Article

Deliberative and Conceptual Inference in Service Robots

Luis A. Pineda [1,*] , Noé Hernández [1], Arturo Rodríguez [2] , Ricardo Cruz [1] and Gibrán Fuentes [1]

1   Department of Computer Science, Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México, Coyoacán 04510, Mexico; nohernan@turing.iimas.unam.mx (N.H.); ccruzm@conacyt.mx (R.C.); gibranfp@unam.mx (G.F.)
2   Facultad de Estudios Superiores Aragón, Universidad Nacional Autónoma de México, Av Hacienda de Rancho Seco S/N, Impulsora Popular Avícola 57130, Mexico; arturorodriguez35@aragon.unam.mx
*   Correspondence: lpineda@unam.mx

Abstract: Service robots need to reason to support people in daily life situations. Reasoning is an expensive resource that should be used on demand whenever the expectations of the robot do not match the situation of the world and the execution of the task is broken down; in such scenarios, the robot must perform *the common sense daily life inference cycle* consisting on diagnosing what happened, deciding what to do about it, and inducing and executing a plan, recurring in such behavior until the service task can be resumed. Here, we examine two strategies to implement this cycle: (1) a pipe-line strategy involving abduction, decision-making, and planning, which we call *deliberative inference* and (2) the use of the knowledge and preferences stored in the robot's knowledge-base, which we call *conceptual inference*. The former involves an explicit definition of a problem-space that is explored through heuristic search, and the latter is based on conceptual knowledge, including the human user preferences, and its representation requires a non-monotonic knowledge-based system. We compare the strengths and limitations of both approaches. We also describe a service robot conceptual model and architecture capable of supporting the daily life inference cycle during the execution of a robotics service task. The model is centered in the declarative specification and interpretation of robot's communication and task structure. We also show the implementation of this framework in the fully autonomous robot Golem-III. The framework is illustrated with two demonstration scenarios.

Keywords: symbolic inference in service robots; non-monotonic reasoning; diagnosis, decision-making, and planning in service robots; reasoning with preferences in service robots; conceptual model of service robots; declarative specification and interpretation of task and communication structure; *SitLog*

1. Inference in Service Robots

Fully autonomous service robots aimed to support people in common daily tasks require competence in an ample range of faculties, such as perception, language, thought, and motor behavior, in which deployment should be highly coordinated for the execution of service robotics tasks. A hypothetical illustrative scenario in which a general purpose service robot performs as a supermarket assistant is shown in the story-board in Figure 1. The overall purposes of the robot in the present scenario are (i) to attend the customer's information and action requests or commands; (ii) to keep the supermarket in order, and (iii) to keep the manager informed about the state of the inventory in the stands. The basic behavior can be specified schematically; but, if the world varies in unexpected ways, due to spontaneous behavior of other agents or to unexpected natural events, the robot has to reason to complete the service tasks successfully. The robot greets the customer, offers help and the customer asks for a beer in the top-left box 1. The command is performed as an indirect speech act in the form of a question. The robot has conceptual knowledge stored in his knowledge-base, including the obligations and preferences of the agents involved. In this case, the restriction that alcoholic beverages can only be served to people over eighteen.

This prompts the robot to issue an information request to confirm the age of the customer. When the user does so, the robot is ready to accomplish the task. The robot has a scheme to deliver the order and also knowledge about the kinds of objects in the supermarket, including their locations. So, if everything is as expected the robot can accomplish the task successfully by executing the scheme. With this information, the robot moves to the stand of drinks where the beer should be found. However, in the present scenario, the Heineken is not there, the scheme is broken down, and, to proceed, the robot needs to reason. As this is an expensive resource, it should be employed on demand.
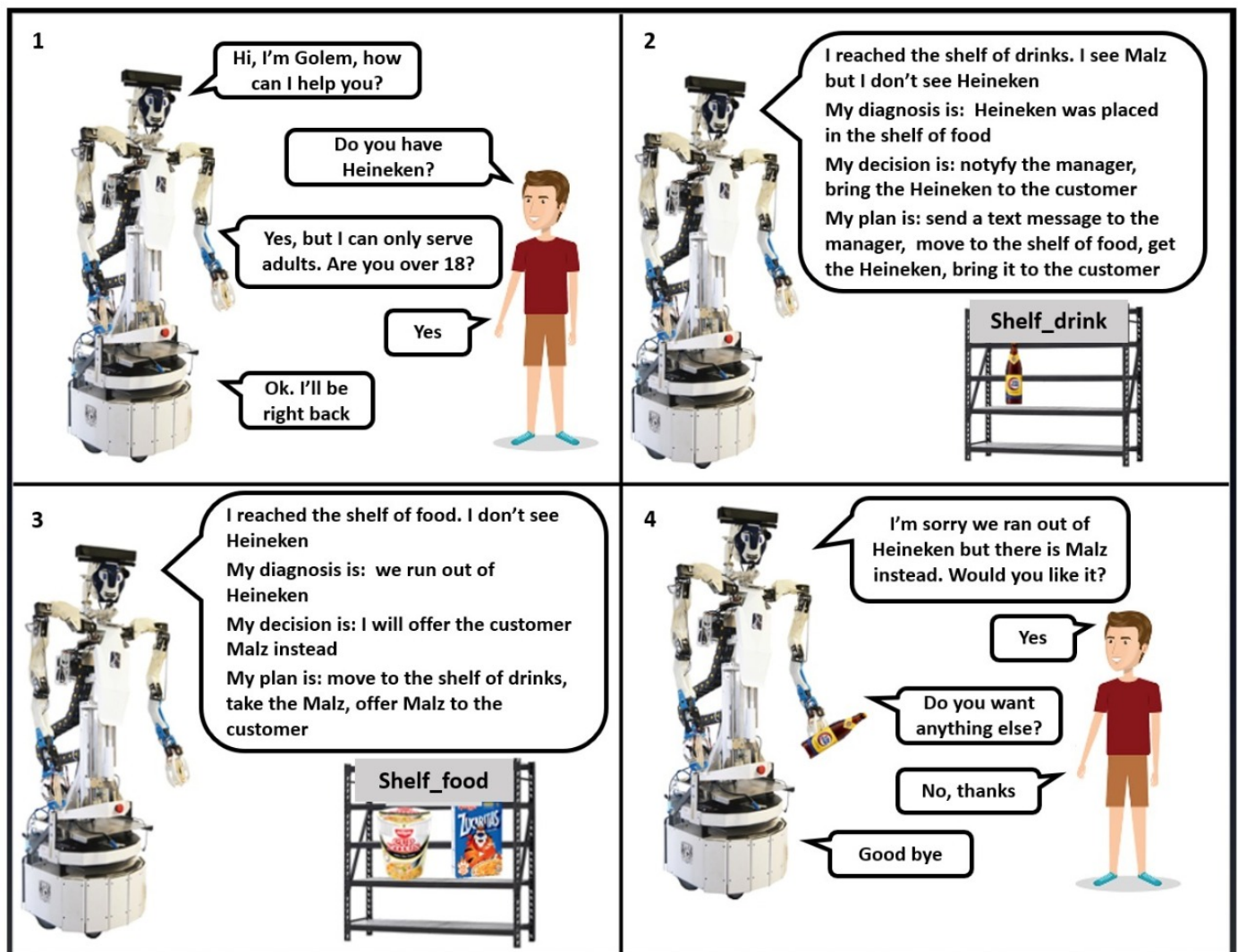

Figure 1. Common sense daily-life inference cycle.

The reasoning process involves three main kinds of inference:

1. An abductive inference process to the effect of diagnosing the cause of the current state of the world which differs from the expected one.
2. A decision-making inference to the effect of deciding what to do to produce the desired state on the basis of the diagnosis and the overall purposes of the agent in the task.
3. The decision made becomes the goal of a plan that has to be induced and carried out to produce the desired state of the world.

We refer to this cycle as the common sense daily-life inference cycle. It also has to be considered that the world may have changed along the development of the task or that the

robot may fail in achieving some actions of the plan, and it needs to check again along the way; if the world is as expected, the execution of the plan is continued; but, if something is wrong, the robot needs to engage recurrently in the daily-life inference cycle until the task is completed, or the robot needs to give up.

The first instance of the daily-life inference cycle in the scenario in Figure 1 is shown in box 2. The inference is prompted by the robot's visual behavior, which is goal-directed, that fails to recognize the intended object. This failure is reported through the declarative speech act *I see Malz, but I don't see the Heineken*. Then, the robot performs a diagnosis inference to the effect of determining what went wrong. This kind of reasoning proceeds from an observation to its causes, has an abductive character and is non-monotonic. In this case, the robot hypothesizes where the beer should be and what was the cause of such state (i.e., *Heineken was placed on the shelf of food*). The decision about what to do next involves two goals: informing the manager the state of the inventory of the drinks stand through a text message—not illustrated in the figure—and delivering the beer to the customer, and a plan to such an effect is induced and executed.

The robot carries on with the plan but fails to find the beer in the stand for food, and the daily-life inference cycle is invoked again, as shown in Figure 1, box 3. The diagnosis this time is that the supermarket ran out of beer, and the decision is to offer the customer the Malz instead. The plan consists of moving back to the shelf of drinks, get the Malz (this action is not shown in the figure), make the offer, and conclude the task, as illustrated in Figure 1, box 4.

The implementation of the scenario relies on two conceptual engines that work together. The first is a methodology and programming environment for specifying and interpreting conversational protocols, which here we call *dialogue models*, that carry on with the schematic behavior, by issuing and interpreting the relevant speech acts during the execution of the task. Dialogue models have two dual aspects and represent both the task structure and the communication structure, which proceed in tandem. The second consists of the inferential machinery that is used on demand, which is called upon within the interpretation of the input and output speech acts. In this paper, we show the conceptual model and physical machinery to support the conversational protocols and inference capabilities require to achieve the kind of service tasks illustrated in Figure 1.

The present framework is used to model two inference strategies to support this kind of tasks, one which we call *deliberative inference* and the other *conceptual inference*, each yielding a form or style of interaction. In the first, the inference conforms to the standard specification of a problem-space that is explored through symbolic search, in which overt diagnosis, decision-making, and planning inferences are performed. This strategy is useful in scenarios in which the robot is given a complex command and is expected to execute it successfully and robustly, dealing with open real-world environments in real-time, such as the General Purpose Service Robot (GPSR) test of the RoboCup@Home competition [1]. The second conforms to situations in which the robots carries out a service task that involves a close interaction in natural language with the user, the use of general and particular concepts, and dynamic specification and interpretation of user beliefs and preferences, which reflect better the needs of socially assistive robotics [2]. Although this scenario also involves diagnosis, decision-making, and planning, these inferences are implicit and result from the interplay between speech act protocols specified in the dialogue models and the use of the knowledge-based service, and nevertheless, achieved effects similar to the first scenario.

The structure of this paper is as follows: A summary of the relevant work on deliberative and conceptual inference in service robots is presented in Section 2. Next, in Section 3, we describe the conceptual model and architecture required to support the inferences deployed by the robot in the present scenario. In Section 4, the *SitLog* programming language for the specification dialogue models or interaction protocols in which inference is used on demand is reviewed. *SitLog* supports the definition of robotics tasks and behaviors, which is the subject of Section 5. The non-monotonic service used to perform

conceptual inferences is reviewed in Section 6. With this machinery in hand, we present the two strategies to implement the daily-life inference cycle. First, in Section 8, we show the pipe-line strategy involving the definition of an explicit problem space and heuristic search. We describe a full demonstration scenario in which the robot Golem-III performs as a supermarket assistant. A previous version of this demo was performed successfully at the final of the RoboCup German Open 2018 in the category @Home. Then, in Section 9, we describe the second scenario in which Golem-III performs as a home assistant. Finally, in Section 10, we discuss the advantage and limitations of both approaches and suggest further work to model better the common sense inferences made by people in practical tasks.

2. Related Work

Service robotics research has traditionally focused on tackling specific functionalities or carrying out tasks that integrate such functionalities (e.g., navigation [3,4], manipulation [5,6], or vision [7,8]). However, relatively few efforts have been made to articulate an integrated concept of service robot. For instance, Rajan and Saffioti [9] organize the field in the dimensions of three cognitive abilities—knowing, reasoning, and cooperating—versus four challenges—uncertainty, incomplete information, complexity, and hybrid reasoning.

In this section, we briefly review related works on high-level programming languages and knowledge representation and reasoning systems for service robots. High-level task programming has been widely studied in robotics, and several domain-specific languages and specialized libraries, as well as extensions to general-purpose programming languages, have been developed for this purpose. Many of these approaches are built upon finite state machines and extensions [10–12], although situation calculus [13–15] and other formalisms are also common. Notable instances of domain-specific languages are the Task Description Language [16], the Extensible Agent Behavior Specification Language (XABSL) [10], XRobots [11], and `ReadyLog` [13]. More recently, specialized frameworks and libraries, such as TREX [5] and SMACH [12], have become attractive alternatives for high-level task programming.

Reasoning is an essential ability in order for service robots to autonomously operate in a realistic scenario, robustly handling its inherent uncertainty and complexity. Most existing reasoning systems in service robots are employed for task planing and decision-making, commonly taking into account spatial and temporal information, in order to allow more adaptable and robust behavior and to facilitate development and deployment (e.g., References [17–19]). These systems typically exploit logical (e.g., References [20–23]) or probabilistic inference (e.g., Partially Observable Markov Decision Processes and variants [24–28]), or combinations of both (e.g., References [29–31]), and have been demonstrated on a wide variety of applications, including manipulation [25], navigation [24], collaborative [23], and interactive [32] tasks. An overview and classification of the so-called "deliberative functions", including planning, acting, monitoring, observing, and learning, is given by Ingrand and Ghallab [33], and the need to integrate deliberative functions with planning and reasoning is emphasized by Ghallab et al. [34].

Reasoning systems rely on knowledge-bases to store and retrieve knowledge about the world, which can be obtained beforehand or dynamically by interacting with the users and/or the environment. One of the most prominent knowledge-base systems for service robots has been KnowRob (Knowledge processing for Robots) [35,36], which is implemented in Prolog and uses the Web Ontology Language (OWL) [37]. KnowRob has multiple knowledge representation and reasoning capabilities and has been successfully deployed in complex tasks, such as identifying missing items on a table [38], operating containers [39], multi-robot coordination [40], and semantic mapping [41]. Non-monotonic knowledge representation and reasoning systems are typically based on Answer Set Programming (ASP) (e.g., References [29,42,43]) and extensions of OWL-DL that allow the use of incomplete information have been defined (e.g., References [44]), some of which have been demonstrated in different complex tasks [42,44–48]. Awaad et al. [44] use OWL-DL to

model preferences and functional affordances for establishing social-accepted behaviors and guidelines to improve human-robot interaction and carrying out tasks in real-world scenarios. In addition, an application of dynamic knowledge-acquisition through the interaction with a teacher is provided by Berlin et al. [49].

In this work, we present a general framework for deliberative and conceptual inference in service robots that its integrated within an interaction-oriented cognitive architecture and accessed through a domain-specific task programming language. The framework allows modeling the common-sense daily life inference cycle, which consists of diagnosis, decision-making, and planning, to endow service robots with robust and flexible behavior. It also implements a light and dynamic knowledge-base system that enables non-monotonic reasoning and the specification of preferences.

3. Conceptual Model and Robotics Architecture

To address the complexity described in Section 1, we have developed an overall conceptual model of service robots [50] and the Interaction-Oriented Cognitive Architecture (IOCA) [51] for its implementation, which illustrated in Figure **??**. IOCA has a number of functional modules and an overall processing strategy that remain fixed across domains and applications, conforming to the functionality of standard cognitive architectures [52–54]. The conceptual, inferential, and linguistic knowledge is specific to domains and applications but is used in a regimented fashion by the system interpreters, which are also fixed. The conceptual model is inspired in Marr's hierarchy of systems levels [55] and consists of the functional, the algorithmic, and the implementation system levels. The functional level is related to *what* the robot does from the point of view of the human-user and focuses on the definition of tools and methodologies for the declarative specification and interpretation of robotic tasks and behaviors; the algorithmic level consists of the specification of *how* behaviors are performed and focuses on the the development of robotics algorithms and devices; finally, the implementation level focuses on system programming, operating systems, and the software agent's specification and coordination.
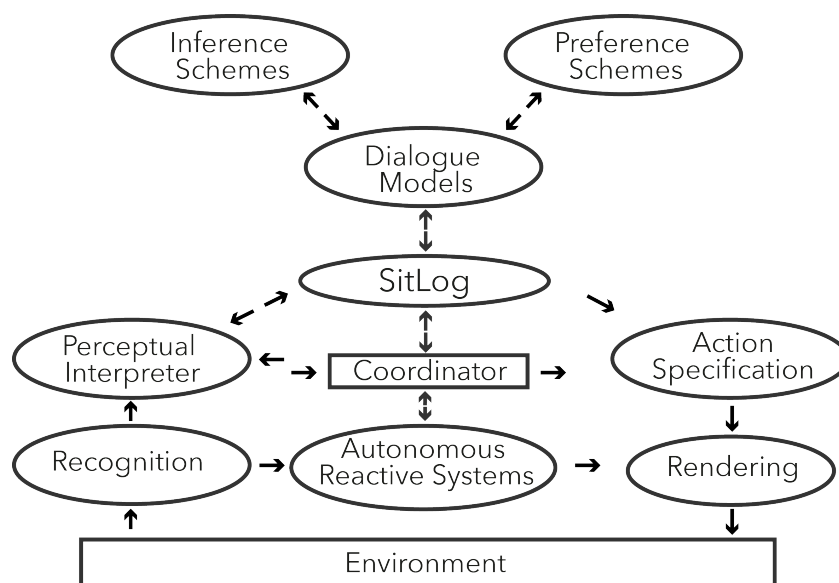

Figure 2. Interaction-Oriented Cognitive Architecture (IOCA).

The present conceptual model provides a simple but powerful way to integrate symbolic AI, which is the focus of the functional system level, and sub-symbolic AI, in which perception, machine learning, and action tasks are specified and executed at the algorithmic system level. The robotics algorithms and devices which have a strong implementation component are defined and integrated at the algorithmic and implementation systems levels. In this way, we share the concerns of bringing AI and robotics efforts closer [9,56].

The core of IOCA is the interpreter of *SitLog* [57]. This is a programming language for the declarative specification and interpretation of the robot's communication and task structure. *SitLog* defines two main abstract data-types: the *situation* and the *dialogue model* (DM). A situation is an information state defined in terms of the expectations of the robot, and a DM is a directed graph of situations representing the task structure. Situations can be *grounded* and correspond to an actual spatial and temporal estate of the robot, where concrete perceptual and motor actions are performed, or *recursive*, consisting on a full dialogue model, possibly including recursive situations, permitting the definition of large abstractions of the task structure.

Dialogue models have a dual interpretation as conversational or speech acts protocols that the robot performs along the execution of a task. From this perspective, expectations are the speech acts that can potentially be expressed by an interlocutor, either human or machine, in the current situation. Actions are thought of as the speech acts performed by the robot as a response to such interpretations. Events in the world that can occur in the situation are also considered expectations that give rise to intentional action by the robot. For this reason, dialogue models represent the communication or interaction structure, and they correspond to the task structure.

Knowledge and inference resources are used on demand within the conversational context. These "thinking" resources are also actions, but, unlike perceptual and motor actions, which are directed to the interaction with the world, thinking is an internal action that mediates the input and output, permitting the robot to anticipate and cope better with the external world. The communication and interaction cycle is then the center of conceptual architecture and is oriented to interpret and act upon the world but also to manage thinking resources that are embedded within the interaction, and the interpreter of *SitLog* coordinates the overall intentional behavior of the robot.

The present architecture supports rich and varied but schematic or stereotyped behavior. Task structure can proceed as long as at least one expectation in the current situation is met by the robot. However, schematic behavior can easily break down in dynamic worlds when none or more than one expectations are satisfied in the situation. When this happens, the interpretation context is lost, and the robot needs to recover it to continue with the task. There are two strategies to deal with such contingencies: (1) to invoke domain independent dialogue models for task management, which here we refer to as *recovery protocols*, or (2) to invoke inference strategies to recover the ground. In this latter case, the robot requires to make an abductive inference or a diagnosis in order to find out why none of its expectations were met, decide the action needed to recover the ground on the basis of such diagnosis, in conjunction with a given set of preferences or obligations, and induce and execute a plan to achieve such goal. Here, we refer to the cycle of diagnosis-decision-making-planning as the *daily life inference cycle*, which is specified in Section 7. This cycle is invoked by the robot when schematic behavior cannot proceed, and a recovery protocol that is likely to recover the ground is not available.

The architecture includes a low level reactive cycle involving low-level recognition and rendering of behaviors that are managed by the Autonomous Reactive System directly. This cycle is embedded within the communication or interaction cycle and has the *SitLog*'s interpreter as its center, which performs the interpretation of the input and the specification of the output in relation to current situation and dialogue model. The reactive and communication cycles normally proceed independently and continuously, the former working at as least one order of magnitude faster than the latter, although there are times in which one needs to take full control of the task for performing a particular process, and there is a loose coordination between the two cycles.

The perceptual interpreters are modality specific and receive the output of the low-level recognition process bottom-up but also the expectations in the current situation top-down, narrowing the possible interpretations of the input. There is one perceptual interpreter for each input modality which instantiates the expectation that is meaningful in relation to the context. Expectations are, hence, *representations of interpretations*. The per-

ceptual interpreters promote sub-symbolic information produced by the input modalities into a fully articulated representation of the world, as seen by the robot in the situation. Standard perception and action robotics algorithms are embedded within modality specific perceptual interpreters for the input and for specifying the external output, respectively.

The present conceptual model supports the so-called *deliberative functions* [33] embodied in our Robot Golem-III, such as planning, acting, monitoring, observing, and acquiring knowledge through language, which is a form of learning, but also other higher-level cognitive functions, such as performing diagnosis and decision-making dynamically, and carrying on intentional dialogues based on speech act protocols. Golem-III is an in-house development including the design and construction of the torso, arms, hands, neck, head and face, using a PatrolBot base built by MobileRobots Inc., Amherst, NH, USA, 2014 https://www.generationrobots.com/media/PatrolBot-PTLB-RevA.pdf.

4. The *SitLog* Programming Language

The overall intelligent behavior of the robot in the present framework depends on the synergy of intentional dialogues oriented to achieve the goals of the task and the inference resources that are used on demand within such purposeful interaction. The code implementing the *SitLog* programming language is available as a GitHub repository at https://github.com/SitLog/source_code/.

4.1. SitLog's Basic Abstract Data-Types

The basic notion of *SitLog* is the *situation*. A situation is an information state characterized by the expectations of an intentional agent, such that the agent, i.e., the robot, remains in the situation as long as its expectation are the same. This notion provides a large spatial and temporal abstraction of the information state because, although there may be large changes in the world or in the knowledge that the agent has in the situation, its expectations may nevertheless remain the same.

A situation is specified as a set of expectations. Each expectation has an associated action that is performed by the agent when such expectation is met in the world and the situation that is reached as a result of such action. If the set of expectations of the robot after performing such an action remain the same, the robot recurs to the same situation. Situations, actions, and next situations may be specified concretely, but *SitLog* also allows these knowledge objects to be specified through functions, possible higher-order, that are evaluated in relation to the interpretation context. The results of such an evaluation are concrete interpretations and actions that are performed by the robot, as well as the concrete situation that is reached next in the robotics task. Hence, a basic task can be modeled with a directed graph with a moderate and normally small set of situations. Such a directed graph is referred to in *SitLog* as a *Dialogue Model*. Dialogue models can have recursive situations including a full dialogue model, providing the means for expressing large abstractions and modeling complex tasks. A situation is specified in *SitLog* as an attribute-value structure, as follows in Listing 1:

SitLog's interpreter is programmed fully in Prolog (We used SWI-Prolog 6.6.6 https://www.swi-prolog.org/) and subsumes Prolog's notation. Following Prolog's standard conventions, strings starting with lower and upper case letters are *atoms* and *variables*, respectively, and the ==> is an operator relating an attribute with its corresponding value. Values are expressions of a functional language, including constants, variables, operators, predicates, and operators, such as unification, variable assignment, and the *apply* operator for dynamic binding and evaluation of functions. The functional language supports the expression of higher-order functions, too. The interpretation of a situation by *SitLog* consists of the interpretation of all its attributes from top to bottom. The attributes `id`, `type`, and `arcs` are mandatory. The value of `prog` is a list of expressions of the functional language, and, in case such attribute is defined, it is evaluated unconditionally before the `arcs` attribute.

Listing 1. Specification of *SitLog*'s Situation

```
[
ID ==> Situation_ID(Arg_List),
type ==> Situtation_Type_ID,
in_arg ==> In_Arg,
out_arg ==> Out_Arg,
prog ==> Expression,
arcs ==> [
Expect1:Action1 => Next_Sit1,
Expect2:Action2 => Next_Sit2,
...
Expectn:Actionn => Next_Sitn
]
]
```

A dialogue model is defined as a set of situations. Each DM has a designated initial situation and at least a final one. A *SitLog* program consists of a set of DMs, one designed as the *main* DM. This may include a number of situations of type `recursive` each containing a full DM. *SitLog*'s interpreter unfolds a concrete graph of situations, starting form the initial situation of the main DM and generates a Recursive Transition Network (RTN) of concrete situations. Thus, the basic expressive power of *SitLog*'s corresponds to a context-free grammar. *SitLog*'s interpreter consists of the coordinated tandem operation of the RTN's interpreter that unfolds the graph of situations and the functional language that evaluates the attributes' values.

All links of the `arcs` attribute are interpreted during the interpretation of a situation. Expectations are sent to the perceptual interpreter top-down, which instantiates the expectation that is met by the information provided by the low-level recognition processes, and sends such expectation back, bottom-up, to *SitLog*'s interpreter. From the perspective of the declarative specification of the task, `EXPECTn` contains the information provided by perception. Once an expectation is selected, the corresponding action and next situation are processed. In this way, *SitLog* abstracts over the external input, and such an interface is transparent for the user in the declarative specification of the task. Expectations and actions may be empty, in which case a transition between situations is performed unconditionally.

4.2. SitLog's Programming Environment

SitLog's programming environment includes the specification of a set of global variables that have scope over the whole of the program, as well as a set of local variables that have scope over the situations of a particular dialogue model. *SitLog*'s variables are also defined as attribute-value pairs, the attribute being a Prolog's atom and its value a standard Prolog's variable. *SitLog* variables can have arbitrary Prolog's expressions as their values. All variables can have default values and can be updated through the standard assignment operator, which is defined in *SitLog*'s functional language.

Dialogue models and situations can also have arguments, in which values are handled by reference in the programming environment, and dialogue models and situations allow input and output values that can propagate through the graph by these interface means.

The programming environment includes also a *pipe* global communication structure that provides an input to the main DM and propagates through all concrete DMs and situations that unfold in the execution of the task. This channel is stated through the attributes `in_arg` and `out_arg`, in which definition is optional. The value of `out_arg` is not specified when the situation is called upon, i.e., it is a variable, and can be given a value in the body of the situation through a variable assignment or through unification. In case there is no such assignment, the input and output pipes are unified when the interpretation of the situation is concluded. The value of `in_arg` can be also underspecified, and given a

value within the body of the situation, too. In case these are not stated explicitly, the value of `in_arg` propagates to `out_arg` by default as was mentioned.

Global and local variables, as well as the values of the pipe, have scope over the local program and within each link of the `arcs` attribute, and their values can be changed through the *SitLog*'s assignment operator or through unification. However, a local program and links are encapsulated local objects that have no scope outside their particular definition. Hence, Prolog's variables defined in `prog` and in different links of `arcs` attribute are not bounded, even if they have the same name. The strict locality of these programs has proven to be very effective for the definition of complex applications.

The programming environment includes, as well, a *history* of the task conformed by the stack structure of all dialogue models and situations, with their corresponding concrete expectation, action and next situation, unfolded along the execution of the task. The current history can be consulted through a function of the functional language and can be used not only to report what happened before but also to make a decision about the future course of action.

The elements of the programming environment augment the expressive power of *SitLog*, which corresponds overall to a context-sensitive grammar. The representation of the task is, hence, very expressive but still preserves the graph structure, and *SitLog* provides a very good compromise between expressiveness and computational cost.

4.3. SitLog's Diagrammatic Representation

SitLog programs have a diagrammatic representation, as illustrated in Figure 3 (the full commented code of the present *SitLog* program is given in Appendix A). DMs are bounded by large dotted ovals, including the corresponding situation's graph (i.e., *main* and *wait*). Situation are represented by circles with labels indicating the situation's ID and type. In the example, the *main* DM has three situations in which IDs are `is`, `fs`, and `rs`. The situation identifier is optional—provided by the user—except the initial situation that has the mandatory ID `is`. The types IDs are also optional with the exception of the `final` and `recursive`, as these are used by *SitLog* to control the stack of DMs. The links between situations are labeled with pairs of form $\alpha{:}\beta$, which stand for expectations and actions, respectively. When the next situation is specified concretely the arrow linking to situations is stated directly; however, if the next situation is stated through a function (e.g., *h*), there is a large bold dot after the $\alpha{:}\beta$ pair with two or more exit arrows. This indicates that the next situation depends on the value of *h* in relation to the current context, and there is a particular next situation for each particular value. For instance, the edge of `is` in de DM *main* that is labeled by [day, *f*]:*g*, representing the expectation as a list of the value of the local variable `day` and the function *f* and the action as the function *g*, is followed by a large black dot labeled by the function *h*; this function has two possible values, one cycling back into `is` and the other leading to the recursive situation `rs`. This depicts when the expectation that is met at the initial situation satisfies the value of the local variable `day` and the value of function *f*, so the action defined by the value of function *g* is performed, and the next situation is the value of function *h*.

The circles representing recursive situations have also large internal dots representing control return points from embedded dialogue models. The dots mark the origin of the exit links that have to be transversal whenever the execution of an embedded DM is concluded, when the embedding DM is popped up from the stack and resumes execution. The labels of the expectations of such arcs and the names of the corresponding final states of the embedded DM are the same, depicting that the expectations of a recursive situation correspond to the designated final states of the embedded DM. This is the only case in which an expectation is made available internally to the interpreter of *SitLog* and is not provided by a perceptual interpreter as a result of an observation from the external world.

Finally, the bold arrows depict the information flow between dialogue models. The output bold arrow leaving *main* at the upper right corner denotes the value of `out_arg` when the task is concluded. The bold arrow from *main* to *wait* denotes the corresponding

pipe connection, such that the value of `out_arg` of the situation `rs` in *main* is the same as the value of `in_arg` in the initial situation of *wait*. The diagram also illustrates that the value of `in_arg` propagates back to *main* through the value of `out_arg` in both final situations `fs1` and `fs2`; since the attribute `out_arg` is never set within the DM *wait*, the original value of `in_arg` keeps passing on through all the situations, including the final ones. The expectations of the arcs of `is` in the DM *wait* take the input from the perceptual interpreter being either the value of `in_arg` or the atom `loop`.
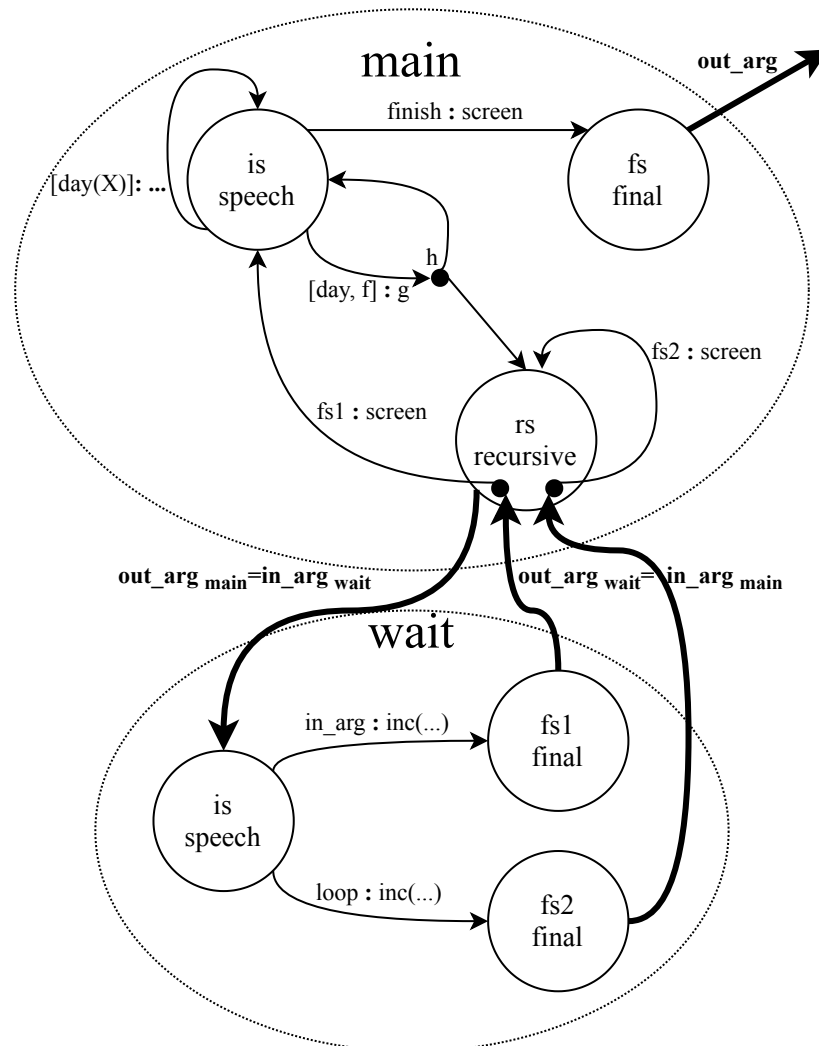

Figure 3. Graphical representation of an example dialogue model written in *SitLog*.

5. Specification of Task Structure and Robotics Behaviors

The functional system level addresses the tools and methodologies to define the robot's competence. In the present model, such competence depends on a set of robotics behaviors and a composition mechanism to specify complex tasks. Behaviors rely on a set of primitive perceptual and motor actions. There is a library of such basic actions, each associated to a particular robotics algorithm. Such algorithms constitute the "innate" capabilities of the robot.

In the present framework, robotics behaviors are *SitLog* programs in which the purpose is to achieve a specific goal by executing one or more basic actions within a behavior's specific logic. Examples of such behaviors are *move*, *see*, *see_object*, *approach*, *take*, *grasp*, *deliver*, *relieve*, *see_person*, *detect_face*, *memorize_face*, *reconize_face*, *point*, *follow*, *guide*, *say*, *ask*, etc. The *SitLog*'s code of *grasp*, for instance, is available at the GitHub repository of *SitLog* https://bit.ly/grasp-dm.

Behaviors are parametric abstract units that can be used as atomic objects but can also be defined as structured objects using other behaviors. For instance, *take* is a composite behavior using *approach* and *grasp* and *deliver* uses *move* and *relieve*. Another example is *see*, which uses *see_object*, *see_person* and *see_gesture* to generally interpret a visual scene.

All behaviors have a number of terminating *status*. If the behavior is executed successfully the status is *ok*; however, there may be a number of failure conditions, particular to the behavior, that may prevent its correct termination, and each is associated with a particular *error* status. The dialogue model at the application layer should consider all possible status of all behaviors in order to improve the robot's reliability.

Through these mechanisms complex behaviors can be defined, such as *find*, that, given a search path and a target object or person, enables the robot to explore the space using the *scan* and *tilt* behaviors to move its head and make visual observations at different positions and orientations. The full *SitLog*'s code of *find* is also provided at the GitHub repository of *SitLog* https://bit.ly/find-dm.

Behaviors should be quite general, robust and flexible, so they can be used in different tasks and domains. There is a library of behaviors that provide the basic capabilities of the robot from the perspective of the human-user. This library evolves with practice and experience and constitutes a rich empirical resource for the construction and application of service robots [50].

The composition mechanism is provided by *SitLog*, too, which allows the specification of dialogue models that represent the tasks and communication structure. Situations in these latter dialogue models represent stages of a task, which can be partitioned into sub-tasks. So, the tasks as a whole can be seen as a story-board, where each situation corresponds to a picture.

For example, if the robot performs as a supermarket assistant, the structure of the tasks can be construed as (1) take an order from the human customer; (2) find and take the requested product; and (3) deliver the product to the customer. These tasks correspond to the situations in the application layer, as illustrated in Figure 4. Situations can be further refined in several sub-tasks specified as more specific dialogue models embedded on the situations of upper levels, and the formalism can be used to model complex task quite effectively.

The dotted lines from the application to the behaviors layer in Figure 4 illustrate that behaviors are used at the application layer as abstract units at different degrees of granularity. For instance, *find* is used as an atomic behavior but also *detect_face* can be used directly by a situations at the level of the task structure, despite that *detect_face* is used by *find*. The task structure at the application layer can be partitioned in subordinated tasks, too. For this, *SitLog*'s supports the recursive specification of dialogue models and situations, enhancing the expressive power of the formalism.

Although both task structure and behaviors are specified through *SitLog*'s programs, these correspond to two different levels of abstraction. The top level specifies the final application task-structure, and is defined by the final user, while the lower level consists of the library of behaviors, which should be generic and potentially useful in diverse application domains.

From the point of view of an ideal division of labor, the behaviors layer is the responsibility of the robot's developing team, while the application's layer is the focus of teams oriented to the development of final service robot applications.
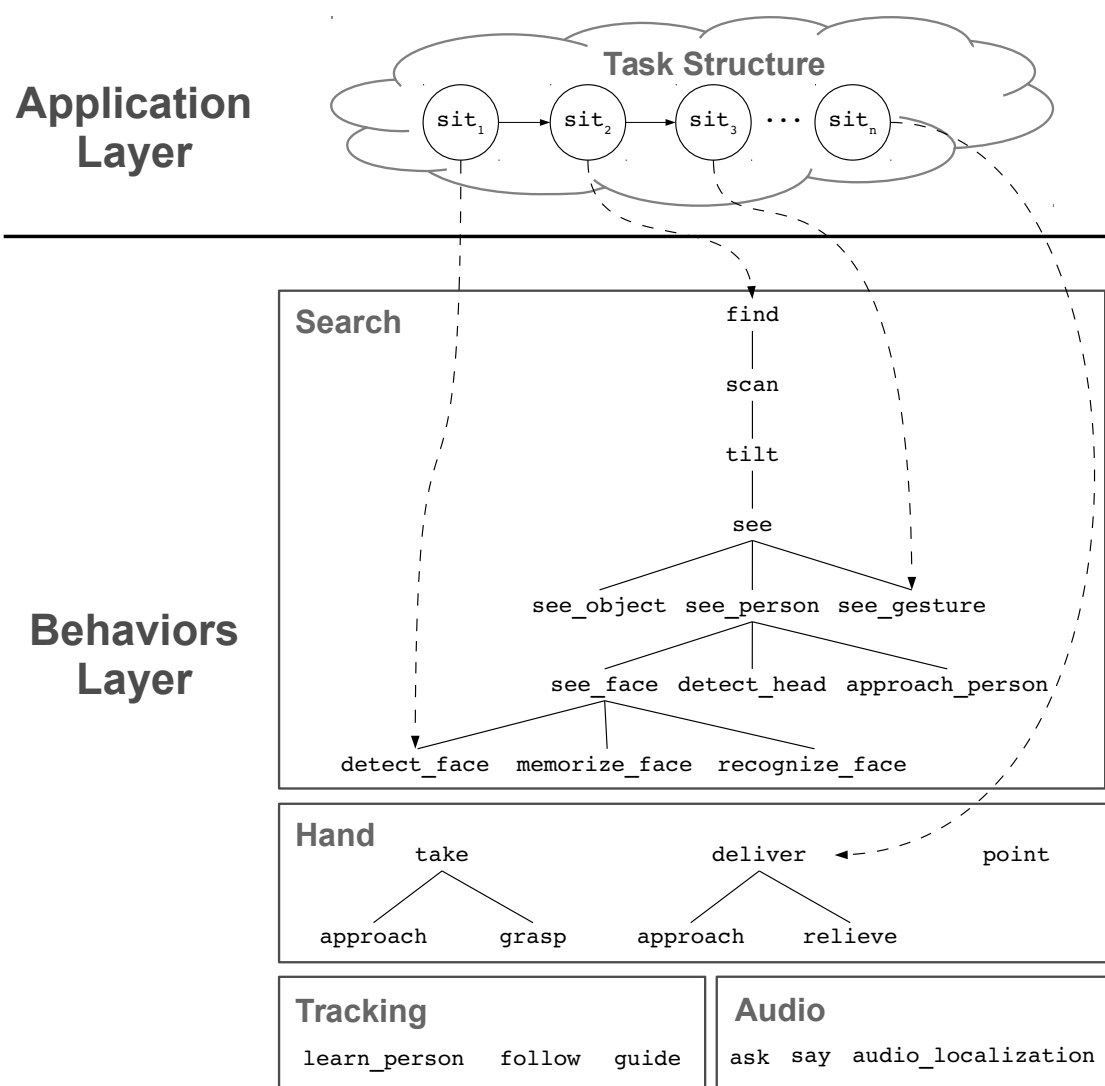
Figure 4. Tasks and behaviors.

The General Purpose Service Robot

Prototypical or schematic robotic tasks can be defined through dialogue models directly. However, the structure of the task has to be known in advance, and there are many scenarios in which this information is not available. For this, in the present framework, we define a general purpose mechanism that translates speech acts performed by the human-user into a sequence of behaviors, which is interpreted by a behavior's dispatcher one behavior at a time, and finishes the task when the list has been emptied [50]. We refer to this mechanism as *General Purpose Service Robot*, or simply *GPSR*.

In the basic case, all the behaviors in the list terminate with the status *ok*. However, whenever the behaviors terminate with a different status, something in the world was unexpected, or the robot failed, and the dispatcher must take an appropriate action. We consider two main types of error situations. The first may be a general but common and known failure, in which case, a recovery protocol is invoked; these protocols are implemented as standard *SitLog*'s dialogue models and undergo a procedure that is specific to fix the error, and, when this is accomplished, they return control to the dispatcher and continue with the task. The second type is about errors that cannot be prevented; to recover from them, the robot needs to engage in the daily-life inference cycle, as discussed in the Section 1 and will be elaborated upon in Sections 7–9.

6. Non-Monotonic Knowledge-Base Service

The specification of service robot's tasks requires an expressive, robust but flexible knowledge-base service. The robot may require to represent, reason and maintain terminological or linguistic knowledge, general and particular concepts about the world, and about the application domain. There may be also defaults, exceptions and preferences, which can be acquired and updated incrementally during the specification and execution of a task, and a non-monotonic KB service is required. Inferences about this kind of knowledge are referred to here as *conceptual inferences*.

To support such functionality, we developed a non-monotonic knowledge-base service based on the specification of a class hierarchy. This system supports the representation of classes and individuals, which can have general or specific properties and relations [58,59]. Classes and individuals are the primitive objects and constitute the ontology. There is a main or top class which includes all the individuals in the universe of discourse; this can be divided in a finite number of mutually exclusive partitions, each corresponding to a subordinated or subsumed class. Subordinated classes can be further partitioned into subordinated mutually exclusive partitions giving rise to a strict hierarchy of an arbitrary depth, and classes are related through a proper inclusion relation. Individual objects can be specified at any level in the taxonomy and the relation between individuals and classes is one of set membership. Classes and individuals can have arbitrary properties and relations, which have generic or specific interpretations, respectively.

The taxonomy has a diagrammatic representation, as illustrated in Figure 5. Classes are represented by circles, and individuals are represented by boxes. The inclusion relation between classes is represented by a directed edge or arrow pointing to the circle representing the subordinated class, and the membership relation is represented by a bold dot pointing to the box representing the corresponding individual. Properties and relations are represented through labels associated to the corresponding circles and boxes; expressions of the form $\alpha => \beta$ stand for a property or a relation, where α stands for the name of the property or relation, and β stands for its corresponding value. The properties or relations are bounded within the scope of their associated circle or box. Classes and individuals can be also labeled with expressions of the form $\alpha =>> \beta, \gamma$ standing for implications, where α is an expression of the form $p_0, p_1, ..., p_n$ for $n \geq 0$, such that p_i is a property or a relation, and β stands for an atomic property or relation with a weight γ, such that β holds for the corresponding class or individual with priority γ, if all p_i in α hold. The KB service allows that objects of relations and values of properties are left underspecified, augmenting its flexibility and expressive power.

For instance, the class `animals` at the top in Figure 5 is partitioned into `fishes`, `birds`, and `mammals`, where the class of birds is further partitioned into `eagles` and `penguins`. The label `fly` stands for a property that all birds have and can be interpreted as an absolute default holding for all individuals of such class and its subsumed classes. The label `eat => animals` denotes a relation between `eagles` with `animals` such that all eagles eat animals, and the question *do eagles eat animals?* is answered *yes*, without specifying which particular eagle eats and which particular animal is eaten. The properties and relations within the scope of a class or an individual, represented by circles and squares, have such class or individual as the subject of the corresponding proposition, but these are not named explicitly. For instance, `like => mexico` within the box for Pete is interpreted as the proposition *Pete likes Mexico*. In the case of classes, such an individual is unspecified, but, in the case of individuals, it is determined. Likewise, the labels `work(y) =>> live(y),3`; `born(y) =>> live(y),5`; and `like(y) =>> live(y),6`, within the scope of `birds` stand for implications that hold for all unnamed individuals *x* of the class `birds` and some individual y, which is the value of the corresponding property or the object of the relation, e.g., *if x works at y, then x lives at y*. Such implications are interpreted as conditional defaults, preferences, or abductive rules holding for all birds that work at, were born in, or like y. The integer numbers are the weights or priorities of such preference, with the convention that the lower the value the larger its priority. Labels without weights are assumed to have

a weight of 0 and represent the absolute properties or relations that classes or individuals have. The label `size  => large` denotes the property `size` of `pete` and its corresponding value, which is `large`. The labels `work  => mexico`; `born => argentina`; and `like =>` `mexico` denote relations of Pete with their corresponding objects (México and Argentina). The system also supports the negation operator `not`, so all atoms can have a positive or a negative form (e.g., `fly`, `not(fly)`).
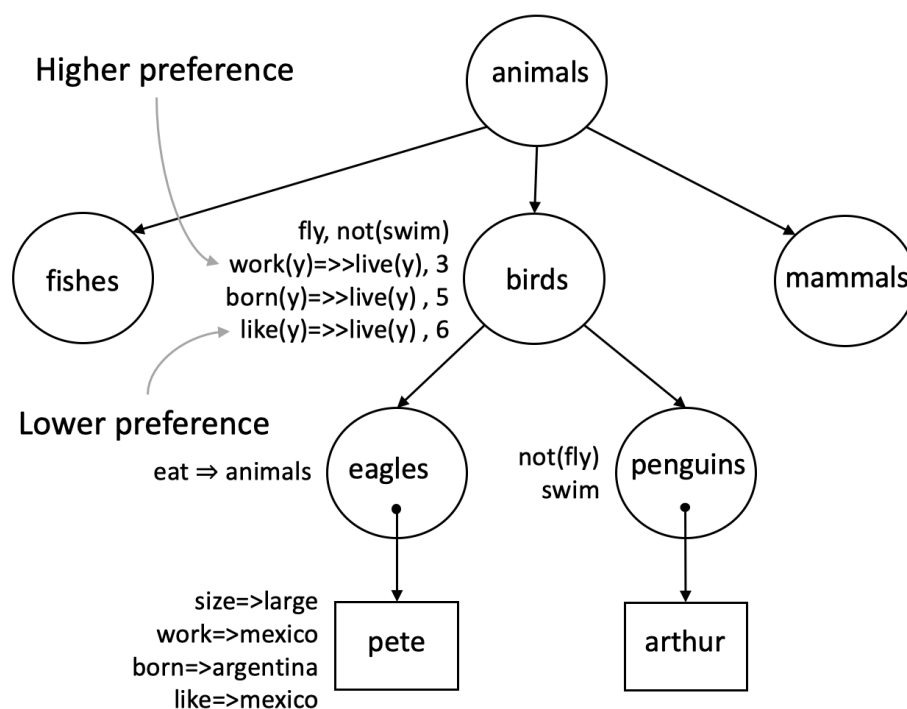

Figure 5. Non-monotonic taxonomy.

Class inclusion and membership are interpreted in terms of the inheritance relation such that all classes inherit the properties, relations, and preferences of their sub-summing or dominant classes, and individuals inherit the properties, relations, and preferences of their class. Hence, the extension or closure of the KB is the knowledge specified explicitly, plus the knowledge stated implicitly, through the inheritance relation.

The addition of the `not` operator allows the expression of incomplete knowledge, as opposed to the Closed World Assumption (CWA). Hence, queries are interpreted in relation to the strong negation and may be answered *yes*, *no* and *not known*. For instance, the questions *do birds fly?*, *do birds swim?*, and *do fish swim?* in relation to Figure 5 are answered *yes*, *no*, and *I don't know* (In case the CWA were assumed, queries would be right only in case complete knowledge about the domain were available, but they could be wrong otherwise. For instance, the queries *do fish swim?* and *do mammals swim?* would be both answered *no* in relation to the CWA, which would be wrong for the former but generally right for the latter).

Properties, relations and preferences can be thought of as defaults that hold in the current class and over all the subsumed classes, as well as for the individual members of such classes. Such defaults can be positive, e.g., birds fly, but also negative, e.g., birds do not swim; defaults can have exceptions, such as penguins, which are birds that do not fly but do swim.

The introduction of negation augments the expressive power of the representational system and allows for the definition of exceptions, but it also allows the expression of contradictions, such as that penguins can and cannot fly, and swim and do not swim. To support this expressiveness and coherently reason about this kind of concept, we adopt the principle of specificity, which states that, in case of conflicts of knowledge, the more

specific propositions are preferred. Subsumed classes are more specific than subsuming classes, and individuals are more specific than their classes. Hence, in the present example, the answer to *do penguins fly?*, *do penguins swim?*, and *does Arthur swim?* are *no*, *yes*, and *yes*.

The principle of specificity chooses a consistent extension of a set of atomic propositions, positive and negative, that can be produced out of the empty set by obtaining two extensions or branches, one with the positive and the other with its negation, one proposition at a time, for all end nodes of each branch and for all atomic propositions that can be formed with the atoms in the theory. These extensions give rise to a binary tree of extended theories in which each path represents a consistent theory, but all different paths are inconsistent between each other. In the present example, the principle of specificity chooses the branch including `{not(fly(pinguins)), swin(pinguins), not(fly(arthur)), swin(arthur)}`. The set of possible theories that can be constructed in this way are referred to as *multiple extensions* [60].

The principle of specificity is a heuristics for choosing a particular consistent theory among all possible extensions. Its utility is that the extension at each particular state of the ontology is determined directly by the structure of the tree, or the strict hierarchy. Changing the ontology, i.e., augmenting or deleting classes or individuals, or changing their properties or relations, changes the current theory; some propositions may change their truth value, and some attributes may change their values, but the inference engine chooses always the corresponding consistent theory or the coherent extension.

Preferences can be thought of as conditional defaults that hold in the current class and over all subsumed classes, as well as for their individual members, if their antecedents hold. However, this additional expressiveness gives rise to contradictions or incoherent theories but this time due to the implication. In the present example, the preferences `work(y) =>> live(y),3` and `born(y) =>> live(y),5` of birds are inherited to Pete whom works in México but was born in Argentina; as Pete works in México and was born in Argentina, therefore, he lives both in México and in Argentina, which is incoherent. This problem is solved by the present KB service through the weight value or priority, and as this is 3 for México and 5 for Argentina, the answer for *where does Pete lives?* is *México*.

Preferences can also be seen as abductive rules that provide the most likely explanation for an observation. For instance, if the property `live=>mexico` is added within the scope of Pete, the question of *why does Pete live in México* can be answered *because he works in México*, i.e., `work(y) =>> live(y),3`, which is preferred over the alternative *because he likes México*, i.e., `like(y) =>> live(y),6`, since the former preference has a lower priority. This kind of rule can also be used to diagnose the causes or reasons of arbitrary observations and constitutes a rich conceptual resource to deal with unexpected events that happen in the world.

The KB is specified as a list of Prolog clauses with five arguments: (1) the class id; (2) the subsuming or mother class; (3) the list of properties of the class; (4) the list of relations of the class; and (5) the list of individual objects of the class. Every individual is specified as a list, with its id, the list of its properties and the list of its relations. Each property and relation is also specified as a list, including the property or relation itself and its corresponding weight. Thus, preferences of classes and individuals may be included in both the property list and the relation list, suggesting that they constitute conditional properties and relations. IDs, properties, and relations are specified as attribute-value pairs, such that values can be objects of well-defined Prolog's forms. The actual code of the KB illustrated in Figure 5 is given in Listing 2.

Listing 2. Full code of example taxonomy.

```
[
%The 'top' class in mandatory
class(top,none,[],[],[]),
class(animals,top,[],[],[]),
class(fish,animals,[],[],[]),
class(birds,animals,[[fly,0],
[not(swim),0],
[work=>'-'=>>live=>>'-',3],
[born=>'-'=>>live=>>'-',5],
[like=>'-'=>>live=>>'-',6]],
[],[]),
class(mammals,animals,[],[],[]),
class(eagles,birds,[],[[eat=>animals,0]],
[[id=>pete,[[size=>large,0]],
[[work=>mexico,0],
[born=>argentina,0],
[like=>mexico,0]
]]
]),
class(penguins,birds,[[swim,0],[not(fly),0]],[],[[id=>arthur,[],[]])
]
```

The KB service provides eight main services for retrieving information from the non-monotonic KB over the closure of the inheritance relations [58], as follows:

1. `class-extension(Class, Extension)` provides the set of individuals in the argument class. If this is top, this service provides the full set of individuals in the KB.
2. `property-extension(Property, Extension)` provides the set of individuals that have the argument property in the KB.
3. `relation-extension(Relation, Extension)` provides the set of individuals that stand as subjects in the argument relation in the KB.
4. `explanation_extension(Property/Relation, Extension)` provides the set of individuals with an explanation supporting why such individuals have the argument property/relation in the KB.
5. `classes_of_individual(Argument, Extension)`: provides the set of mother classes of the argument individual.
6. `properties_of_individual(Argument, Extension)`: provides the set of properties that the argument individual has.
7. `relations_of_individual(Argument, Extension)`: provides the set of relations in which the argument individual stands as subject.
8. `explanation_of_individual(Argument, Extension)` provides the supporting explanations of the conditional properties and relations that hold for the argument individual.

These services provide the full extension of the KB at a particular state. There are, in addition, services to update the values of the KB. There are also services to change, add, or delete all objects in the KB, including classes and individuals, with their properties and relations. Hence, the KB can be developed incrementally and also updated during the execution of a task, and the KB service always provides a coherent value. The full Prolog's code of the KB service is available at https://bit.ly/non-monotonic-kb.

The KB services are manipulated by dialogue models as *SitLog*'s user functions. These services are included as standard *SitLog*'s programs that are used on demand during the interpretation of *SitLog*'s situations. Such services are commonly part of larger *SitLog* programs representing input and output speech acts that are interpreted within structured dialogues defined through dialogue models. Hence, conceptual inferences made on demand during the performance of linguistic and interaction behavior constitute the core of our conceptual model of service robots.

The non-monotonic KB-Service is general and allows the specification of non-monotonic taxonomies, including the expression of preferences and abductive rules in a simple declar-

ative format for arbitrary domains. The system allows the expression of classes with properties and relations, which correspond to roles in Description Logics [61], but also of individuals with particular properties and relations. However, most description logics are monotonic, such as OWL [37], and the expressive power of our system should be compared to Answer Set Programming [62] and systems that can handle incomplete information, as the use of OWL-DL for modeling preferences and functional affordances [44]. There are ontological assumptions and practical considerations that distinguish our approach from alternative representation schemes, but such discussion and comparative evaluation are beyond the scope of the present paper.

7. The Daily-Life Inference Cycle

Next, we address the specification and interpretation of the daily-life inference cycle, as described in Section 1. This cycle is studied from two different perspectives: the first consists of the pipe-line execution of a diagnosis, a decision-making, and a planning inference, and it involves the explicit definition of a problem space and heuristic search; the second is modeled through the interaction of appropriate speech-acts protocols and the extensive use of preferences. We refer to these two approaches as *deliberative* and *conceptual* inference strategies. The former is illustrated with a supermarket scenario, where the robot plays the role of an assistant, and the latter with a home scenario, where the robot plays the role of a butler, as described in Sections 8 and 9, respectively. The actors play analog roles in both settings, e.g., attending commands and information request related to a service task, and bringing objects involved in such requests or placing objects in their right locations, but each scenario emphasizes a particular aspect of the kind of support that can be provided by service robots.

The robot behaves cooperatively and must satisfy a number of cognitive, conversational, and task obligations, as follows:

- Cognitive obligations (*CO*):
    - update its KB whenever it realizes that it has a false belief;
    - notify the human user of such changes, so he or she can be aware of the beliefs of the robot;

- Conversational obligations: to attend successfully the action directives or information requests expressed by the human user;
- Task obligations (*TO*): to position the misplaced objects in their corresponding shelves or tables.

The cognitive obligations manage the state of beliefs of the robot and its communication to the human user. These are associated to perception and language and are stated for the specific scenario. Conversational and task obligations may have some associated cognitive obligations, too, that must be fulfilled in conjunction with the corresponding speech acts or actions.

In both the supermarket and home scenarios, there is a set of objects that belong to a specific class, e.g., food, drinks, bread, snacks, etc., and each shelf or table should hold objects of a designated class. Let $P_i = \{p_1, ..., p_j\}$, $Q_i = \{q_1, ..., q_j\}$ and $M_i = \{m_1, ..., m_j\}$ be the sets of observed, unseen/missing and misplaced objects, respectively, on the shelf or the table s_i in a particular observation at time $o_t = < s_i, P_t, Q_t, M_t >$ in relation to the current state of the KB. We assume that the behavior *see* inspects the whole shelf or table in every single observation, and these three sets can be computed directly. $M_t \subseteq P_t$ must hold, and all objects in $P_t - M_t$ should belong to the class associated to the shelf s_i.

Let M_{KB_i} be the set of objects of the class s_i that are believed to be misplaced in other shelves at the observation o_t and $Misplaced_{KB}$ the full set of believed misplaced objects in the KB at any given time. Let $Missing_{o_t}$ be $Q_t - M_{KB_i}$; i.e., the set of objects of the shelf's class that the robot does not know where are placed at the time of the particular observation o_t.

Whenever an observation o_t is made, the robot has the cognitive obligation of verifying whether it is consistent with the current state of the KB, and correct the false believes, if any, as follows:

1. For every object in $Missing_{o_t}$, state the exception in the KB, i.e., that the object is not in its corresponding shelf; notify the exception and that the robot does not know where such an object is!
2. For every object in M_t, verify that the object is marked in the KB as misplaced at the current shelf; otherwise, update the KB accordingly, and notify the exception.

The conversational obligations are associated to the linguistic interaction with the human user. For instance, if he or she expresses a fetch command, the robot should move to the expected location of the requested object, grasp it, move back to the location where the user is expected to be, and hand the object over to him or her. The command can be simple, such as *bring me a coke* or *place the coke in the shelve of drinks*; or composite, such as *bring me a coke and a bag of crisps*.

A core functionality of the GPSR is to interpret the speech acts in relation to the context and produce the appropriate sequence of behaviors, which is taken to be the meaning of the speech act. Such list of behaviors can also be seen as a schematic plan that needs to be executed to satisfy the command successfully. The general problem-solving strategy is defined along the lines of the GPSR as described above [50].

The task obligations are generated along the execution of a task, when the world is not as expected and should eventually be fixed. For instance, the $see(object_i)$ behavior produces, in addition to its cognitive obligations, the task obligations of placing the objects in the sets Q_t, M_t, and $Missing_{o_t}$ in their right places. These are included in the list *Pending_Task*.

All behaviors have a *status* indicating whether the behavior was accomplished successfully or whether there was an error, and in this latter case, its type. Every behavior has also an associated manager that handles the possible termination status; if the status is *ok*, the behavior's manager finishes and passes the control back to the dialogue manager or the GPSR dispatcher.

However, when the behavior terminates with an error, the manager executes the action corresponding to the status type. There are two main cases: (i) when the status can be handled with a recovery protocol and (ii) when inference is required. An instance of case (i) is the $move(shelf_j)$ behavior that may fail because there is a person blocking the robot's path, or a door is closed and needs to be opened. The recovery protocols may ask the person to move away and, in the latter situation, either ask someone around to open the door or execute the open-door behavior instead, if the robot does have such behavior in its behaviors library. An instance of case (ii) is when the *take* behavior, which includes a *see* behavior, fails to find the object in its expected shelf. This failure prompts the execution of the daily-life inference cycle.

Whenever the expectations of the robot are not met in the environment, the robot needs first to make a diagnosis inference and find a potential explanation for the failure; induce a decision dynamically as how to proceed and such decision becomes the goal for the induction and execution of a plan. The present model makes extensive use of diagnosis and decision-making and contrasts in this regard with models focus on planning mostly [34,44,63–65], as discussed in relation of the deliberative functions [33].

8. Deliberative Inference

This inference strategy is illustrated with the supermarket scenario in Figure 1. This has the following elements:

1. The supermarket consists of a finite set of shelves $S = \{s_1, ...s_n\}$ at their corresponding locations $L = \{l_1, ..., l_n\}$, each having an arbitrary number of objects or entities $O = \{o_1, ..., o_n\}$ of a particular designated class $c_i \in C$, the set of classes; for instance, $C = \{drinks, food, bread\}$;
2. The human client, who may require assistance;

3. The robot, which has a number of conversational, task, and cognitive obligations;
4. A human supermarket assistant whose job is to bring the products from the supermarket's storage and place them on their corresponding shelves.

The cognitive, conversational, and task obligations are as stated above. A typical command is *bring me a coke*, which is interpreted as [*acknowledge*, *grasp*(*coke*), *find*(*user*), *deliver*(*coke*, *user*)], where *grasp*(*coke*) is the composite behavior *kb_get_shelf_of_object*(*object$_i$*, *shelf$_j$*), *move*(*shelf$_j$*), *find*(*object$_i$*), and *take*(*object$_i$*).

In this scenario, the priority is to satisfy the customer, and a sensible strategy is to achieve the *take*(*object$_i$*) action as soon as possible and complete the execution of the command, and use the idle time to carry on with the *Pending_Task*. These two goals interact, and the robot may place some misplaced objects along the way if the actions deviate little from the main conversational obligation. If the sought object is placed at its right shelf, the command can be performed directly; otherwise, the robot must engaged in the daily-life inference cycle to find the object, take it, and deliver it to the customer. These conditions are handled by the behavior's manager of the behavior *take*, which, in turn, uses the behavior *see*, with its associated cognitive obligations.

The arguments of the inference procedure are:

1. The current *take*(*object$_i$*) behavior;
2. The list *Previous_Shelves* of shelves already inspected by the robot including the objects placed on them, which corresponds to the states of the shelves as arranged by the human assistant when the scenario was created, as discussed below in 8.1; this list is initially empty;
3. The set *Objects_Placed* of objects already put in their right locations by previous successful place actions performed by the robot in the current inference cycle; this set is initially empty.

The inference cycle proceeds as follows:

1. Perform a diagnosis inference in relation to the actual observations already made by the robot; this inference renders the assumed actions made by the human assistant when he or she filled up the stands including the misplaced objects *Misplaced$_{KB}$*;
2. Compute the *Decision* in relation to the current goal, e.g., *take*(*object$_i$*), and possibly other subordinated place actions in the current *Pending_Task*;
3. Induce the plan consisting of the list of behaviors *Plan* to achieved *Decision*;
4. Execute the plan in *Plan*; this involves the following actions:

   (a) update *Previous_Shelves* every time the robot sees a new shelf;
   (b) update the KB whenever an object is placed on its right shelf, and accordingly update the current *Pending_Task_Obligations*; and update *Objects_Placed*;
   (c) if the *object$_i$* is not found at its expected shelf when the goal *take*(*object$_i$*) is executed, invoke the inference cycle recursively with the same goal and the current values of *Previous_Shelves* and *Objects_Placed* which may not be empty.

8.1. Diagnosis Inference

The diagnosis inference model is based on a number of assumptions that are specific to the task and the scenario, as follows:

1. The objects, e.g., drinks, food, and bread products, were placed in their corresponding shelves by the human assistant who can perform the actions *move*(*s$_i$*)—move to *location l$_i$* of *shelf s$_i$* from its current location—and *place*(*o$_i$*), i.e., place *object$_i$* on the shelf at the current location. The assistant can start the delivery path at any of arbitrary shelf, can carry as many objects as needed in every move action, and he or she places all the objects in a single round.
2. The believed content of the shelves is stored in the robot's KB. This information can be provided in advance or by the human assistant through a natural language conversation, which may be defined as a part of the task structure.

3. If an object is not found by the robot in its expected shelf, it is assumed that it was originally misplaced in another shelf by the human assistant. Although, in actual supermarkets, there is an open-ended number of reasons for objects to be misplaced, in the present scenario, this is the only reason considered.

4. The robot performs local observations and can only see one shelf at a time, but it sees all the objects on the shelf in a single observation.

The diagnosis consists of the set of moves and placing actions that the human assistant is assumed to have performed to fill up all the unseen shelves given the current and possibly previously observed shelves. Whenever there are mismatches between the state of the KB and the observed world, a diagnosis is rendered by the inferential machinery (It should be considered that even the states of observed shelves are also hypothetical as there may have been visual precision and/or recall errors, i.e, objects may have been wrongly recognized or missed out; however, when this happens, the robot can recover only later on when it realizes that the state of the world is not consistent with its expectations, and it has to reconsider previous diagnoses.).

The diagnosis inference is invoked when the $see(object_j)$ at shelf s_i within the $take(object_j)$ behavior fails. The KB is updated according to the current observation, and contains the beliefs of the robot about the content of the current and the remaining shelves. The current observation $see(object_j)$ renders the set of missing objects $Missing_{o_t}$ and M_t of missing and misplaced objects at the current shelf s_i. If the sought object is of the class c_i of the shelf, it must be within $Missing_{o_t}$ or the supermarket has run out of such object; otherwise, the robot believed that the sought object was already misplaced in a shelf of a different class c_j, but the failed observation showed that such belief was false. Consequently, the sought object must be included in $Missing_{o_t}$, and the KB must be updated with the double exception in the KB: that the object is not in the current shelf and was not in its corresponding shelf; hence, it must be in on one of the shelves that remain to be inspected in the current inference cycle. This illustrates that negative propositions increase the knowledge productively, as the uncertainty is reduced.

The diagnosis procedure involves extending the believed content of all unseen shelves S_k with the content of $Missing_{o_t}$, avoiding repetitions. The content of the shelves seen in previous observations is already known.

There are many possible heuristics to make such an assignment; here, we simply assume that s_j is the closest unseen shelf—in metrical distance—to the current shelf s_i and distribute the remaining objects of $Missing_{o_t}$ in the remaining unseen shelves randomly. The procedure renders the known state of shelf s_i, unless there were visual perception errors, and the assumed or hypothetical states of the remaining unseen shelves.

The diagnosis is then rendered directly by assuming that the human assistant moved to each shelf and placed on it all the objects in its assumed and known states. There may be more than one known state because the assumption made at a particular inference cycle may have turned out wrong, and the diagnosis may have been invoked with a list of previous observed shelves in which states are already known.

8.2. Decision-Making Inference

In the present model, deciding what to do next depends on the task obligation that invoked the inference cycle in the first place *TO*, e.g., $take(object_i)$, and the current *Pending_Task*. Let the set *Potential_Decisions* $= TO \cup Pending\_Task$. Compute the set *Potential_Decisions*$_{subsets}$ consisting of all subsets of *Potential_Decisions* that include *TO*.

The model could also consider other parameters, such as the mood of the customer or whether he or she is in a hurry, which can be thought of as constraints in the decision-making process; here, we state a global parameter r_{max} that is interpreted as the maximum cost that can be afforded for the completion of the task.

We also consider that each action performed by the robot has an associated cost in time, e.g., the parameters associated to the behaviors *take* and *deliver*, and a probability to

be achieved successfully, e.g., the parameters associated to a *move* action. The total cost of an action is computed by a restriction function r.

The decision-making module in relation to $Potential\_Decisions_{subsets}$ proceeds as follows:

1. Compute the cost r_i for all sets in $Potential\_Decisions_{subsets}$;
2. *Decisions* is the set with maximal cost r_i such that $r_i \leq r_{max}$.

8.3. Plan Inference

The planning module searches the most efficient way to solve a set of *CO* and *TO*. Each element of $CO \cup TO$ implies a realignment in the position of the objects in the scenario, either carrying and object to another shelf or delivering it to a client.

Each *TO* is transformed in a list of basic actions of the form:
$[move(s_a), take(o_k), move(s_b), deliver(o_k)]$,
and each *CO* is transformed in a list of basic actions of the form:
$[move(s_a), take(o_k), search(client), deliver(o_k)]$,
where s_a is the shelf containing the object o_k according to the diagnosis module, and s_b is the correct shelf, where o_k should be according to the KB. All the lists are joined in a multiset of basic actions B.

The initial state S_0 of the search tree contains:

- The current location of the robot (l_k).
- The actual state of the right hand (free or carrying the object o_r).
- The actual state of the left hand (free or carrying the object o_l).
- The list R of remaining $CO \cup TO$ to solve.
- The multiset B of basic actions to solve the elements in R.
- The list of basic actions of the plan P (in this moment is still empty).

The initial state is put on a list F of all the not expanded nodes in the frontier of the tree. The search algorithm proceeds as follows:

1. Select one node to expand from F. The selection criteria of the node of F is DFS. The cost and probability of each action in the current plan P in the node is used to compute a score.
2. When a node S_i has been selected, a rigorous analysis of B is performed. For each basic action in B, check if the following preconditions are satisfied:
    - Two subsequent navigation moves are banned. If the action is a *move* or a *search(user)*, discard if the last action of P is a *move* or a *search(user)*.
    - Only useful observations. If the action is a *search(object)*, discard if the last action of P is a *search(object)*, *search(user)*, or if the robot actually has objects in both hands.
    - Only deliveries after taking. If the action is *deliver(o_i)*, the action *take(o_i)* should be included previously in the plan.
    - Only take actions if at least one hand is free.
3. For each basic actions of B not discarded using the preconditions, generate a successor node S_{ij} in this way:
    - If the basic action is *move(s)* or *search(user)* change the current location of the robot to s or the user position, respectively. If not, the current location of the robot in S_{ij} is the same as S_i.
    - Update the state of the right and left hand if the basic action is a *take* or a *deliver*.
    - If the basic action was a *deliver*, delete the associated element in the list R of remaining $CO \cup TO$. If the list gets empty, a solution has been found.
    - Remove the basic action used to create this node from B.
    - Add the basic action to the plan P.
4. Return to step 1 to select a new node.

When a solution has been found in the tree the plan *P* is post processed to generate a list of actions specified in terms of *SitLog* basic behaviors, which can be used by the dispatcher. A video showing a demo of the robot Golem-III performing as a supermarket assistant, including all the features described in this section is available at http://golem.iimas.unam.mx/inference-in-service-robots. The KB-system and the full Sitlog's code are also available at https://bit.ly/deliberative-inference.

We could have employed a ready-to-use planning systems for the implementation (e.g., Reference [65]) instead of the particular planning algorithm presented here, which we acknowledge is focused in the particular scenario, but our model is stated at the functional system level in opposition to particular algorithms that are used at the algorithmic level, and these two system levels should not be confused.

9. Conceptual Inference

This inference strategy is illustrated with a home scenario in which the robot plays the role of a butler, as follows:

1. The home has a number of rooms $R = \{r_1, ... r_n\}$; a finite set of shelves $S = \{s_1, ... s_n\}$ at their corresponding locations $L = \{l_1, ..., l_n\}$, which may be within any arbitrary room, each having an arbitrary number of objects $\{o_1, ..., o_n\}$ of a particular designated class $c_i \in C$, the set of classes;
2. The human user, who may require assistance;
3. The human user has a child that plays at home and misplaces objects on wrong tables;
4. The robot has a number of conversational, task, and cognitive obligations, which are the same as in the supermarket scenario, but, in addition, it may provide the user plausible explanations about the causes of unexpected observations;
5. The user has a number of preferences that are stored in the robot's KB, which can be changed dynamically; in addition, the robot may know about general health recommendations that can be deployed along the execution of a task.

The details of the implementation of the home scenario, including the use of the conditional defaults in the KB and the way they are managed, are explained in Appendix B. The roles played by the agents and the speech acts involved are analogous to those of the supermarket scenario. Additionally, in the home scenario, the robot is familiar with the human user, has knowledge about his or her preferences and/or habits, and may use this knowledge to assist him or her better. The robot's behavior is proactive, and the task relies on a more extensive use of cooperative dialogues, where the robot may have conversational initiatives. A task oriented conversation in this setting between the robot Golem-III and a human user named Luis is as follows:

1. **Golem** : *Hi, Luis.*
2. **Golem**: *Please tell me what do you like best: malz or coke?*
3. **Luis**: *Hi, Golem. I like malz.*
4. **Golem**: *Excellent! Malz is much healthier than the coke!*
5. **Golem**: *What do you like best: noodles or biscuits?*
6. **Luis**: *I like noodles best.*
7. **Golem**: *Great! I will recall your choice!*
8. **Golem**: *Do you have any more preferences?*
9. **Luis**: *No thanks, Golem; that is ok.*
10. **Golem**: *Thank you, Luis.*

At this point Luis goes to work and takes some time to be back; when he does Golem meets him at the door.

11. **Golem**: *Hi, Luis, how was your day?*
12. **Luis**: *I had a bad day!*
13. **Golem**: *Sorry to hear that; do you want me to do something for you?*
14. **Luis**: *Bring me something to drink and biscuits.*
15. **Golem**: *Ok. I will bring you malz, your favorite drink.*

16. **Luis**: *Yes.*
17. **Golem**: *But you like noodles better than biscuits! Shall I bring you the biscuits?*
18. **Luis**: *No, Golem, noodles are ok.*
19. **Golem**: *Ok. I will bring you the malz and the noodles. I will get the malz.*

Golem goes to the expected locations of the offered objects since they should be at their respective shelves, although there may be exceptions. So, Golem reaches the shelf of drinks where the malz should be placed and finds it.

20. **Golem**: *Attempting to grab the malz with my left arm.*

Golem grabs the malz.

21. **Golem**: *I took the malz.*
22. **Golem**: *I will get the noodles.*

Golem goes to the expected location of noodles, which is the table for food; however, the noodles are not there.

23. **Golem**: *The noodles are not in the shelf of food.*

Golem goes to the next preferred shelf for noodles, which is snacks; the noodles are found there.

24. **Golem**: *Attempting to grab the noodles with my right arm.*

Golem grabs the noodles;

25. **Golem**: *I took the noodles.*

Golem goes to the location where Luis is expected to be, which is the living room since he is back from work and tired after a bad day. Golem finds him there.

26. **Golem**: *Here is the noodles.*

Golem handles the noodles to Luis.

27. **Golem**: *Here is the malz.*

Golem handles the malz to Luis.

28. **Golem**: *I found the noodles on the shelf of snacks, but it should be on the shelf of food; do you want me to change the preferred location of noodles to the shelf of snacks?*
29. **Luis**: *Yes, Golem, please.*
30. **Golem**: *Ok. I updated my KB with your new preference.*
31. **Golem**: *I also noticed that the coke is not in its right place.*
32. **Golem**: *I think that the explanation for this is that the coke was misplaced there by your child.*
33. **Golem**: *Do you want me to take it to its right shelf?*
34. **Luis**: *Yes, Golem, please.*
35. **Golem**: *Ok. I will take it to the shelf of drinks.*

Golem goes to the shelf of snacks and takes the coke.

36. **Golem**: *Attempting to grab the coke with my right arm.*
37. **Golem**: *I took the coke.*

Golem goes to the shelf of drinks, places the coke there, and goes back to the living room with Luis.

38. **Golem**: *I put the coke on its right shelf.*
39. **Golem**: *All the objects are placed on their right shelves.*
40. **Golem**: *The task is finished.*
41. **Golem**: *Goodbye.*

This scenario could be implemented with the *GPSR*, too; however, here, we use an alternative strategy that offers an additional perspective of the framework. This is based on the direct specification of speech act protocols defined in *SitLog*. These are intentional structures in which performing a speech act establishes a number of conversational obligations that must be fulfilled before the dialogue proceeds to the next transaction. For instance, a command must be executed, and a question must be answered. The dialogue models are

designed considering the user's preferences and the whole task oriented conversation is modeled as the interpretation of one main protocol that embeds the goals of the task. The design of the dialogue models is loosely based on the notion of balanced transactions of the DIME-DAMSL. annotation scheme [66].

In the first section of the dialogue from (1) to (10), the robot asks for the user's preferences, and the KB is updated accordingly. The interpretation considers the user' utterances in relation to his or her current preferences and also in relation to other generic preferences that are stated in advance in the KB.

Utterances (11) to (19) consist on a speech act protocol to make and accept an offer. The protocol starts with greeting an open offer expressed by the robot in (11–13), which is answered with a user's request in (14); however, this is under-specified and vague; the robot resolves it using the user's preferences—his favorite drink—but also by contrasting the ordered food with the user's own food preferences, which results in a confirmation question in (17). The user changes his request, and the robot confirms the whole command in (18.19).

The robot executes the command with the corresponding embedded actions from (20) to (27). At this point, a new protocol is performed from (28) to (30) due to the task obligation that was generated when the robot noticed that an object—the noodles—was not placed on its corresponding shelf and asks for the confirmation of a user's preference.

Then, another protocol to deal with a new task obligation is performed from (31) to (32), including the corresponding embedded actions. This protocol involves an abductive explanation that is performed directly on the basis of the observation and the preference rule used backwards, as explained above in Section 6. This protocol is concluded with a new offer that is accepted and confirmed in (33–35). The new task is carried out as reported in (36–38). The task is concluded with the final protocol performed from (39) to (41).

The speech acts and actions performed by the robot rely on the state and dynamic evolution of the knowledge. The initial KB supporting the current dialogue is illustrated in Figure 6 and its actual code available at Listing 3. In it, the preferences are written as conditional defaults (e.g., `bad_day=»tired`), which are considered to successfully interact with the user and to achieve abductive reasoning. As the demo is performed, some new elements are defined in the KB, such as the properties `back_from_work` and `ask_comestible` added to the individual user. Later in the execution of the task, such properties play an important role determining the preferences of the user.

Listing 3. KB with preferences.

```
[ 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],
[moved_by=>child=>>misplaced,1], [moved_by=>partner=>>misplaced,2] ], [], []),
class(comestible, object, [], [], []),
class(food, comestible, [ ['-'=>>loc=>shelf_food,2],
['-'=>>loc=>shelf_snacks,3],['-'=>>loc=>shelf_drinks,4],
[last_seen=>'-'=>>loc=>'-',1] ], [],
[[id=>noodles, [], []], [id=>bisquits, [], []]]),
class(drink, comestible,[ ['-'=>>loc=>shelf_drinks,2],
['-'=>>loc=>shelf_snacks,3],['-'=>>loc=>shelf_food,4],
[last_seen=>'-'=>>loc=>'-',1] ], [],
[[id=>coke, [], []], [id=>malz, [], []]]),
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]],[]]
]) ]
```
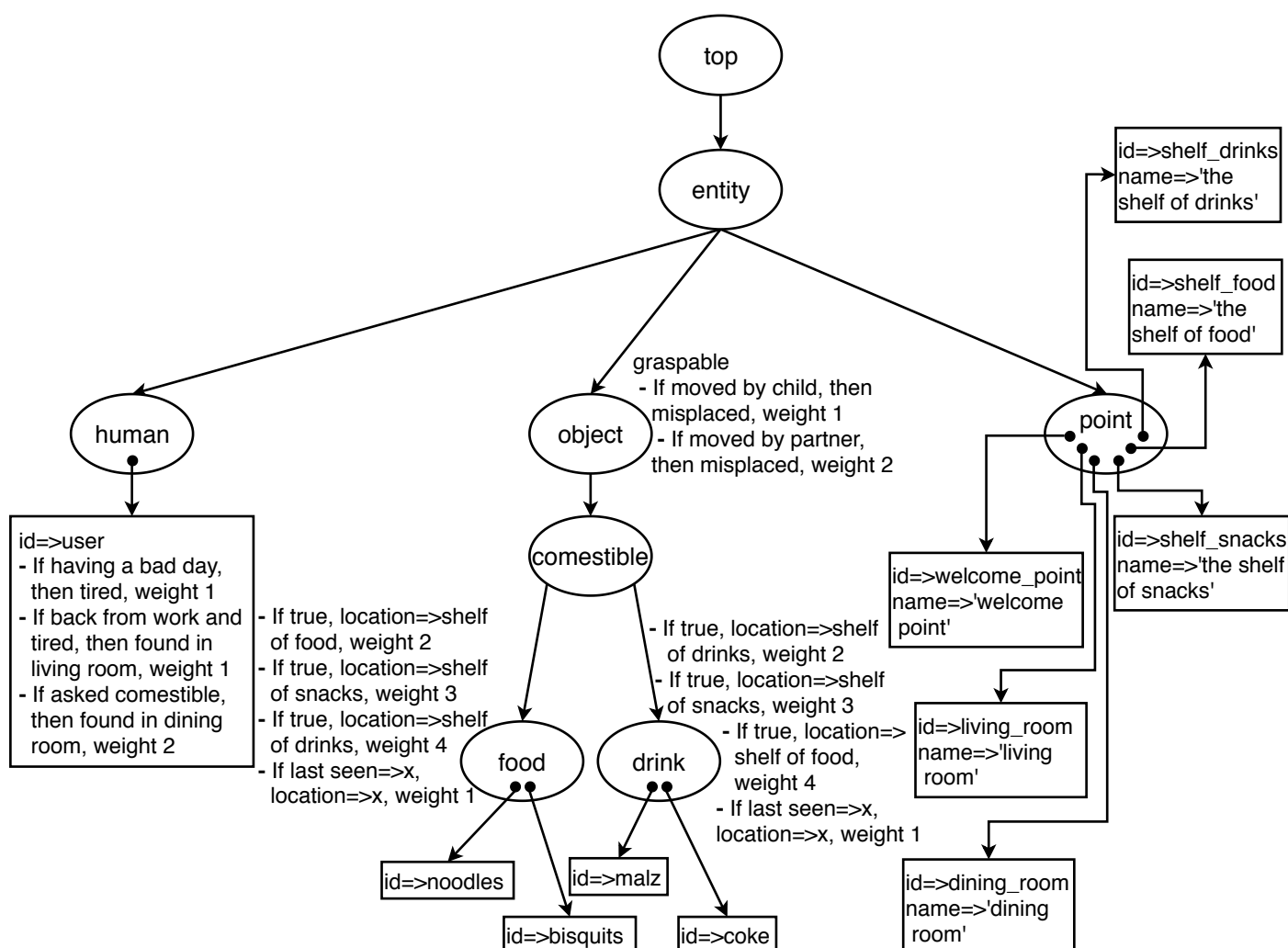
Figure 6. KB with preferences.

The daily-life inference cycle is also carried on in this scenario, although it surfaces differently from its explicit manifestation as a pipe-line inference sequence.

As in the deliberative scenario, a diagnosis inference emerges when the expectations of the robot are not met in the world, although, in the present case, such a failure creates a task obligation that will be fulfilled later, such as in (28) and (31–33). However, instead of producing the whole set of actions that lead to the observed state, the robot focuses only on the next action or on producing the abductive explanation directly from the observed fact and the corresponding KB-Service, as in (32).

In this setting, there is also an implicit diagnosis that is produced from continuously verifying whether there is a discrepancy between the user's manifested beliefs and intentions, and the preferences in the KB. For instance, this form of implicit diagnosis underlies utterances (5) and (17).

The decision-making in this setting is also linked to the conversational structure and the preferences in the KB. Decisions are made on the basis of diagnoses, and have the purpose of reestablishing the desired state of the world, or to make the world and the KB consistent with the known preferences, and the robot makes suggestions to the user who is the one who makes the actual decisions, as in (28,29) and (33,34).

The planning inference is also implicit, as the robot has the obligation to perform the action that conforms with the preferences, as when it inspects the shelves looking for

objects in terms of their preferred locations, as in (23) and its associated previous and following actions.

The conceptual inference strategy relies on an interplay between the structure of speech acts transactions and the preferences stored in the KB, and it avoids the explicit definition of a problem space and heuristic search. The inferences are sub-optimal and rely on the conversational structure, a continuous interaction between the language and the KB, and the interaction with the world.

A video showing a demo of the robot Golem-III as a home assistant performing the task oriented conversation (1–41) is available at http://golem.iimas.unam.mx/inference-in-service-robots. The corresponding KB and *SitLog* dialogue models are available at https://bit.ly/conceptual-inference.

This demo is still placed on the kind of tasks that are common in robotics competitions, but we hope that the present framework and methodology can be applied to develop practical fully autonomous applications of the kind studied in socially assistive robotics [2,44,56,67–69].

10. Conclusions and Further Work

In the paper, we reviewed the framework for specification and development of service robots that we developed over the last few years. This framework includes a conceptual model for service robots, a cognitive architecture to support it, and the *SitLog* programming language for the declarative specification and interpretation of robotics task structure and behaviors. This language supports the definition of speech acts protocols that the robot performs during the execution of the task, fulfilling implicitly the objectives of goal-oriented conversations.

We also presented a non-monotonic knowledge-base system for the specification of terminological and factual knowledge in robotics applications. The approach consists of the definition of an strict taxonomy that support defaults and exceptions, which can be updated dynamically. Conflicts of knowledge are resolved through the principle of specificity, and contingent propositions have an associated weight. The system allows the specification of preferences that are employed in the reasoning process and can provide plausible explanations about unexpected facts that are realized by the robot while performing the task.

The present framework allows us to model service robotics tasks through the definition of speech acts protocols; these protocols proceed while the expectations of the robot are met in the world. However, whenever no expectation is satisfied in a particular situation, the ground is lost, the robot gets out of context, and cannot proceed with the task. Such a contingency is met with two strategies: the first consists of invoking a recovering protocol, in which the purpose is to restore the ground through interacting with other agents or the world; the second consists of resorting to symbolic reasoning, or thinking, by invoking and executing the daily-life inference cycle.

This cycle is studied through two different approaches: the first consists of the pipeline implementation of a diagnosis, a decision-making, and a planning inference, and it involves the explicit definition of a problem-space and heuristics search; and the second consists of the definition of the tasks in terms of speech act protocols that are carried on cooperatively between the robot and the human user, in which the ground is kept through the intense use of preferences stored in the robot's KB, which are deployed along the robotics tasks.

We illustrated these two approaches with two fully detailed scenarios and showed how these are deployed in real-time in a full autonomous manner by the robot Golem-III. In the former, the robot performs as a supermarket assistant, and, in the latter, it performs as a butler at home.

The deliberative inference scenario is structured along the lines of the traditional symbolic problem-solving strategy and renders explicitly the three main stages of the daily-life inference cycle. Inference is conceived as a problem of optimization, where the chosen diagnosis, decisions, and plans are the best solutions that can be found given

the constraints of the task. The methodology is clear and highlights the different aspects of inference.

However, the three kinds of inferences are carefully designed and programmed beforehand; the methods and algorithms are specific to the domain; and it is unlikely that a general and domain independent set of algorithms can be developed. The method adopts a game playing strategy, and the interaction with the human-user is reduced to listening to the commands and performing them in long working cycles. The conversational initiative is mostly on the human side, and the robot plays a subordinated role. For these reasons, although the strategy may yield acceptable solutions, it is somehow unnatural and reflects poorly the strategy employed by people facing this kind of problems in similar kind of environments. These are typical scenarios of the RoboCup@Home competition, particularly of the so-called *General Purpose Service Robot* (GPSR), which has a strong robotics orientation. This scenario illustrates the nature of service tasks from the perspective of the community focused on developing robotics devices and algorithms.

The conceptual strategy carries on with the three kind of inference but implicitly, based on informed guesses that use the preferences stored in the KB. In this latter approach, the ground is not broken when the robot realizes that the world in not as expected, and the robot does not perform an explicit diagnosis, decision-making, and planning inferences; the focus is rather on what is the closer world or situation in which the current problem can be solved and act accordingly. This approach renders much richer dialogues than the pipe-line strategy, where the inference load is shared between the two agents. The robot makes informed offers on the basis of the preferences, and the human user makes the choices; but, the robot can also make choices, which may be confirmed by the user, and the robot takes conversational initiatives to a certain extent. Overall, the task is deployed along a cooperative conversation through the deployment of a speech acts protocol, which makes an intensive use of the knowledge and preferences stored in the KB, and the goals of the task are fulfilled as a co-lateral effect of carrying on with such protocols. In this approach, the robot does not define a dynamic problem space and greatly limits heuristic search, as the uncertainty is captured in the preference. The strategy has a strong AI component and may be applied in the scenarios focused by the socially assistive robotics community.

Deliberative and conceptual inferences, and the style of interaction that they render, are specified in our conceptual model and implemented in our framework. We hope that the methods and tools developed in our project foster a closer interaction between AI and robotics.

Although, at the present time, the speech act protocols are specific for the task, we envisage the development of generic protocols that can be induced from the analysis of task oriented conversation, which can be instantiated dynamically with the content stored in the KB and the interaction with the world, and the approach can be made domain independent to a larger extent than the present one, for instance, by providing abstract speech acts protocols for making offers, information, action requests, etc. However, for the moment, this enterprise is left for further work.

Author Contributions: Conceptualization, L.A.P.; Formal analysis, L.A.P., N.H., A.R., and R.C.; Funding acquisition, L.A.P.; Investigation, L.A.P. and G.F.; Methodology, L.A.P.; Project administration, L.A.P.; Software, N.H., A.R., and R.C.; Writing—original draft, L.A.P.; Writing—review & editing, L.A.P., N.H., A.R., R.C., and G.F. All authors have read and agreed to the published version of the manuscript.

Appendix A. An Example Program in *SitLog*

In order to show the expressive power of *SitLog*, the full code of the program in Figure 3 is provided. A DM is defined as a clause with three arguments as follows in Listing A1.

Listing A1: Definition of Dialogue Models

```
diag_mod(id(Arg_List), Situations, Local_Vars).
```

The first argument is an atom (i.e., the DM's name or Id) or a predicate in which case the functor is the DM's ID and the list arguments are the arguments of the DM, which are visible within its body; the second is the list of situations; and the third the list of local variables. A situations is defined as a set of attribute-value pairs as was mentioned; situation IDs need not be unique, and different instances of the same situation with the same ID but different arguments or values can be defined.

Listings A3 and A4 include the clauses with the definitions of *main* and *wait* of the program illustrated in Figure 3. The value of the input pipe is initialized by the value provided in the first occurrence of out_arg and the global variables are declared as a global parameter of the application as follows in Listing A2:

Listing A2: Specification of Global Variables

```
Global_Vars = [g_count_fs1 ==> 0,
g_count_fs2 ==> 0].
```

The DM *main* includes the list of the three situations (see Figure 3) (In this *SitLog* program, neither the DM nor the situations use parameters; these are illustrated in the DMs representing the demos task structure and behaviors). The situation is has an application specific type, in this case speech. There is a specific perceptual interpreter for all user-defined types. These interpreters specify the interface between the expectations of the situation and the low level recognition processes. The type speech specifies that the expectation of the situation will be input through the speech modality. The notation of expectations is defined in the corresponding perceptual interpreter which instantiates the current expectations and returns the one satisfied in the current interpretation situation.

The is situation includes a local program—defined by the prog attribute—consisting of the *SitLog*'s operator inc that increases by one the value of the local variable count_init each time the situation is visited during the execution of the *main* DM. Its arcs attribute is a list with the specification of its three exit edges. Each exemplifies a kind of expectation: a concrete one (i.e., finish), a list including one open predicate (i.e., day(X)) and a complex expression defined as the list with the value of the local variable day and the application of the function f to the value of the input pipe (i.e., [get(day,Day),apply(f(X),[In_Arg])]); in the function's application the Prolog's variable *X* gets bounded to the current value of *In_Arg*. The definition of f is given in Listing A5. As can be observed, the value of function f is *ok* or *not ok* depending on whether the value of the input pipe is the same or different from the current value of the local variable day.

Listing A3: *SitLog*'s Specification of the DM *main*.

```
% Main Dialogue Model
diag_mod(main,
%Second argument: List of Situations
[% Initial situation
[id ==> is,
type ==> speech,
in_arg ==> In_Arg,
out_arg ==> apply(when(If,True,False),
[In_Arg=='monday','tuesday','monday']),
% Local program
prog ==> [inc(count_init,Count_Init)],
arcs ==>
[% Examples of Grounded forms
finish:screen('Good Bye') => fs,
% Example of predicate expectation and action
[day(X)]:[date(get(day,Y)),  next_date(set(day,X))] => is,
% Example of functional specification of
% expectation, action and next situation
[get(day,Day),apply(f(X),[In_Arg])]: [apply(g(X),[_])] =>
apply(h(X,Y),[In_Arg,Day])
]
],
% Second Situation
[id ==> rs,
type ==> recursive,
prog ==> [inc(count_rec, Count_Rec)],
embedded_dm ==> sample_wait,
arcs ==> [fs1:screen('Back to initial sit') => is,
fs2:screen('Cont. recursive sit') => rs]
],
% Final Situation
[id ==> fs, type ==> final]
], % End list of situations
% Third Argument: List of Local Variables
[day ==> monday, count_init ==> 0, count_rec ==> 0]
). %End DM (main)
```

Listing A4: *SitLog*'s Specification of the DM *wait*.

```
% Second Dialogue Model
diag_mod(wait,
% Second argument: List of Situations
[% First situation
[id ==> is,
type ==> speech,
in_arg ==> In_Arg,
arcs ==>
[
In_Arg:[inc(g_count_fs1, G1)] => fs1,
loop:[inc(g_count_fs2, G2)] => fs2
]
],
% Final Situation 1
[id ==> fs1, type ==> final],
% Final Situation 2
[id ==> fs2, type ==> final]
], % End List of Situations
% Third argument: local variables (empty)
[ ]
). % End DM (wait)
```

Each arc of `is` illustrates also a particular kind of action: `screen('Good Bye')` is a speech act that renders the expression *Good Bye* when the `finish` expectation is met. The predicate `screen` is defined as a *SitLog*'s basic action and has an associated algorithm that is executed by IOCA when it is interpreted, when its argument is rendered through

speech (i.e., the robot says 'Goodbye'). The second edge illustrates a composite action: the list [date(get(day,Y)),next_date(set(day,X))], where date and next_date are user defined predicates, as opposed to *SitLog*'s basic actions, and get and set are *SitLog*'s operators that consult and set the local variable day. When the corresponding expectation is met, these operators are executed, and the action is grounded as the list of the two predicates with their corresponding values and is available for inspection in the history of the task, as explained below. Finally, the action in the third edge illustrates the application of the function g that consults the last grounded edge traversed in the history of the task, and the action's value is the specification of such transition; the definition of g is given in Listing A5.

Listing A5: User functions of the dummy application.

```
%Example of user functions structure
f(X) :- var_op(get(day, Day)),
(X == Day -> Y = ok |
otherwise -> Y = 'not ok'),
% Assign function value
assign_func_value(Y).
% Example of function consulting the history
g(_) :- get_history(History),
get_last_transition(History,Last),
% Assign function value
assign_func_value(Last).
% Example of next state selection function
h(X, Y) :- (X == Y -> Next_Sit = is |
otherwise -> Next_Sit = rs),
% Assign function value
assign_func_value(Next_Sit).
```

The first two arcs illustrate the concrete specification of next situations, fs and is, respectively, and the third one shows the functional specification of the next situation through the function h, in which the arguments are the current input pipe value and the current value of the local variable day, the latter value is conveyed in the Prolog's variable Day. The definition of h is given in Listing A5, too.

User functions are defined as standard Prolog programs that can access the current *SitLog*'s environment (i.e., the local and global variables), as well as the history of the task through *SitLog*'s operators, in which execution is finished with the special predicate assign_func_value(Next_Sit), as can be seen in Listing A5.

The conceptual and deliberative resources used on the demand during the interpretation of situations are defined as user functions. There is a set of user functions to retrieve information and update the content and structure of the knowledge-base service, and also to diagnose, make decisions and induce and execute plans during the interpretation of situations and dialogue models.

The second situation rs is of type recursive. It also has a local program that increments the local variable count_rec each time the corresponding embedded DM *wait* is called upon. This DM is specified by the attribute embedded_dm. Recursive situations consists of control information, and the arcs attribute includes only the exit edges, that depend on the final state in which the embedded DM terminates; the expectation of each arch is an atom with the name of the corresponding final situation of the embedded DM (i.e., fs1 and fs2). The corresponding screen actions render a messages through speech, as previously explained.

Final situations do not have exit edges and are specified simply by their ids and the designated type final. When a situation of this type is reached in the main DM the whole *SitLog*'s program is terminated; otherwise, when a final situation of an embedded DM is

reached, the control is passed back to the embedding DM, which is popped up from the DM's stack.

Finally, the third argument of the *main* DM is the list of its local variables [day ==> monday, count_init ==> 0, count_rec ==> 0]. As was mentioned these variables are only visible within *main* and are outside the scope of *wait*. Hence, in the present environment DMs can see their local varibales and the global variables defined for the whole *SitLog* application, but local variables are not seen in embedded DMs. This locality principle has also proved to be very helpful for the definition of complex applications.

The definition of the embedded DM *wait* proceeds along similar lines. The initial situation is is of type speech, too. It defines two arcs, the expectation of the first is the value of the input pipe and that of the second arc is the atom loop. If the speech interpreter matches the input pipe the global variable g_count_fs1 is incremented, the final state fs1 is reached and the execution of the *wait* DM is terminated. Otherwise, the external input turns out to be unified with tha atom loop. When this latter path is selected the global variable g_count_fs2 is incremented, the final state fs2 is reached and the execution of the *wait* DM is terminated. Finally, the list of local variables of *wait* is empty, and this DM can only see global variables. Noticeably, since the out_arg attribute is not set in the current DM, the input pipe of the *main* DM propagates all the way back to the reentry point of the situation rs, that invokes the embedded DM wait.

We conclude the presentation of this dummy program with the history of an actual task, which is illustrated in Listing A6. The reader is invited to trace the program and the expectations that were met at each situation, with their corresponding next situations. The story of the whole task is provided by the *SitLog* interpreter when its execution is finished. The full Prolog's code of the *SitLog*'s interpreter is available as a GitHub repository at https://github.com/SitLog/source_code.

<div align="center">Listing A6: History of a session.</div>

```
main: (is,[day(tuesday)]:
[date(monday),next_date(tuesday)])
main: (is,[tuesday,'not ok']:
([day(tuesday)]:
[date(monday),next_date(tuesday)]))
[wait: (is,loop:[1])
wait: (fs2,empty:empty)]
main: (rs,fs2:
screen(Cont. recursive sit))
[wait: (is,tuesday:[1])
wait: (fs1,empty:empty)]
main: (rs,fs1:
screen(Back to initial sit))
main: (is,[day(monday)]:
[date(tuesday),next_date(monday)])
main: (is,[monday,ok]:
([day(monday)]:
[date(tuesday),next_date(monday)]))
main: (is,finish:screen(Good Bye))
main: (fs,empty:empty)

Out Arg: monday
Out Global Vars: [g_count_fs1==>1,
g_count_fs2==>1]
```

Appendix B. Conceptual Inference Scenario

The human experience and the robot's performance is improved during the execution of a task by the robot knowing the things the user likes, social patterns, healthy guidelines, etc. Such aspects can be expressed in the KB as preferences, or conditional defaults, that helps resolving conflicting situations arising from incompatible conclusions. Let *CD* be the set of conditional defaults:

$$
\begin{aligned}
\{[Ant_{11}, Ant_{12}, \ldots, Ant_{1i} &\rightarrow Con_1, W_1], \\
[Ant_{21}, Ant_{22}, \ldots, Ant_{2j} &\rightarrow Con_2, W_2], \\
&\vdots \\
[Ant_{n1}, Ant_{n2}, \ldots, Ant_{nm} &\rightarrow Con_m, W_m]\},
\end{aligned}
$$

where each element in *CD* is of the form $[Antecedents \rightarrow Consequent, Weight]$, with *Antecedents* a list such that the *Consequent* appended to *Antecedents* compose a list of either properties or relations alone. Furthermore, assume that at some point in the execution of the task all antecedents are satisfied for some (more than one) conditional defaults; therefore, the corresponding consequents are also satisfied, which may cause a problem since they might be representing incompatible conclusions. This problem is solved by the Principle of Specificity applied to the weight of the conditional defaults; thus, only one consequent will be considered, the one in which its associated weight is the lowest. The structure of the conceptual inference scenario can be broken up in three parts: (i) *retrieving user preferences and getting the order*, (ii) *fetching and delivering items*, and (iii) *updating the KB and applying abductive reasoning*; each one will be explained in detail next.

Appendix B.1. Retrieving User Preferences and Getting the Order

The preferences of the human user, and all relevant information for a successful interaction, should be present in the KB. This is likely to be a dynamic process since user preferences, healthy guidelines, designated home locations, items in the shelves, and so on may greatly vary from time to time. One way to keep the KB updated, probably the optimal way, is to directly querying the user. For example, if there are k different new drinks, the robot proceeds by repeatedly taking two drinks at a time and asking the user to choose the preferred one that he or she would like to be served; thus, the total number of queries is $O(k^2)$ to get the appropriate weight of all drinks with respect to the user preference.

Once the preferences are known to the robot, it can make use of them in the course of the daily routine to reason about the state of its surroundings, the conduct of the user and the speech acts it is faced with. In the present scenario, the robot offers its assistant to the user, who replies by asking for comestible objects $o_1, \ldots, o_l$ to be fetched to him. For each object o_i, the robot examines whether o_i is the preferred object to be served among the individuals of its class, C_{o_i}. If so, o_i is added to the list L_{final} of final objects to be delivered. Otherwise, the preferred object to be served o_{pref} of the class under consideration C_{o_i} is obtained, and the user is queried to choose between the object he originally asked for o_i and the preferred one o_{pref}. The user's choice is added to L_{final}.

It can be noticed that getting the preferred member of a class C is an important operation. Recall that preferences are conceived as conditional defaults bound to their weight, so the lower the weight the higher its preference. The steps involved in finding the preferred value of a property or relation defined in the class C are:

1. Retrieve from the KB the list of conditional defaults defined within C and its ancestor classes.
2. Let ℓ_{sorted} be the result of sorting in increasing order the list generated in the previous step. The key to sort this list is the weight value defined within the conditional defaults.
3. For each conditional default in ℓ_{sorted}, verify if its antecedents are satisfied. In that case, keep its consequent, which is a property or relation. Otherwise, dismiss the conditional default. Then, delete from left to right consequents that define a property

or relation more that once, preserving the first occurrence. Let ℓ_{del} be the list that is obtained after this deletion.

4.   In ℓ_{del}, find the property or relation of interest in which value is the desired output.

For the situation described above, the preferred object is sought as the argument of the property *to serve*, occurring in the consequent of the conditional defaults present in the class C_{o_i}.

Interestingly, the robot can adequately deal with user commands that are underspecified, i.e., commands asking for an individual object but missing specific information that uniquely identifies it, providing instead general information. For instance, *bring me something to drink*. The robot deals with this kind of commands by taking the preferred individual of the class being asked.

Therefore, at the end of the speech act, whether the user requests objects by name or by giving general information, the robot is able to formulate the final list L_{final} of objects to be fetched to the user.

Appendix B.2. Fetching and Delivering Items

For each $o_i \in L_{final}$, the robot queries the KB to retrieve the list $[loc_1, \ldots, loc_n]$ of preferred locations where o_i is likely to be found, such that loc_1 is the most preferred location and loc_n is the least preferred location. Furthermore, $[loc_1, \ldots, loc_n]$ is a permutation of the locations $l_1, \ldots, l_n$ in L (see the settings of the conceptual inference in Section 9).

Obtaining the list of preferred locations of an object is an operation closely related to the operation outlined above that finds the preferred member of a class. The steps are:

Step 1   retrieves not only the conditional defaults of the object's class and its ancestors, but also the conditional defaults of the object itself.

Step 2   is the same of that to find the preferred member of a class.

Step 3   keeps the consequent of conditional defaults in which antecedents are satisfied but does not delete any of them, although a property or relation may be defined multiple times.

Step 4   subtracts in order all the values of the property or relation of interest, thus producing the desired list.

For the list of preferred locations needed by the robot, the property of interest in step 4 is the location, or *loc* as it is defined in the KB.

Next, the robot visits the shelves at the locations $[loc_1, \ldots, loc_n]$ in their order of appearance, searching for the object o_i in each of them. If the object o_i is found at loc_j, the robot takes it, and repeats now the process for the object o_{i+1} in L_{final}. If o_i is not found at loc_j, the robot searches for it in the shelf located at loc_{j+1}. When an error arises taking an object, moving to a new location, or realizing that the object is not found after visiting all shelves, then a recovery protocol can be invoked or the daily life inference cycle triggered.

At this point two important observations have to be made:

1.   As a side effect of searching for an object, the properties of other objects in the robot's KB may change. Suppose that the robot makes an observation in the shelf located at loc_j trying to find object o_i. Regardless of recognizing o_i, the robot may have seen a set $O = \{o_{i1}, \ldots, o_{i\bar{n}}\}$ of objects. Hence, the robot is now aware of the precise shelf where such objects are placed. So, the property *last seen* for these objects is assigned to loc_j in the KB. Therefore, for any $obj \in O$ the first element of its list of preferred locations has to be loc_j. The KB works in this way since a conditional default for the object obj is defined with antecedent *last seen*, consequent *loc* and weight 1, as it is seen in Figure 6.

2.   When the robot takes two objects, using its two hands, or when it is holding one object and there are no more left to take, the objects must be delivered to the user. But first, the robot needs to determine the room where the user may have gone based on his or her preferences and properties. The preferred room is retrieved from the KB

by an operation that can be derived from the steps explained above. In the current scenario, two conditional defaults for the individual *user* have been defined in which consequent is the property *found in* that indicates the room where the user is located, and in which antecedents are conditions that cause him or her to go to one room or to another depending on the user's mood or physical state, as shown also in Figure 6. After this delivery, the robot examines the list L_{final} to know whether there are more objects to be fetched or not.

Noticeably, the inference mechanism on conditional defaults handles chained implications, so it is plausible to have a conditional default such that its antecedents may be satisfied by the consequents of other conditional defaults. Let *Prop* be the list of known closed propositions (properties or relations) and L_{CD} be the list $[Ant \rightarrow Consequent \mid More_{CD}]$ of conditional defaults of a class or individual, such that the conditional defaults in L_{CD} are sorted in increasing order with respect to their weight, value that is omitted in L_{CD}. The mechanism proceeds recursively over the head element $Ant \rightarrow Consequent$ of L_{CD} according to the cases:

- *Ant* is a single property or relation. Examine all valid pattern matching situations for $Ant \rightarrow Consequent$ as follows: (a) Since *Ant* and *Consequent* are both properties or both relations, their pattern is $\alpha => \beta$; nonetheless, β may be a variable, and the pattern becomes $\alpha =>'-'$. (b) A property may be a single label with no associated value. (c) *Ant* may be absent, which is represented as $'-' \rightarrow Consequent$, indicating that the consequent of the conditional default is always satisfied.
  1. Now, execute a backward analysis checking whether *Ant* is already part of *Prop*; if so, add the corresponding *Consequent* to *Prop*.
  2. Otherwise, execute a forward analysis verifying if *Ant* occurs as the consequent of a conditional default in $More_{CD}$; in that case, apply the current analysis to the list $More_{CD}$, in which output is a temporary set of new closed propositions Tem_{Prop}, such that *Consequent* is added to *Prop* whenever *Ant* is part of Tem_{Prop}.

  Remarkably, matching *Ant* with an element of *Prop* or Tem_{Prop} instantiates the variable that *Ant* might have. Since the variable in the antecedent is bound in the consequent, instantiating the variable in *Ant* provides a value for the variable in *Consequent*.
- *Ant* is a list of properties or relations. Check whether the elements in the list *Ant* are part of *Prop* or Tem_{Prop}, as explained above. If that is true for all elements in *Ant*, then add *Consequent* to *Prop*.

Finally, the desired property or relation is searched for in the resulting list *Prop*, and its first occurrence is output.

This mechanism is illustrated in our scenario as the user is assumed to be home after a bad day at work and to have requested comestible objects. Therefore, the first conditional default for the individual *user* implies that he or she is also tired, this consequent is chained to the next implication since being tired and back from work implies that the user is found at the living room; but having requested comestible objects implies that he or she is in a different room. The conditional default concluding that the user is in the living room has lower weight, so that room is the preferred one where the user may be found.

Appendix B.3. Updating the KB and Applying Abductive Reasoning

Once the robot reaches the user, it hands over the objects it is carrying. At this point, the robot knows the location from where such objects were taken. For each delivered object o_{del}, if there is an inconsistency between the preferred location to find o_{del} as stated in the KB, loc_{pref}, and the actual location where o_{del} was taken, loc_{act}, then the robot informs the user of this situation and asks him to choose the preferred location between loc_{pref} and loc_{act}. The data for o_{del} in KB is updated when the user picks loc_{act} over loc_{pref}.

After delivering all requested objects, the robot examines the location of other objects that it may have seen during the execution of the whole task. As described above, for the observed objects the property *last seen* in the KB is updated with the location where

they were last seen. Let $O' = \{o'_1, \ldots, o'_t\}$ be the set of non-requested objects seen while executing the task. For each $o'_i \in O'$:

- The robot retrieves from the KB the value of the property *last seen* for o'_i, denoted loc_{last}, and the list $[loc_1, loc_2, \ldots, loc_n]$ of preferred locations where o'_i is likely to be found. Now, loc_{last} and $[loc_1, loc_2, \ldots, loc_n]$ are examined to determine any inconsistency. In fact, $loc_{last} = loc_1$ since the location where o'_i was last seen has the highest preference; loc_2 is next as the predefined location where o'_i is more likely to be found. If $loc_{last} \neq loc_2$, then a problem is detected having seen o'_i not in its predefined location. Thus, the robot defines the property *misplaced* for o'_i in the KB.

- The abductive reasoner is triggered to find out a possible explanation for the misplacement of o'_i. This reasoner takes the lists L_{CD} and *Prop*, as defined previously, and recursively examines the pattern for each conditional default $Ant \rightarrow Consequent$ in L_{CD} similarly to the inference mechanism described above, but *Consequent* is checked to belong to *Prop* instead of *Ant*. If this check turns out to be true, then the pair $Consequent : Ant$ is added to the list of explanations. After all conditional defaults are analyzed, the list of explanations is trimmed by keeping the first occurrence of a pair with a given *Consequent* and removing all others. This respects the order of preference since the explanation drawn from the conditional default with lowest weight is kept.

- The application of the abductive reasoner on the current scenario reveals, by the conditional defaults defined in the class *object*, that o'_i is misplaced because it was moved by the user's child or by the user's partner. The weight associated to each conditional default is considered to conclude that o'_i is misplaced because the user's child moved it to loc_{last}. Finally, if allowed by the user, the robot goes to loc_{last}, takes o'_i and places it in loc_2. After this, the robot is finished examining o'_i.

References

1. RuleBook for RoboCup @Home. Available online: https://github.com/RoboCupAtHome/RuleBook/ (accessed on 23 January 2021).
2. Matarić, M.J.; Scassellati, B. Socially Assistive Robotics. In *Springer Handbook of Robotics*; Springer International Publishing: Cham, Switzerland, 2016; pp. 1973–1994.
3. Durham, J.; Bullo, F. Smooth Nearness-Diagram Navigation. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Nice, France, 22–26 September 2008; pp. 690–695.
4. Grisetti, G.; Stachniss, C.; Burgard, W. Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. *IEEE Trans. Robot.* **2007**, *23*, 34–46. [CrossRef]
5. Chitta, S.; Jones, E.G.; Ciocarlie, M.; Hsiao, K. Mobile Manipulation in Unstructured Environments: Perception, Planning, and Execution. *IEEE Robot. Autom. Mag.* **2012**, *19*, 58–71. [CrossRef]
6. Srinivasa, S.; Berenson, D.; Cakmak, M.; Collet Romea, A.; Dogar, M.; Dragan, A.; Knepper, R.A.; Niemueller, T.D.; Strabala, K.; Vandeweghe, J.M.; et al. HERB 2.0: Lessons Learned from Developing a Mobile Manipulator for the Home. *Proc. IEEE* **2012**, *100*, 1–19. [CrossRef]
7. Collet, A.; Martinez, M.; Srinivasa, S.S. The MOPED framework: Object Recognition and Pose Estimation for Manipulation. *Int. J. Robot. Res.* **2011**, *30*, 1284–1306. [CrossRef]
8. Espinace, P.; Kollar, T.; Roy, N.; Soto, A. Indoor scene recognition by a mobile robot through adaptive object detection. *Robot. Auton. Syst.* **2013**, *61*, 932—947. [CrossRef]
9. Rajan, K.; Saffiotti, A. Towards a science of integrated AI and Robotics. *Artif. Intell.* **2017**, *247*, 1–9. [CrossRef]
10. Lotzsch, M.; Risler, M.; Jüngel, M. XABSL—A Pragmatic Approach to Behavior Engineering. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Beijing, China, 9–15 October 2006; pp. 5124–5129.
11. Tousignant, S.; Van Wyk, E.; Gini, M. An Overview of XRobots: A Hierarchical State Machine-Based Language. In Proceedings of the Workshop on Software Development and Integration in Robotics (SDIR), Shanghai, China, 9 May 2011.
12. Bohren, J.; Rusu, R.B.; Jones, E.G.; Marder-Eppstein, E.; Pantofaru, C.; Wise, M.; Mösenlechner, L.; Meeussen, W.; Holzer, S. Towards autonomous robotic butlers: Lessons learned with the PR2. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, 9–13 May 2011; pp. 5568–5575.
13. Fritz, C. Integrating Decision-Theoretic Planning and Programming for Robot Control in Highly Dynamic Domains. Master's Thesis, RWTH Aachen University, Knowledge-Based Systems Group, Aachen, Germany, 2003.
14. Schiffer, S.; Ferrein, A.; Lakemeyer, G. Reasoning with Qualitative Positional Information for Domestic Domains in the Situation Calculus. *J. Intell. Robot. Syst.* **2012**, *66*, 273–300. [CrossRef]

15. Hähnel, D.; Burgard, W.; Lakemeyer, G. GOLEX–Bridging the Gap between Logic (GOLOG) and a real robot. In *Advances in Artificial Intelligence*; Herzog, O., Günter, A., Eds.; Springer: Berlin/Heidelberg, Germany, 1998; Volume 1504, pp. 165–176.

16. Simmons, R.; Apfelbaum, D. A Task Description Language for Robot Control. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Victoria, BC, Canada, 13–17 October 1998; pp. 1931–1937.

17. Galindo, C.; Fernández-Madrigal, J.A.; González, J.; Saffiotti, A. Robot task planning using semantic maps. *Robot. Auton. Syst.* **2008**, *56*, 955–966. [CrossRef]

18. Lim, G.H.; Suh, I.H.; Suh, H. Ontology-based unified robot knowledge for service robots in indoor environments. *IEEE Trans. Syst. Man Cybern. Part A Syst. Hum.* **2011**, *41*, 492–509. [CrossRef]

19. Karg, M.; Kirsch, A. Acquisition and use of transferable, spatio-temporal plan representations for human-robot interaction. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura, Algarve, Portugal, 7–12 October 2012; pp. 5220–5226.

20. Brenner, M. Situation-Aware Interpretation, Planning and Execution of User Commands by Autonomous Robots. In Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Jeju Island, Korea, 26–29 August 2007; pp. 540–545.

21. Schiffer, S.; Ferrein, A.; Lakemeyer, G. Caesar: An intelligent domestic service robot. *Intell. Serv. Robot.* **2012**, *5*, 259–273. [CrossRef]

22. Beetz, M.; Mosenlechner, L.; Tenorth, M. CRAM—A Cognitive Robot Abstract Machine for everyday manipulation in human environments. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Taipei, Taiwan, 18–22 October 2010; pp. 1012–1017.

23. Lemaignan, S.; Warnier, M.; Sisbot, E.A.; Clodic, A.; Alami, R. Artificial cognition for social human-robot interaction: An implementation. *Artif. Intell.* **2017**, *247*, 45–69. [CrossRef]

24. Foka, A.; Trahanias, P. Real-time hierarchical POMDPs for autonomous robot navigation. *Robot. Auton. Syst.* **2007**, *55*, 561–571. [CrossRef]

25. Hsiao, K.; Kaelbling, L.P.; Lozano-Perez, T. Grasping POMDPs. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Roma, Italy, 10–14 April 2007; pp. 4685–4692.

26. Schmidt-Rohr, S.R.; Knoop, S.; Lösch, M. Bridging the Gap of Abstraction for Probabilistic Decision Making on a Multi-Modal Service Robot. *Robot. Sci. Syst. IV* **2008**, 105–110.

27. Schmidt-Rohr, S.R.; Dirschl, G.; Meissner, P.; Dillmann, R. A knowledge base for learning probabilistic decision making from human demonstrations by a multimodal service robot. In Proceedings of the International Conference on Advanced Robotics (ICAR), Tallinn, Estonia, 20–23 June 2011; pp. 235–240.

28. Williams, J.D.; Young, S. Partially Observable Markov Decision Processes for Spoken Dialog Systems. *Comput. Speech Lang.* **2007**, *21*, 393–422. [CrossRef]

29. Zhang, S.; Sridharan, M.; Sheng Bao, F. 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 (ICDL-EpiRob), San Diego, CA, USA, 7–9 November 2012.

30. Zhang, S.; Sridharan, M.; Wyatt, J.L. Mixed Logical Inference and Probabilistic Planning for Robots in Unreliable Worlds. *IEEE Trans. Robot.* **2015**, *31*, 699–713. [CrossRef]

31. Lahijanian, M.; Andersson, S.B.; Belta, C. Temporal Logic Motion Planning and Control with Probabilistic Satisfaction Guarantees. *IEEE Trans. Robot.* **2012**, *28*, 396–409. [CrossRef]

32. Agostini, A.; Torras, C.; Wörgötter, F. Efficient interactive decision-making framework for robotic applications. *Artif. Intell.* **2017**, *247*, 187–212. [CrossRef]

33. Ingrand, F.; Ghallab, M. Deliberation for autonomous robots: A survey. *Artif. Intell.* **2017**, *247*, 10–44. [CrossRef]

34. Ghallab, M.; Nau, D.; Traverso, P. The actor's view of automated planning and acting: A position paper. *Artif. Intell.* **2014**, *208*, 1–17. [CrossRef]

35. Tenorth, M.; Beetz, M. KnowRob: A knowledge processing infrastructure for cognition-enabled robots. *Int. J. Robot. Res.* **2013**, *32*, 566–590. [CrossRef]

36. Tenorth, M.; Beetz, M. Representations for robot knowledge in the KnowRob framework. *Artif. Intell.* **2017**, *247*, 151–169. [CrossRef]

37. OWL Web Ontology Language Overview. Available online: https://www.w3.org/TR/owl-features/ (accessed on 23 January 2021).

38. Pangercic, D.; Tenorth, M.; Jain, D.; Beetz, M. Combining perception and knowledge processing for everyday manipulation. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Taipei, Taiwan, 18–22 October 2010; pp. 1065–1071.

39. Becker, J.; Bersch, C.; Pangercic, D.; Pitzer, B.; Rühr, T.; Sankaran, B.; Sturm, J.; Stachniss, C.; Beetz, M.; Burgard, W. The pr2 workshop-mobile manipulation of kitchen containers. In Proceedings of the IROS Workshop on Results, Challenges and Lessons Learned in Advancing Robots with a Common Platform, San Francisco, CA, USA, September 2011; Volume 120.

40. Fan, Z.; Tosello, E.; Palmia, M.; Pagello, E. Applying Semantic Web Technologies to Multi-Robot Coordination. In Proceedings of the Workshop on New Research Frontiers for Intelligent Autonomous Systems (NRF-IAS), Venice, Italy, 18–19 July 2014.

41. Tenorth, M.; Kunze, L.; Jain, D.; Beetz, M. KNOWROB-MAP - knowledge-linked semantic object maps. In Proceedings of the IEEE-RAS International Conference on Humanoid Robots (Humanoids), Nashville, TN, USA, 6–8 December 2010; pp. 430–435.

42. Chen, X.; Ji, J.; Jiang, J.; Jin, G.; Wang, F.; Xie, J. Developing High-level Cognitive Functions for Service Robots. In Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Toronto, ON, Canada, 10–14 May 2010; Volume 1, pp. 989–996.

43. Opfer, S.; Jakob, S.; Geihs, K. Teaching Commonsense and Dynamic Knowledge to Service Robots. In Proceedings of the Social Robotics, Madrid, Spain, 26–29 November 2019; pp. 645–654.

44. Awaad, I.; Kraetzschmar, G.K.; Hertzberg, J. The Role of Functional Affordances in Socializing Robots. *Int. J. Soc. Robot.* **2015**, *7*, 421–438. [CrossRef]

45. Jakob, S.; Opfer, S.; Jahl, A.; Baraki, H.; Geihs, K. Handling Semantic Inconsistencies in Commonsense Knowledge for Autonomous Service Robots. In Proceedings of the IEEE International Conference on Semantic Computing (ICSC), San Diego, CA, USA, 3–5 February 2020; pp. 136–140.

46. Chen, X.; Ji, J.; Sui, Z.; Xie, J. Handling open knowledge for service robots. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Beijing, China, 3–9 August 2013; pp. 2459–2465.

47. Chen, K.; Lu, D.; Chen, Y.; Tang, K.; Wang, N.; Chen, X. The Intelligent Techniques in Robot KeJia—The Champion of RoboCup@Home 2014. In *RoboCup 2014: Robot World Cup XVIII*; Bianchi, R.A.C., Akin, H.L., Ramamoorthy, S., Sugiura, K., Eds.; Springer: Cham, Switzerland, 2015; pp. 130–141.

48. Chen, Y.; Wu, F.; Shuai, W.; Wang, N.; Chen, R.; Chen, X. KeJia Robot–An Attractive Shopping Mall Guider. In Proceedings of the International Conference on Social Robotics, Hefei, China, 23 July 2015; pp. 145–154.

49. Berlin, M.; Gray, J.; Thomaz, A.; Breazeal, C. Perspective Taking: An Organizing Principle for Learning in Human-Robot Interaction; In Proceedings of the International Conference on Artificial Intelligence (AAAI), Boston, MA, USA, 16–20 July 2006; Volume 2, pp. 1444–1450.

50. Pineda, L.A.; Rodriguez, A.; Fuentes, G.; Rascon, C.; Meza, I.V. Concept and Functional Structure of a Service Robot. *Int. J. Adv. Robot. Syst.* **2015**, *12*, 6. [CrossRef]

51. Pineda, L.A.; Meza, I.; Aviles, H.; Gershenson, C.; Rascon, C.; Alvarado, M.; Salinas, L. IOCA: Interaction-Oriented Cognitive Architecture. *Res. Comput. Sci.* **2011**, *54*, 273–284.

52. Laird, J.E.; Newell, A.; Rosenbloom, P.S. SOAR: An architecture for general intelligence. *Artif. Intell.* **1987**, *33*, 1–64.

53. Langley, P.; Laird, J.; Rogers, S. Cognitive Architectures: Research Issues and Challenges. *Cogn. Syst. Res.* **2009**, *10*, 141–160. [CrossRef]

54. Lieto, A.; Bhatt, M.; Oltramari, A.; Vernon, D. The Role of Cognitive Architectures in General Artificial Intelligence. *Cogn. Syst. Res.* **2018**, *48*, 1–3. [CrossRef]

55. Marr, D. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*; Henry Holt and Co., Inc.: New York, NY, USA, 1982.

56. Umbrico, A.; Cesta, A.; Cortellessa, G.; Orlandini, A. A Holistic Approach to Behavior Adaptation for Socially Assistive Robots. *Int. J. Soc. Robot.* **2020**, *12*, 617–637. [CrossRef]

57. Pineda, L.A.; Salinas, L.; Meza, I.; Rascon, C.; Fuentes, G. *SitLog*: A Programming Language for Service Robot Tasks. *Int. J. Adv. Robot. Syst.* **2013**, *10*, 358. [CrossRef]

58. Pineda, L.A.; Rodríguez, A.; Fuentes, G.; Rascón, C.; Meza, I.V. A light non-monotonic knowledge-base for service robots. *Intel Serv Robot.* **2017**, *10*, 159–171. [CrossRef]

59. Torres, I.; Hernández, N.; Rodríguez, A.; Fuentes, G.; Pineda, L.A. Reasoning with preferences in service robots. *J. Intell. Fuzzy Syst.* **2019**, *36*, 5105–5114. [CrossRef]

60. Reiter, R. A logic for default reasoning. *Artif. Intell.* **1980**, *13*, 81–132. [CrossRef]

61. Baader, F.; Calvanese, D.; Mcguinness, D.; Nardi, D.; Patel-Schneider, P. The Description Logic Handbook: Theory, Implementation, and Applications. In *The Description Logic Handbook*; Cambridge University Press: Cambridge, UK, 2003.

62. Brewka, G.; Eiter, T.; Truszczyński, M. Answer Set Programming at a Glance. *Commun. ACM* **2011**, *54*, 92–103. [CrossRef]

63. Py, F.; Rajan, K.; Mcgann, C. A systematic agent framework for situated autonomous systems. In Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Toronto, ON, Canada, 10–14 May 2010; Volume 2, pp. 583–590.

64. Cialdea Mayer, M.; Orlandini, A.; Umbrico, A. Planning and execution with flexible timelines: A formal account. *Acta Inform.* **2016**, *53*, 649–680. [CrossRef]

65. Fox, M.; Long, D. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.* **2003**, *20*, 61–124. [CrossRef]

66. Pineda, L.; Estrada, V.; Coria, S.; Allen, J. The obligations and common ground structure of practical dialogues. *Intel. Artif. Rev. Iberoam. Intel. Artif.* **2007**, *11*, 9–17. [CrossRef]

67. Rossi, S.; Ferland, F.; Tapus, A. User profiling and behavioral adaptation for HRI: A survey. *Pattern Recognit. Lett.* **2017**, *99*, 3–12. [CrossRef]

68. De Benedictis, R.; Umbrico, A.; Fracasso, F.; Cortellessa, G.; Orlandini, A.; Cesta, A. A Two-Layered Approach to Adaptive Dialogues for Robotic Assistance. In Proceedings of the IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), Naples, Italy, 31 August–4 September 2020; pp. 82–89.

69. Bruno, B.; Recchiuto, C.T.; Papadopoulos, I.; Saffiotti, A.; Koulouglioti, C.; Menicatti, R.; Mastrogiovanni, F.; Zaccaria, R.; Sgorbissa, A. Knowledge Representation for Culturally Competent Personal Robots: Requirements, Design Principles, Implementation, and Assessment. *Int. J. Soc. Robot.* **2019**, *11*, 515–538. [CrossRef]