KheDeLP: A Framework to Support Defeasible Logic Programming for the Khepera Robots

Edgardo Ferretti, Marcelo Errecalde, Alejandro García, and Guillermo Simari

***Abstract*— In this paper we present *KheDeLP*, a framework to support Defeasible Logic Programming (DeLP) for the *Khepera* robots. *KheDeLP* is a layered system where lower level layers allow interaction with simulated and real *Khepera* robots. Upper layers represent more abstract capabilities of the robots and provide a set of services which would facilitate our work in cognitive robotics. These layers hide low-level robot-computer communication and provides a high-order set of predicates to develop programs in a declarative manner. The most abstract layer in this framework provides support for knowledge representation and high-level reasoning. At this end, we use DeLP, a formalism which allows to deal with partial and potentially contradictory information. This formalism could be a valuable tool to face the coordination problems we are interesting in, where the dynamic features of the environment make this kind of information be the rule, not the exception.**

***Keywords*— Cognitive Robotics, DeLP, Khepera, Webots, Prolog.**

I. INTRODUCTION

Research in robotics in the last two decades was significantly influenced by the behavior-based approach to Artificial Intelligence (AI), which essentially postulates that in order to achieve good performance in a situated agent, like a robot, the agent's ability to react properly to the external environment should be the fundamental aspect to be considered. Nowadays, most of AI researchers recognize the importance of reactivity but also it is out of discussion, that this aspect alone is not enough to create successful situated agents capable of performing complex tasks. For instance, in cases like self-governing space-crafts and robots with complex social interaction capabilities, high-level representational thinking is required. An important aspect that usually arises in these areas is that it is very important not to make mistakes, as pointed out by Brian Williams[1]in [1]: "You have to think very carefully about all the things that could happen."

Ronald Arkin[2] also shares this vision and considers that intelligent robots should use a mix of behavioral-based intelligence and representational thought. Arkin defends this position stating that "There is strong neuro-physiological evidence of the co-existence of these systems in human-level intelligence, . . . , if we strive to reach artificial intelligence above that of bugs, it is important to incorporate both."

In this context, where the importance of higher level representation and reasoning in robotics is recognized, several groups have recently begun to work on what is called "cognitive robotics." Cognitive robotics intends to capture the application of logical formalisms and computational models of high-level cognitive functions, such as planning, to real-world and simulated robots.

This paper adheres the trends of cognitive robotics and therefore we propose the *KheDeLP* framework, that facilitates the experimentation with real and simulated *Khepera* robots, and provides a set of facilities that may contribute significantly in the development of the robots' deliberative component. This framework can play an important role in our research group which has as one of its main objectives, the design, implementation, and application of high-level multi-agent coordination models.

KheDeLP is a layered framework to support Defeasible Logic Programming (DeLP) [2] for the *Khepera* robots. The lower level layers allow the interaction with a group of *Khepera 2* mobile robots [3], with capabilities to pick and transport objects and perform different kinds of environment sensing. Moreover, before the direct experimentation with the robots we also perform robots simulations with *Webots* [4], a 3D realistic professional simulator. Furthermore, the use of this simulator allows us to model situations with more than three robots, the number of robots that we have at the laboratory.

On the other hand, upper layers represent more abstract capabilities of the robots and provide a set of services which can benefit profitably our future work in cognitive robotics.

Our aim is to develop deliberative agents to control the robots coordination, and many of the aspects related to the robots' behavior require an expressive representation language that easily reflect the decision processes made by the agents. In consequence, these layers hide low-level robot-computer communication and provides a high-order set of predicates to develop programs in a declarative manner.

One of the basic functionalities that *KheDeLP* provides is an interface in Prolog which represents all the sensorial and effectorial capabilities of the *Khepera 2* robots. Prolog is a programming language that has already been used to develop applications in the field of cognitive robotics [5], [6]. The layer on the top, named *cognitive layer* provides support for knowledge representation and high-level reasoning. Thus, the *KheDeLP* framework could be used in the direct experimentation of stand-alone applications as well as in the

E. Ferretti and M. Errecalde belong to the Computer Science Department, Universidad Nacional de San Luis, Ejército de los Andes 950, (D5700HHW) San Luis, Argentina. email:{ferretti, merreca}@unsl.edu.ar

A. García is a researcher of CONICET and with G. Simari they belong to the Department of Computer Science and Engineering, Universidad Nacional del Sur, Av. Alem 1253, (B8000CPB) Bahía Blanca, Argentina. email:{ajg, grs}@cs.uns.edu.ar

[1]Brian Williams, lead researchers at MIT's Artificial Intelligence Model-based Embedded and Robotic System Group.

[2]Ronald Arkin, Director of the Georgia Institute of Technology's Mobile Robot Laboratory.

development of the deliberative component of a cognitive robot architecture.

The paper is organized as follows: Section II presents an overview of the *Khepera 2* robots, the hardware platform we use at the laboratory. Section III describes the *KheDeLP* framework. Finally, section IV and V put forward the future work and conclusions.

II. KHEPERA 2 ROBOT OVERVIEW

The *Khepera 2* robot, is a miniature mobile robot that allows confrontation to the real world of algorithms developed in simulation for trajectory execution, obstacle avoidance, pre-processing of sensory information, hypothesis on behaviors processing, among others. Its small size (60 mm diameter, 30 mm height), light weight (approx. 70 grams), and compact shape are ideal for micro-world experimentation. The *Khepera 2* has eight infrared sensors to sense both ambient light levels and proximity to nearby objects. It also has two DC motors that are capable of independent variable speed motion, allowing the robot to move forward, backward, and complete a variety of turns at different speeds.

As can be observed in Figure 1, the *Khepera 2* has several extension modules that can be plugged into the top of the robot. These include an arm with a gripper, a linear vision system, and a matrix vision camera. The *Khepera 2* has an on-board Motorola 68331 (25MHz) processor, 512 KB RAM, 512 KB Flash memory programmable via serial port, and rechargeable NiMH batteries that allows it up to 60 minutes of autonomy. Thus, the *Khepera 2* has sufficient sensors and actuators to ensure that it can be programmed to complete a wide variety of tasks.

When connected to a host computer through the serial port, the `SemCor` control protocol is used to send control messages to the robot. As the robot may need to send an answer message to the host, ASCII messages are used to communicate between them. Each interaction consist of:

- A command, beginning with one or two ASCII capital letters and followed, if necessary, by numerical or literal parameters separated by a comma and terminated by a carriage return or a line feed, sent by the host computer to the *Khepera 2* robot.
- A response, beginning with the same one or two ASCII letters of the command but in lower case and followed, if necessary, by numerical or literal parameters separated by a comma and terminated by a carriage return and a line feed, sent by the *Khepera 2* to the host computer.

During the entire communication, the host computer acts as a master and the robot as a slave. All communications are initiated by the master.

Code can also be uploaded into the *Khepera's* memory for a stand-alone execution. Programs written in C language or in M68000 assembly language, can be compiled under many environments using a cross-compiler and uploaded in RAM or flashed in non-volatile memory. A complete API is available, either in C or assembly language, for programs to interface with the robot hardware.

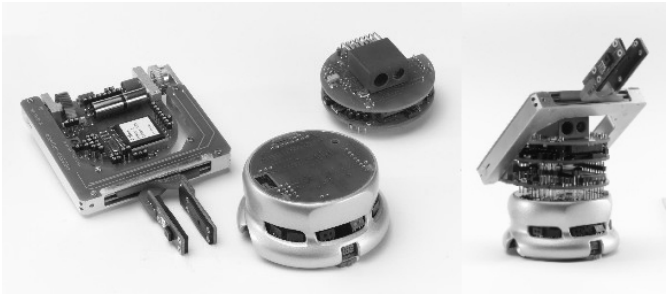

Fig. 1. *Khepera 2* robot and its accessories

III. THE KHEDELP FRAMEWORK

This framework is currently developed as a set of *Ciao* Prolog [7] modules running under the Linux operating system. To our view, *Ciao* is one of the most complete Prolog systems that allows the programmer to use sockets, multi-threads, Java and C embedded code in Prolog programs and vice versa, among others. In addition, it provides a fully integrated programming environment with the text editor *Emacs*, that allows the programmer to run, debug, compile, and syntax correction of Prolog programs.

In Figure 2 the four-layer *KheDeLP* framework is shown. The two lower layers handle the interactions with real and simulated robots. These layers have been designed to communicate with *Webots* in the same way they do with the real *Khepera 2* robots. On the contrary, the two layers on the top are dedicated to provide a set of high-level services which include: a) an interface in *Ciao* Prolog representing all the sensorial and effectorial capabilities of the *Khepera 2* robots (the *Sensorial / Effectorial layer*) and b) a support for knowledge representation and high-level reasoning (the *cognitive layer*.)

As can be observed in the figure, to manage the serial port communication with the robot this framework uses the *KRobot class* developed by Harlan *et al.* [8]. The *KRobot class* hides low-level robot-computer communication and allows developers to focus on robot-environment interaction.

As the Sensorial / Effectorial layer has been programmed in Prolog, it extends the functionalities provided by the *KRobot class* implementing a higher level interface which allows an easier interaction with the modules that represent the knowledge about the world, in a declarative manner.

Finally, the cognitive layer provides the high-level cognitive functions for the software that controls the robot. At this stage of the framework development, these functions are restricted to support knowledge representation in DeLP. This support will be an useful tool to deal with incomplete and contradictory information that is characteristic of this dynamic domain.

As can be noted in the figure, the cognitive and Sensorial / Effectorial layers can directly interact with stand-alone applications and a software component that we call "agent module."

With the phrase "stand-alone applications" we refer to those programs that allows the user to interact with the simulator or the real robot in an easy and direct way. This type of software can play an important role in the early stages of

experimentation where some particular aspects of the control of the robot need to be developed and tested.

The agent module will contain the specification of the behavior of the agent that controls the robot and in a future would be able to be considered as the most abstract layer of the framework. This layer is not implemented yet, but we can observe that the two upper layer of the actual framework will facilitate to a great extent this task, because they provide the necessary services to implement the traditional cycle "sense-deliberate-act" for controlling a cognitive robot.

Next, we describe in detail these four layers that compose our framework.

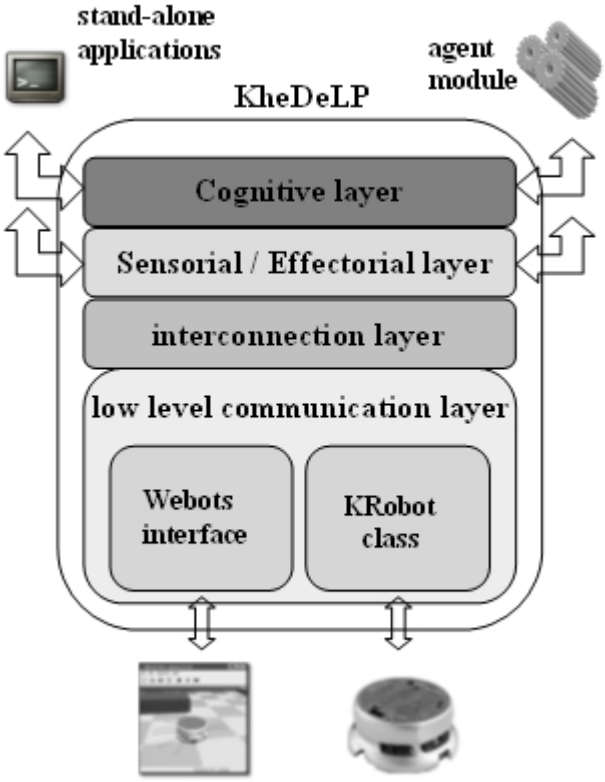

Fig. 2. The *KheDeLP* framework scheme

A. The low-level communication layer

This layer handles all the details related with serial communication among the robots and the high-level predicates of our framework. This layer is composed by two modules, one for the real robots and another for interfacing the simulator.

1) The KRobot class: The *KRobot class* is the base building block for the module that communicates with the robots. This C++ class maintains the information of the robot' state and provides a set of methods equivalent to the `SemCor` protocol commands. For example, the command to read from the proximity sensors situated around the robot is "`N`", where to this command the *Khepera 2* would respond with the following string, if it had hit an object by its front part: `n,0,259,1023,1023,278,0,0,0`. The response is returned as a C-style string and must be parsed to determine the values of each of the proximity sensors.

In contrast, if we want to read the proximity sensors of a *Khepera 2* robot associated with an object `r` of the type `KRobot`, we just have to invoke the method `r.readProxSensors();` and it saves these values in an internal structure of the object. Then, each of these sensor values can be accessed by the method `r.getProxSensor(i);` with $0 \leq i \leq 7$.

2) Webots interface: In *Webots*, the `DifferentialWheels` node defines any differentially wheeled robot. Thus, the *Khepera 2* robot is an instance of the `DifferentialWheels` node with its fields completed to match its shape and functionalities.

In this way, the module that handles the communication between the Prolog interpreter and the simulator translates the predicates available in the Sensorial / Effectorial layer, to their respective commands of the *Webots*' controllers API.

B. The interconnection layer

The development of this layer adheres the paradigm of a TCP/IP connection-oriented protocol, using Berkeley sockets.

In Section II was mentioned that during the entire communication, the host computer acts as a master and the robot as a slave, and that all the communications were initiated by the master. In consequence, the robots' control modules were programmed with the corresponding code of a server, while the predicates available in the Sensorial / Effectorial layer are seen as clients.

When the Sensorial / Effectorial layer's predicates should sent a command to a real or simulated robot, they launch a temporary client (programmed in C) that communicates to the server (the real or simulated robot) and waits for its answer. In Figure 3, this communication process is depicted.

As a final remark, one advantage of using TCP/IP sockets to develop this layer, is that it makes it possible to interact with a global camera (that covers the robots' world) and its video and command communication servers that process the images it obtains, and generate information packets that are made available to be used by the agents that control the robots. For instance, one alternative would be using the *Doraemon* video server [9], the one used in the E-League competition [10].

C. The Sensorial / Effectorial layer

This layer is composed by 40 predicates that represent the sensorial / effectorial capabilities of the *Khepera 2* robot and its extension turrets. As all the `SemCor` commands after being issued receive a result or a confirmation from the *Khepera 2* robots, all the predicates have a variable as parameter where this answer is returned.

The parameters denoted `OutB`, `OutInt` and `OutL` will return an appropriate boolean, integer or a list value. This returned value will contain useful information for the caller of the predicate. In the case of `OutB`, it will generally return 'true' when the execution of the required action is successful.

For example, in III-A.1 was mentioned that the `SemCor` command to read from the proximity sensors of the *Khepera 2* robot is "`N`", where to this command the robot responds the letter "`n`" followed by the eight values separated by commas

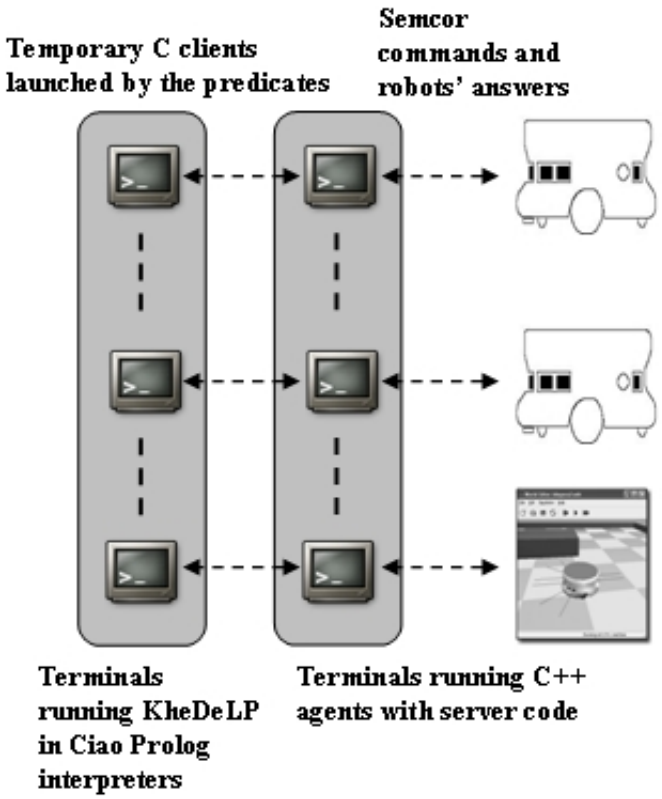

Fig. 3. Functioning of the interconnection layer

(*e.g.*, `n,0,259,1023,1023,278,0,0,0`.) Besides, it was noted that the method `r.readProxSensors();` of an object `r` of the type `KRobot`, saves these values in an internal structure of the object and later they can be accessed by the method `r.getProxSensor(i);` with $0 \leq i \leq 7$. In our case, the predicate `get_prox_sensors(OutL)` would return all the proximity sensors values in the list `OutL`.

Due to space constrains, in the Appendix a subset of all the predicates available in this layer are presented. Finally, it is important to remark that these predicates are used without distinction to communicate with a real or simulated robot.

D. Cognitive layer

This layer provides the high-level cognitive functions for the software that controls the robot, such as reasoning, knowledge representation, learning, planning, among others. At this stage of the framework development, these functions are restricted to support knowledge representation and reasoning in DeLP [2]. The overall idea is that the agent's knowledge is represented as a DeLP program[3] and this layer provides a predicate `answer(Q,Answer)` to query the DeLP interpreter. To this query, and depending on the agent's knowledge, the interpreter would respond YES, if `Q` is warranted; NO, if the complement of `Q` is warranted; UNDECIDED, if neither `Q` nor its complement is warranted; and UNKNOWN, if `Q` is not in the language of the program. In the future, the DeLP interpreter will constitute the base of the deliberative component of the agents to be developed.

Defeasible argumentation is a powerful formalism suitable for reasoning with potentially contradictory information in dynamic environments [11], [12]. To deal with contradictory and dynamic information, in DeLP, *arguments* for conflicting pieces of information are built and then compared in order to decide which one prevails. The argument that prevails provides a *warrant* for the information that it supports.

A brief explanation of how warrants are obtained using DeLP is included below (the interested reader is referred to [2] for a detailed explanation.) In DeLP, knowledge is represented using facts, strict rules or defeasible rules:

- *Facts* are ground literals representing atomic information or the negation of atomic information using the strong negation "$\sim$" (*e.g.*, $target(white)$).
- *Strict Rules* are denoted $L_0 \leftarrow L_1, \ldots, L_n$, where the *head* L_0 is a ground literal and the *body* $\{L_i\}_{i>0}$ is a set of ground literals (*e.g.*, $\sim target(black) \leftarrow target(white)$).
- *Defeasible Rules* are denoted $L_0 \prec\!\!- L_1, \ldots, L_n$, where the *head* L_0 is a ground literal and the *body* $\{L_i\}_{i>0}$ is a set of ground literals (*e.g.*, $\sim move\_forward \prec\!\!- obstacle(ahead)$).

Syntactically, the symbol "$\prec\!\!-$" is all that distinguishes a defeasible rule from a strict one. Pragmatically, a defeasible rule is used to represent defeasible knowledge, *i.e.*, tentative information that may be used if nothing could be posed against it. A defeasible rule "$Head \prec\!\!- Body$" is understood as expressing that "*reasons to believe in the antecedent* $Body$ *provide reasons to believe in the consequent* $Head$" [13].

A Defeasible Logic Program (*de.l.p.*) $\mathcal{P}$ is a set of facts, strict rules and defeasible rules. When required, $\mathcal{P}$ is denoted (Π, Δ) distinguishing the subset Π of facts and strict rules, and the subset Δ of defeasible rules. Observe that strict and defeasible rules are ground. However, following the usual convention [14], some examples will use "schematic rules" with variables. Given a "schematic rule" R, $Ground(R)$ stands for the set of all ground instances of R. Given a program $\mathcal{P}$ with schematic rules, we define: $Ground(\mathcal{P}) = \bigcup_{R \in \mathcal{P}} Ground(R)$. In order to distinguish variables, they are denoted with an initial uppercase letter.

Strong negation is allowed in the head of program rules, and hence may be used to represent contradictory knowledge. From a program (Π, Δ) contradictory literals could be derived, however, the set Π (which is used to represent non-defeasible information) must possess certain internal coherence. Therefore, Π has to be non-contradictory, *i.e.*, no pair of contradictory literals can be derived from Π. Given a literal L the complement with respect to strong negation will be denoted $\overline{L}$ (*i.e.*, $\overline{a}$=$\sim a$ and $\overline{\sim a}$=a.)

DeLP incorporates an argumentation formalism for the treatment of the contradictory knowledge that can be derived from (Π, Δ). This formalism allows the identification of the pieces of knowledge that are in contradiction. A dialectical process is used for deciding which information prevails. In particular, the argumentation-based definition of the inference relation

[3]The implementation (interpreter) of DeLP that satisfies the semantics described in [2] is currently accessible online at `http://lidia.cs.uns.edu.ar/DeLP`.

makes it possible to incorporate a treatment of preferences in an elegant way.

In DeLP a literal L is *warranted* from (Π, Δ) if a non-defeated argument $\mathcal{A}$ supporting L exists. To put it briefly, an *argument* for a literal L, denoted $\langle \mathcal{A}, L\rangle$, is a minimal set of defeasible rules $\mathcal{A}\subseteq\Delta$, such that $\mathcal{A} \cup \Pi$ is non-contradictory and there is a derivation for L from $\mathcal{A} \cup \Pi$. In order to establish if $\langle \mathcal{A}, L\rangle$ is a non-defeated argument, *argument rebuttals* or *counter-arguments* that could be *defeaters* for $\langle \mathcal{A}, L\rangle$ are considered, *i.e.,* counter-arguments that by some criterion are preferred to $\langle \mathcal{A}, L\rangle$. Since counter-arguments are arguments, defeaters for them may exist, and defeaters for these defeaters, and so on. Thus, a sequence of arguments called *argumentation line* is constructed, where each argument defeats its predecessor in the line (for a detailed explanation of this dialectical process see [2].) In DeLP, given a query Q there are four possible answers: YES, if Q is warranted; NO, if the complement of Q is warranted; UNDECIDED, if neither Q nor its complement is warranted; and UNKNOWN, if Q is not in the language of the program.

For example, consider an agent that represents some of its knowledge with the following *de.l.p.* program:

$$\left\{\begin{array}{l} target(white) \\ \sim target(black) \\ obstacle(X) \leftarrow \sim target(X) \\ move\_forward \prec target\_ahead \\ \sim move\_forward \prec target\_ahead, obst\_ahead \\ \sim move\_forward \prec target\_ahead, at\_target \\ target\_ahead \prec target(X), camera\_detects(X) \\ at\_target \prec target(X), prox\_sensor\_detects(X) \\ turn \prec obst\_ahead \\ turn \prec target(X), \sim camera\_detects(X) \\ obst\_ahead \prec obstacle(X), prox\_sensor\_detects(X) \end{array}\right\}$$

In our example, the target is a white object and everything that is not white (*e.g.,* black) it is considered as an obstacle. The agent has defeasible rules that encode its (defeasible) reasons for moving forward or for turning. Notice that the literal $camera\_detects$ represents an object that a linear vision camera detects, and $prox\_sensor\_detects$ represents an object that the proximity sensors detect. Both literals should be provided by a lower layer that interacts directly with the robots. For instance, consider a situation where both the camera and the proximity sensors detect a white object. In such a case, both $camera\_detects(white)$ and $prox\_sensor\_detects(white)$ succeed, and the following conflicting arguments can be obtained:[4]

$$\mathcal{A}_1 = \left\{\begin{array}{l} move\_forward \prec target\_ahead \\ target\_ahead \prec target(w), camera\_detects(w) \end{array}\right\}$$

$$\mathcal{A}_2 = \left\{\begin{array}{l} \sim move\_forward \prec target\_ahead, at\_target \\ target\_ahead \prec target(w), camera\_detects(w) \\ at\_target \prec target(w), prox\_sensor\_detects(w) \end{array}\right\}$$

Since $\mathcal{A}_2$ is a proper defeater for $\mathcal{A}_1$ (because it is *more informed* [2]) then there is a warrant for $\sim move\_forward$. In this situation, the answer for at_target is YES, for $\sim move\_forward$ is YES, and for $move\_forward$ is NO.

Since the robots are in a dynamic environment, these answers could differ if something changes. Consider now that the situation changes and the robot is not "at_target" (*e.g.,* the white object was moved by other robot) then only $camera\_detects(white)$ holds. In this new situation, the argument $\mathcal{A}_2$ can not be constructed and therefore, the answer for $move\_forward$ will be YES, providing reasons for moving forward.

Finally, consider a new situation in which the target is ahead but there is an obstacle to avoid, *i.e.*, $camera\_detects(white)$ and $prox\_sensor\_detects(black)$ holds. In this new situation, the argument $\mathcal{A}_1$ can be obtained but $\mathcal{A}_2$ can not. However, $\mathcal{A}_3$, a new proper defeater for $\mathcal{A}_1$, appears:

$$\mathcal{A}_3 = \left\{\begin{array}{l} \sim move\_forward \prec target\_ahead, obst\_ahead \\ obst\_ahead \prec obstacle(b), prox\_sensor\_detects(b) \end{array}\right\}$$

Hence, the answer for $\sim move\_forward$ is YES, and for $move\_forward$ is NO. Notice that in this last situation the following argument can be constructed:

$$\mathcal{A}_4 = \left\{\begin{array}{l} turn \prec obst\_ahead \\ obst\_ahead \prec obstacle(b), prox\_sensor\_detects(b) \end{array}\right\}$$

that provides a warrant for $turn$.

Next, a possible implementation for the predicates $camera\_detects(X)$ and $prox\_sensor\_detects(X)$, based on lower layers, follows:

camera_detects(white) :- k213_get_image(L),
search_sublist(whiteValue, L).
camera_detects(black) :- k213_get_image(L),
search_sublist(blackValue, L).
prox_sensors_detects(white) :- get_prox_sensors(L),
search_prox(whiteValue, L).
prox_sensors_detects(black) :- get_prox_sensors(L),
search_prox(blackValue, L).

In this example, the predicate *k213_get_image(L)* would return in *L* a list with 64 pixel values. Then, the predicate *search_sublist(whiteValue, L)* would search *white*, this involves finding a sublist of *constP* pixels with values higher than *whiteValue*. In the same way, looking for *black* would imply to find a sublist of *constP* pixels with values lower than *blackValue*.

If we now consider the predicate *prox_sensors_detects(X)*, in this case *get_prox_sensors(L)* would return in *L* a list with 8 values corresponding to the proximity sensors and *search_prox(whiteValue, L)* would succeed if any of the sensors' values is higher than *whiteValue*. On the contrary, *search_prox(blackValue, L)* would succeed if any of the sensors' values is lower than *blackValue*.

IV. FUTURE DEVELOPMENTS

At the moment, we have already programmed all the predicates of the Sensorial / Effectorial layer for simulated robots, while for the real *Khepera 2* robots, none of the predicates

[4]Due to space restrictions "white" will be abbreviated "w", and "black", "b".

related with the extension turrets have been programmed yet. To complete this task, we will have to add new classes to Harlan *et al.* C++ interface with one class per extension module, and methods for each `SemCor` command of the k213 linear vision extension turret, k6300 matrix vision extension turret, and the gripper-arm extension turret.

Even though this framework has not been tested under the Windows operating system, we think that its code should be easily ported because *Ciao* and *Webots* versions for Windows exist. However, minimal changes will be needed, for example, the serial port definition should be changed from `/dev/ttyS0` to `COM1`.

We plan to extend the low-level layer to allow the communication among the robots. This is a key feature to develop coordination models. However, as we are interested in developing coordination models where point-to-point and broadcast explicit communication exist, only this kind of facilities will be provided. In this way, this interface would also be useful to those researchers that have the *Khepera'* radio base module, because the robots could communicate among them in a wireless mode.

In the future, other capabilities like learning and high-level planning will also be incorporated in the cognitive layer.

V. Conclusions

In this paper we have presented *KheDeLP*, a flexible framework that helps researchers, teachers, and students in the development of applications for the *Khepera* robots that require high-level cognitive capabilities. The framework hides low-level robot-computer communication and provides a high-order set of predicates to help us to concentrate on the high-level problem specification. Besides, it has the advantage of communicating with *Webots* in the same way it does with the real *Khepera 2* robots.

KheDeLP is intended to be used in complex robotic scenarios which may involve changing goals, and partial and potentially contradictory information. At this end, *KheDeLP* provides support to use DeLP, a formalism amenable to deal with this kind of situations. In particular, we think that *KheDeLP* will play an important role in our research group which has as one of its main objectives, the design, implementation, and application of high-level multi-agent coordination models.

Acknowledgment

We thank the Universidad Nacional de San Luis and the Universidad Nacional del Sur for their unstinting support.

This work is partially supported by, Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) PIP 5050, and Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT) (PICT 2002, Nro.13096 and PICT 2002 Nro.12600).

Appendix
A subset of the predicates of the Sensorial / Effectorial layer

Here, some of the predicates belonging to the Sensorial / Effectorial layer with a brief explanation for each one, are presented:

A. Robot's predicates

- `move_forward(Sw, OutB)`: Makes the robot' motors to move forward indefinitely at speed `Sw` ($0 \leq$ `Sw` $\leq 1000\ mm/sec.$) In `OutB` returns an acknowledgment indicating the status of the operation required.
- `turn_right(Dg, OutB)`: Makes robot turn right `Dg` degrees passed as parameter. In `OutB` returns an acknowledgment indicating the status of the operation required.
- `get_prox_sensors(OutL)`: Returns all the proximity sensors' values in `OutL`.

B. K213 and K6300 vision extension turrets predicates

- `k213_get_image(OutL)`: Returns in `OutL` as a list, the 64 grey-level values corresponding to the pixels of the image.
- `k6300_get_line(Line, OutL)`: Returns in `OutL` as a list, the 160 decimal values of the row `Line` ($0 \leq$ `Line` ≤ 119.) An image must be acquired first.

References

[1] T. Simon, "Robots that reason, learn and crave improvement," http://www.exn.ca/ai/machine.asp, 2001, an interview to Brian Williams.
[2] A. J. García and G. R. Simari, "Defeasible logic programming: an argumentative approach," *Theory and Practice of Logic Programming*, vol. 4, no. 2, pp. 95–138, 2004.
[3] K-Team, "Khepera 2," http://www.k-team.com, a miniature mobile robot designed as a research and teaching tool.
[4] O. Michel, "Webots: Professional mobile robot simulation," *Journal of Advanced Robotics Systems*, vol. 1, no. 1, pp. 39–42, 2004. [Online]. Available: http://www.ars-journal.com/ars/SubscriberArea/Volume1/39-42.pdf
[5] A. J. García, G. I. Simari, and T. Delladio, "Designing an agent system for controlling a robotic soccer team," in *X Congreso Argentino de Ciencias de la Computación*, 2004.
[6] H. J. Levesque and M. Pagnucco, "Legolog: Inexpensive experiments in cognitive robotics," in *The Second International Cognitive Robotics Workshop*, Berlin, Germany, August 2000, pp. 104–109. [Online]. Available: citeseer.ist.psu.edu/levesque00legolog.html
[7] F. Bueno, D. Cabeza, M. Carro, M. Hermenegildo, P. López-García, and G. Puebla, "The ciao prolog system. reference manual," School of Computer Science, Technical University of Madrid (UPM), Tech. Rep. CLIP3/97.1, August 1997, available from http://www.clip.dia.fi.upm.es/.
[8] R. M. Harlan, D. B. Levine, and S. McClarigan, "The khepera robot and the krobot class: a platform for introducing robotics in the undergraduate curriculum," *SIGCSE Bulletin*, vol. 33, no. 1, pp. 105–109, 2001.
[9] B. Vosseteig, J. Baltes, and J. Anderson, "Robocup e-league video server," http://sourceforge.net/projects/robocup-video.
[10] "Oficial e-league webpage," http://agents.cs.columbia.edu/eleague/.
[11] C. I. Chesñevar, A. G. Maguitman, and R. P. Loui, "Logical Models of Argument," *ACM Computing Surveys*, vol. 32, no. 4, pp. 337–383, Dec. 2000.
[12] H. Prakken and G. Vreeswijk, "Logical systems for defeasible argumentation," in *Handbook of Philosophical Logic, 2nd ed.*, D.Gabbay, Ed. Kluwer Academic Pub., 2002.
[13] G. R. Simari and R. P. Loui, "A mathematical treatment of defeasible reasoning and its implementation," *Artificial Intelligence*, vol. 53, no. 2–3, pp. 125–157, 1992.
[14] V. Lifschitz, "Foundations of logic programming," in *Principles of Knowledge Representation*, G. Brewka, Ed. CSLI, 1996, pp. 69–127.