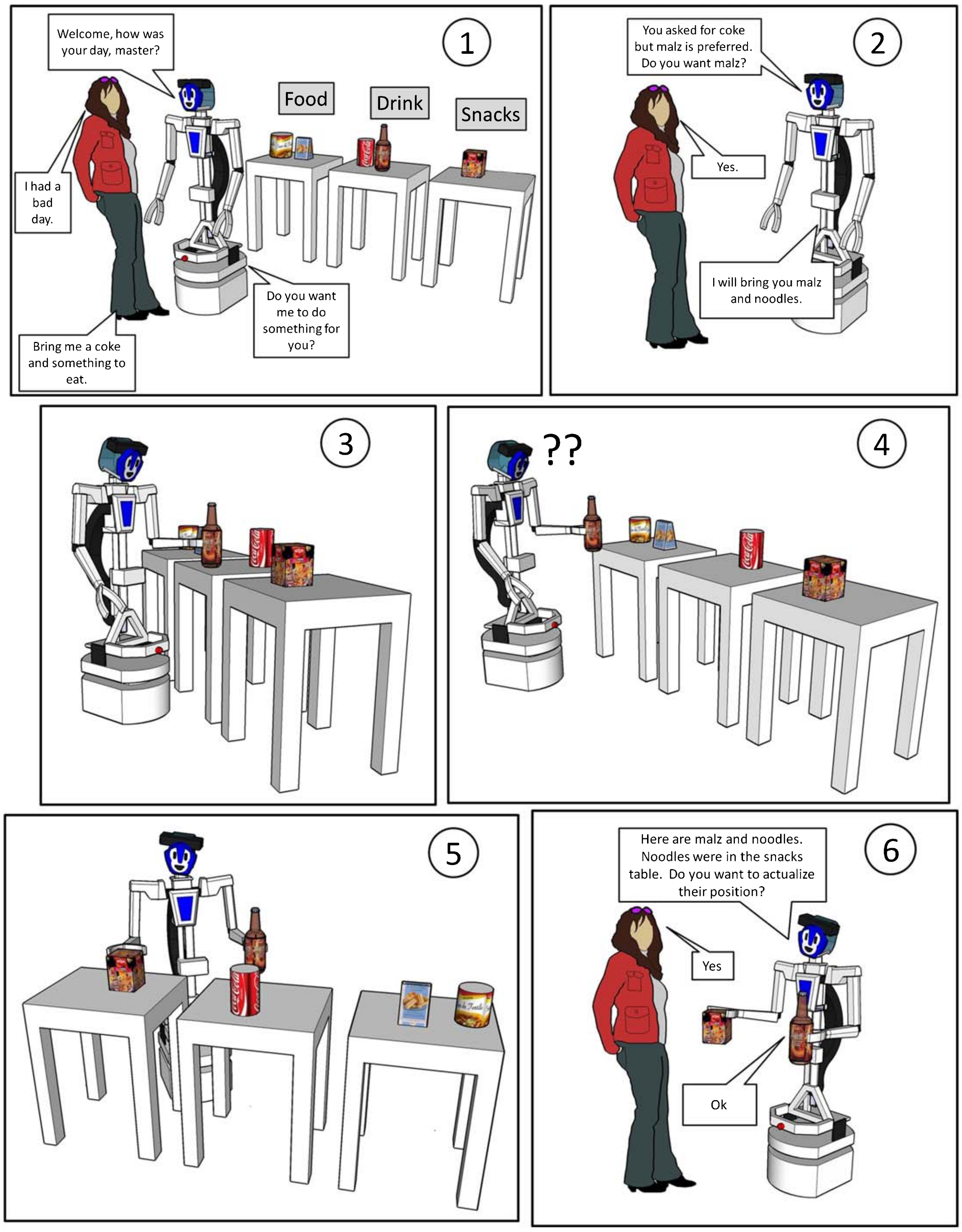
1
Welcome, how was your day, master?
Food
Drink
Snacks
I had a bad day.
Do you want me to do something for you?
Bring me a coke and something to eat.
2
You asked for coke but malz is preferred. Do you want malz?
Yes.
I will bring you malz and noodles.
3
4
??
5
6
Here are malz and noodles. Noodles were in the snacks table. Do you want to actualize their position?
Yes
Ok

Fig. 1. Storyboard of a practical task in which the service robot is reasoning with preferences.

Although in principle some non-monotonic knowledge representation and reasoning systems implemented in service robots could handle preferences, to the best of our knowledge its use has not been thoroughly studied in the context of complex tasks. In this paper, we present a non-monotonic knowledge-base system for service robots, which is able to reason with preferences among properties and relations, and demonstrate its usefulness in different practical scenarios.

3. Knowledge-base system with multiple extension

Service robots need to manipulate knowledge in their daily tasks. For this a very useful structure is the conceptual hierarchy. This type of structure allows us to express explicit knowledge and to infer its consequences. In previous work we presented a knowledge structure that represents a hierarchical partition of the domain, where each partition corresponds to a class [16]. Classes may contain more specific classes and individuals with their corresponding properties and relations. The inference mechanism of the conceptual hierarchy is inheritance, so properties and relations of the upper classes are inherited to the lower ones. In this way, an individual has his own properties and relations, plus those given to him by his class, which in turn may have inherited from a class with a higher hierarchy.

A knowledge structure can assume complete or incomplete knowledge. According to the former, the so-called *closed world assumption* (*CWA*), if something is not expressed in the KB it is considered false while in the latter case it is simply not known. The limitation of *CWA* is that the KB may not contain all the facts of the world and, although answers to queries are correct in relation to such assumption, may not be correct in the actual world, whereas the addition of an explicit negation in the case of incomplete knowledge increases the expressive power but also the computational cost. The knowledge structure presented in our previous work assumes that knowledge is incomplete, so the answer to a yes/no query can have three forms (*yes*, *no* and *unknown*), unlike the definitive answers that would be given in relation to *CWA*.

However, increasing the expressiveness power can lead to inconsistencies since there can be implicit or explicit knowledge conflicts; therefore, the KB becomes non-monotonic. In our previous work non-monotonicity is handled through the Principle of Specificity as mentioned above [19].

3.1. Specifications of preferences

In this paper, we present an extension to the KB such that preferences can be expressed. A preference is distinguished from properties and relations by its more contingent flavor. An example of concept hierarchy is illustrated in Fig. 2; here the property may be that mammals do not lay eggs while a preference, like large and carnivorous animals are dangerous, has

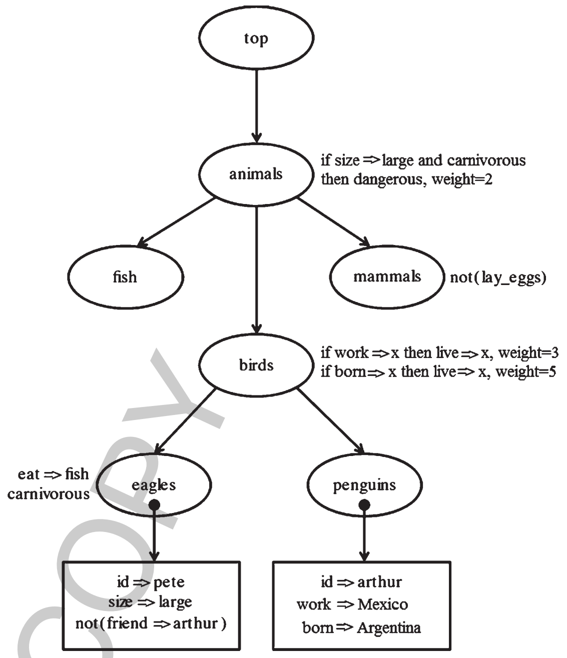

Fig. 2. Concept hierarchy of example KB.

a more contingent character. A class or individual can contain multiple preferences, both for properties and relations; therefore, there may be knowledge conflicts that the Principle of Specificity criterion cannot handle directly. In the present example the preferences are that birds live where they work and that birds live where they were born; there is also an individual in the birds class named Arthur, who was born in Argentina and works in Mexico. Hence, there is a knowledge conflict because the KB expresses that Arthur lives in Mexico and Argentina at the same time. As these are relations of the same individual, the inconsistency cannot be handle by specificity directly, and an additional mechanism is needed. For dealing with this type of conflicts, we use weights for preferences with the convention that the smaller the weight the larger the preference. For simplicity properties and relations have a weight of 0, keeping priority over the preferences. Weights are provided with the specification of the corresponding preferences. In the example the weights of 3 and 5 are assigned to the preferences that birds live where they work and birds live where they were born respectively, so the former has preference over the latter. These weights reflect the intuition that it is more plausible that one lives in the place where one works than in the place where one was born.

3.2. Reasoning with preferences

Preferences are composed of a set of antecedents and a consequent, defined in the KB with the following format: [$antecedent_1$, $antecedent_2$, ..., $antecedent_n$] =>>*consequent*. The antecedents and consequent may be properties or relations. Given the set of antecedents, the consequent must be added to satisfy the following implication:

$$antecedent_1 \cap antecedent_2 ... \cap antecedent_n \Rightarrow consequent$$

Preferences are written as follows: [*antecedent* =>> *consequent, weight*]. If there are multiple antecedents, these are specified as a list, as follows: [[$antecedent_1$, $antecedent_2$, ..., $antecedent_n$] =>> *consequent, weight*]. Antecedents and consequent values may be undefined. In this way, it is enough for an antecedent to have a property regardless of its value. For example, the antecedent *work* => *'-'* is satisfied because the individual has the property *work*=>*mexico*, but if the property were *work*=>*canada* instead, it would be satisfied anyway. Also it may be that both antecedent and consequent have undefined values as *(work=>'-')=>>(live=>'-')*; in this case the consequent takes the value of the antecedent; for example, the antecedent *work*=>*'-'* is satisfied with the property *work*=>*mexico*, so the consequent takes this value and the property *live*=>*mexico* is added.

In the example KB, the class *birds* has two preferences which are stated as: [*work*=>*'-'* =>> *live*=>*'-', 3*] and [*born*=>*'-'* =>> *live*=>*'-', 5*] which means birds live where they work and birds live where they were born respectively. The class *animals* has the preference [[*size*=>*large, carnivorous*] =>> *dangerous, 2*] that means large and carnivorous animals are dangerous. The complete KB is shown in Listing 1. Therefore, if it is queried where Arthur lives, the preference handler verifies the preferences that satisfy its antecedents –where Arthur was born and works– and adds its consequence –where does he live– to the list of properties/relations; since Arthur inherits the properties and relations of the birds class, the list of properties ordered from left to right by priority is: [*work*=>*mexico, born*=>*argentina, live*=>*mexico, live*=>*argentina*]. So the answer to the query will be that Arthur lives in Mexico since the property *live*=>*mexico* has higher priority over *live*=>*argentina*.

4. Dialogue models and preference management

Dialogue modules (DMs) are abstract communication protocols that handle the interaction between the robot and the world. They also manage the inference and knowledge resources. A DM characterizes the communication between a robot and a human by specifying expectations, actions and control information. Moreover, the conceptual knowledge of the robot, consisting of individuals with their properties, relations and preferences, is included in the robot's knowledge-base (KB). The task structure is understood as a sequence of DMs, each contributing with an independent behaviour to the final outcome of the task. In the context of the present project, DMs are defined through the Sitlog programming language [15] running in the robot Golem-III.

4.1. Dialogue model for reasoning about preferences

In this section we show a DM named *get_preferences* that specifies a linguistic protocol for managing preference knowledge. This DM is independent of the particular situation, domain of discourse and final goal. It is assumed that the robot is in front of the user and a conversation between them is viable. The DM's main structure is depicted in Fig. 3.

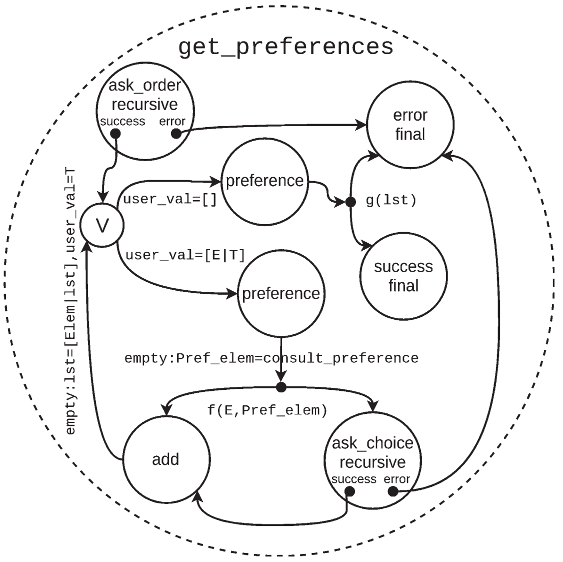

Fig. 3. Diagrammatic representation of the DM for reasoning about preferences.

```
[
 class(top,none,[],[],[]),
 class(animals,top,[[[size=>large,carnivorous]=>>dangerous,2]],[],[]),
 class(fish,animals,[],[],[]),
 class(mammals,animals,[[not(lay_eggs),0]],[],[])
 class(birds,animals,[[work=>'-'=>>live=>>'-',3],[born=>'-'=>>live=>>'-',5]],[],[])
 class(eagles,birds,[[carnivorous,0]],[[eat=>fish,0]],[
  [id=>pete,[[size=>large,0]],[[not(friend=>arthur),0]]]
  ])
 class(penguins,birds,[],[],[
  [id=>arthur,[[work=>mexico,0],[born=>argentina,0]],[]]
  ])
]
```

Listing 1. The example KB.

DMs are defined as directed graphs where nodes represent situations, and links point to the next situations. A link is traversed when its expectations are satisfied and its actions executed. In the diagram, a link may be labelled with its expectations and actions. Among such actions are increment and assignment of variables, and function applications. A situation type is either *neutral*, when a straightforward execution takes place; *final*, representing the exit point of the DM; and *recursive*, when another DM is embedded and executed recurrently. As an example of the latter type, consider a recursive situation executing the *ask* DM, a behaviour that allows the robot to ask a question to the user and get an answer. A large dot in the diagram means that a function determines the next situation. Finally, a small circle including the symbol $\vee$ represents a disjuntive situation, such that every link leaving it is labelled with the condition that must be met when such a link is traversed [15].

In the initial situation *ask_order* the robot asks the user for a command, adds the requested objects to the order list (e.g., coke and something to eat) in the variable *user_val* and moves to the next situation *preference*. This is a disjuntive situation with two instances corresponding to the base case in which *user_val* is empty, depicted in the upper output edge, and the generative case where *user_val* contains the list of requests that could be replaced by other more preferred objects (lower output edge). The function *g* examines the final list of preferred elements *lst*; if this list is empty the original order was ill-formed and the system ends in error; otherwise, the list contains the actual user preferences and the DM terminates successfully. The generative case is selected when the user's order list is not empty (i.e., *user_val=[E|T]*) where *E* is the object to be considered currently (e.g. the coke) and *T* is the list of the remaining objects

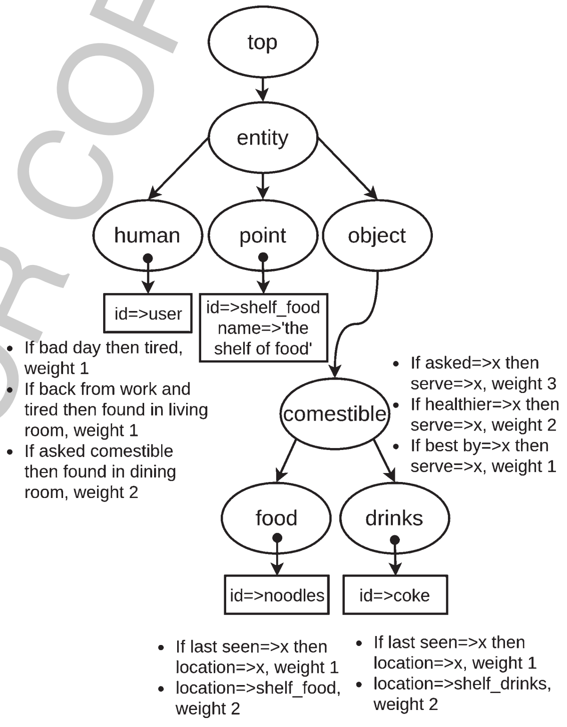
top
entity
human
point
object
id=>user
id=>shelf_food name=>'the shelf of food'
If bad day then tired, weight 1
If back from work and tired then found in living room, weight 1
If asked comestible then found in dining room, weight 2
comestible
If asked=>x then serve=>x, weight 3
If healthier=>x then serve=>x, weight 2
If best by=>x then serve=>x, weight 1
food
drinks
id=>noodles
id=>coke
If last seen=>x then location=>x, weight 1
location=>shelf_food, weight 2
If last seen=>x then location=>x, weight 1
location=>shelf_drinks, weight 2

Fig. 4. Hierarchy of the KB with preferences used in the demonstration.

of the original order (e.g. something to eat). The next situation is selected through the function *consult_preference* which queries the KB with the current object *E* and returns a preferred object *Pref_elem*. The function *f* analyzes whether *E* and *Pref_elem* are the same. If so, *E* is added to the final list (the object ordered explicitly is the preferred one); otherwise the user is asked to reconsider his/her original request and choose the preference instead (i.e., *Pref_elem*). In case the user does not care the robot chooses the most preferred object of the corresponding class. The procedure is repeated recurrently until the list *T* is empty.

4.2. Preference management in the KB

The KB is a resource that is queried on demand by DMs. Moreover, preferences are stored in the KB and retrieved by applying the function *consult_preference*.

The KB of the example is specified in Listing 2. The hierarchy for this KB, with its main class structure and preferences is illustrated in Fig. 4. The specification is provided directly when the KB is defined and contains the initial preferences of the task. Preferences can be updated dynamically along the execution of the task.

For the class *human*, one particular instance is given, namely *user*. In the KB, the specification of *user* means that if he had a bad day then he is tired with preference 1. Furthermore, the room where the user might be found follows two principles: (i) after coming back from work and feeling tired the user goes to the living room with preference 1; and (ii) if the user has already asked the robot for a comestible product he moves to the dining room with preference 2.

The class *comestible* states that the item with the nearest best-by date is to be served with preference 1; the healthiest item is to be served with preference 2; and the item asked by the user has preference 3. Both food and drinks are comestible products. For food, noodles are about to expire and bisquits is the healthiest item; for drinks, there is no data concerning their best-by date but malz is the healthiest drink. The property *asked* in the KB is set on once the user asks for food or drinks.

The object noodles specifies that it could be found in the shelves of food, snacks and drinks, with preferences 2, 3 and 4, respectively. The location with preference 1 to find noodles is the place where the robot last saw them, if that ever happened. We proceed similarly for the specification of the remaining objects.

The class *point* has five instances, each of them defines a position in the map Golem-III can go to. A name is provided for every point.

```
[
 class(top,none,[],[],[]),
 class(entity, top, [], [], []),
 class(human, entity, [], [], [
    [id=>user, [ [bad_day=>>tired,1],[[back_from_work,tired]=>>found_in=>living_room,1],
                 [asked_comestible=>>found_in=>dining_room,2] ], []]
 ]),
 class(object, entity, [  [graspable=>yes,0]  ], [], []),
 class(comestible,object,[ [asked=>’-’=>>to_serve=>’-’,3],[healthier=>’-’=>>to_serve=>’-’,2],
                                                   [best_by=>’-’=>>to_serve=>’-’,1] ],[],[]),
 class(food,comestible,[  [healthier=>bisquits,0],[best_by=>noodles,0]  ],[],[
  [id=>noodles, [ [’-’=>>loc=>shelf_food,2],[’-’=>>loc=>shelf_snacks,3],
                  [’-’=>>loc=>shelf_drinks,4],[last_seen=>’-’=>>loc=>’-’,1] ],[]],
  [id=>bisquits,[ [’-’=>>loc=>shelf_food,2],[’-’=>>loc=>shelf_snacks,3],
                  [’-’=>>loc=>shelf_drinks,4],[last_seen=>’-’=>>loc=>’-’,1] ],[]]
 ]),
 class(drink,comestible,[  [healthier=>malz,0]  ], [], [
  [id=>coke, [ [’-’=>>loc=>shelf_drinks,2],[’-’=>>loc=>shelf_snacks,3],
               [’-’=>>loc=>shelf_food,4],[last_seen=>’-’=>>loc=>’-’,1] ],[]],
  [id=>malz, [ [’-’=>>loc=>shelf_drinks,2],[’-’=>>loc=>shelf_snacks,3],
               [’-’=>>loc=>shelf_food,4],[last_seen=>’-’=>>loc=>’-’,1] ],[]]
 ]),
 class(point,entity,[],[],[
  [id=>welcome_point,[ [name=>’welcome_point’,0]  ],[]],
  [id=>living_room,  [  [name=>’living room’,0]  ],[]],
  [id=>dining_room,  [  [name=>’dining room’,0]  ],[]],
  [id=>shelf_food,   [  [name=>’the shelf of food’,0]  ],[]],
  [id=>shelf_drinks, [  [name=>’the shelf of drinks’,0]  ],[]],
  [id=>shelf_snacks, [  [name=>’the shelf of snacks’,0]  ],[]]
 ])
]
```

Listing 2. The KB with the preferences associated to the storyboard of Fig. 1.

4.3. Inference mechanism on preferences in the KB

The mechanism that makes the KB work may activate a sequence of inferences on preferences in a way that consequents from one or more preferences become the list of antecedents of another preference, repeating this process until no further consequent is obtained. Each inference is resolved by applying the Principle of Specificity on the hierarchy tree.

In the example considered earlier, Golem-III defines two properties in the KB related to the user after welcome him: *back_from_work* and *bad_day*. Later, the user asked for comestibles, so the property *asked_comestible* is added to the KB. When Golem-III wants to deliver malz and noodles to the user, the robot queries the KB to get the preference on where the human may be found. This triggers a chained inference on preferences concluding that the user is in the dining room with preference 2 and in the living room with preference 1. Therefore, Golem-III goes to the latter location to meet the user.

Finally, the KB is also able to update preferences. This is shown at the end of the interaction, when the robot notices that the noodles were not found in the shelf of food, but in the shelf of snacks. The user is asked whether the location for noodles have to be updated. The user agrees and the preferences are updated accordingly.

The code for the task described in this paper, including the DMs, user functions and the KB is available at http://golem.iimas.unam.mx/reasoning_about_prefs. A video of the task can be seen at the same site.

5. The robot golem-III

Golem-III is a realization of the conceptual module presented in [14] and its behaviour is regulated by the Interaction Oriented Cognitive Architecture (IOCA) [13]. The IOCA architecture specifies the types of modules which integrate the system. A diagram of IOCA can be seen in Fig. 5. Recognition modules encode external stimuli into specific modalities (e.g., speech into transcriptions, images to SIFT features). The perceptual interpreter modules assign interpretations to the output of the recognition modules for the different modalities. On the output side, specification modules ground global parameters to values that are able to be rendered at a low level (e.g., "room" to [x,y] coordinates). Rendering modules control the execu-

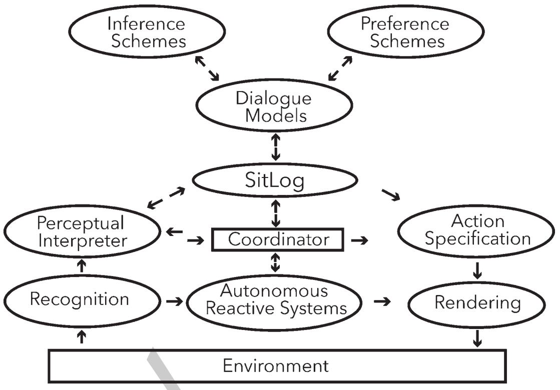

Fig. 5. The Interaction-Oriented Cognitive Architecture.

tion of the actions (e.g., perform navigation actions to arrive the room). The dialogue manager is the interpreter of the SitLog programming language [15], designed for task description and interpretation, and controls the input and output behaviors, and manage the knowledge resources on demand.

Reactive behaviors, such as obstacle evasion or the emergency stop, do not require any task context. Thus, management from the top dialogue manager is unnecessary. To achieve this, recognition and rendering modules are tightly joined into Autonomous Reactive Systems (ARSs) that can communicate to the dialogue manager directly by a coordinator.

The robot Golem-III, shown in Fig. 6 is our in-house service robot. It has a mobile base, adapted from a MobileRobot Research PatrolBot. Golem-III has a functional torso, which controls both arms, the robot's height, and a 2-DOF head.

6. Conclusions

In this paper we have presented a service robot able to support its users considering preferences explicitly. Preferences are useful when action directives or commands stated by the user stand in conflict with his or her own believes or attitudes, healthy or ethical guide-lines or simply with common sense knowledge. Preferences are also useful when commands are underspecified or incomplete.

Preferences are defeasible knowledge as they can be reviewed with time or along robots service tasks and their expression requires a non-monotonic knowledge service. In this paper we have augmented our previous non-monotonic knowledge-base for service robots –that was capable of expressing class hierarchies with incomplete knowledge, strong

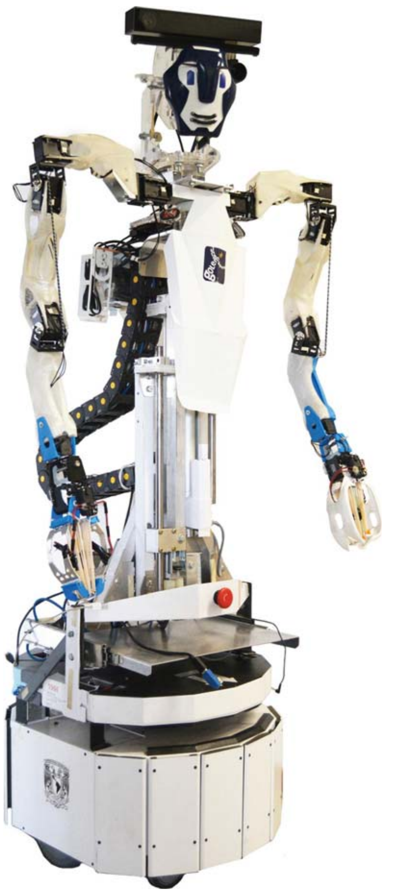

Fig. 6. Golem III+.

negation, defaults and exceptions– with multiple extensions, through a generalization of the Principle of Specificity. Multiple extension can be thought of as defaults or exceptions that result from different implication chains that can stand in conflict with each other.

Preferences are handled on demand along the interaction. For this we have presented a generic, domain independent protocol or dialogue model that models a typical conversation involving preferences. The dialogue is specified through the SitLog programming language for the declarative specification and interpretation of the task and communication structure of service robots. The dialogue manager handles the sequence of situations in which preferences are suggested by the robot and confirmed by the user. Preferences are consulted and updated on demand from the knowledge-base until the completion of the task. We have also presented an scenario in which the Golem-III service robots performs a task involving preferences.

The present work shows how a non-monotonic knowledge-base with multiple extensions can be used to support reasoning about preferences by service robots in a practical setting.

Acknowledgements

We are grateful to Mauricio Reyes, Ricardo Cruz, Hernando Ortega and Dennis Mendoza, members of the Golem Group at IIMAS, UNAM, for their support in the development of the robot Golem-III and the implementation of the scenario presented in this paper. We also acknowledge the support of grants PAPIIT-UNAM's IN-109816 and CONACYT's 178673.

References

[1] R. Alami, A. Clodic, V. Montreuil, E.A. Sisbot and R. Chatila, Task planning for human-robot interaction, In: *Proceedings of the Joint Conference on Smart Objects and Ambient Intelligence: Innovative Context-aware Services: Usages and Technologies*, 2005, pp. 81–85.

[2] J. Becker, C. Bersch, D. Pangercic, B. Pitzer, T. Rühr, B. Sankaran, J. Sturm, C. Stachniss, M. Beetz and W. Burgard, The pr2 workshop-mobile manipulation of kitchen containers, In: *IROS Workshop on Results, Challenges and Lessons Learned in Advancing Robots with a Common Platform*, vol 120, 2011.

[3] S. Benferhat, C. Cayrol, D. Dubois, J. Lang and H. Prade, Inconsistency management and prioritized syntax-based entailment, In: *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*, 1993, pp. 640–645.

[4] G. Brewka, Reasoning about priorities in default logic, In: *Proceedings of the Twelfth National Conference on Artificial Intelligence*, vol 2, 1994, pp. 940–945.

[5] G. Brewka and T. Eiter, Preferred answer sets for extended logic programs, *Artificial Intelligence* **109**(1) (1999), 297–356.

[6] X. Chen, J. Ji, J. Jiang, G. Jin, F. Wang and J. Xie, Developing high-level cognitive functions for service robots, In: *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems*, vol 1, 2010, pp. 989–996.

[7] X. Chen, D. Lu, K. Chen, Y. Chen and N. Wang, Kejia: The intelligent service robot for robocup@home 2014. Tech. rep., Multi-Agent Systems Lab., Department of Computer Science and Technology, University of Science and Technology of China, 2014.

[8] J.P. Delgrande and T. Schaub, Expressing preferences in default logic, *Artificial Intelligence* **123**(1) (2000), 41–87.

[9] J.P. Delgrande, T. Schaub and H. Tompits, A framework for compiling preferences in logic programs, *Theory Pract Log Program* **3**(2) (2003), 129–187.

[10] Z. Fan, E. Tosello, M. Palmia and E. Pagello, Applying semantic web technologies to multi-robot coordination, In: *Proceedings of the International Conference Intelligent Autonomous Systems*, 2014.

[11] E. Ferretti, M. Errecalde, A.J. García and G.R. Simari, An application of defeasible logic programming to decision making in a robotic environment, In: *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning*, 2007, pp. 297–302.

[12] D. Pangercic, M. Tenorth, D. Jain and M. Beetz, Combining perception and knowledge processing for everyday manipulation, In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 1065–1071.

[13] L.A. Pineda, I. Meza, H. Aviles, C. Gershenson, C. Rascon, M. Alvarado and L. Salinas, Ioca: Interaction-oriented cognitive architecture, *Research in Computing Science* **54** (2011), 273–284.

[14] L.A. Pineda, A. Rodriguez, G. Fuentes, C. Rascon and I.V. Meza, Concept and functional structure of a service robot, *International Journal of Advanced Robotic Systems*, 2013, pp. 1–15.

[15] L.A. Pineda, L. Salinas, I. Meza, C. Rascon and G. Fuentes, Sitlog: A programming language for service robot tasks, *International Journal of Advanced Robotic Systems* (2013), 1–12.

[16] L.A. Pineda, A. Rodríguez, G. Fuentes, C. Rascón and I. Meza, A light non-monotonic knowledge-base for service robots, *Intel Serv Robotics* **10** (2017), 159–171.

[17] R. Reiter, A logic for default reasoning, *Artificial Intelligence* **13** (1980), 81–132.

[18] E.A. Sisbot, L.F. Marin and R. Alami, Spatial reasoning for human robot interaction, In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007, pp. 2281–2287.

[19] C. Strasser and G.A. Antonelli, Non-monotonic logic. In: Ward N Zalta (ed) The Stanford Encyclopedia of Philosophy (Winter 2014 Edition), New York: Academic Press, 2014.

[20] M. Tenorth and M. Beetz, Knowrob: A knowledge processing infrastructure for cognition-enabled robots, *The International Journal of Robotics Research* **32**(5) (2013), 566–590.

[21] M. Tenorth and M. Beetz, Representations for robot knowledge in the knowrob framework, *Artificial Intelligence* (2015).

[22] M. Tenorth, L. Kunze, D. Jain and M. Beetz, Knowrob-map - knowledge-linked semantic object maps, In: *IEEE-RAS International Conference on Humanoid Robots*, 2010, pp. 430–435.

[23] S. Zhang, M. Sridharan and F. Sheng Bao, ASP+POMDP: Integrating non-monotonic logic programming and probabilistic planning on robots, In: *Proceedings of the IEEE International Conference on Development and Learning and Epigenetic Robotics*, 2012.