Percepts symbols or Action symbols? Generalizing how all modules interact within a software architecture for cognitive robotics

R. Marfil, L.J. Manso, J.P. Bandera, A. Romero-Garcés, A. Bandera, P. Bustos, L.V. Calderita, J.C. González, A. García-Olaya, R. Fuentetaja and F. Fernández

***Abstract*—Robots require a close coupling of perception and action. Cognitive robots go beyond this to require a further coupling with cognition. From the perspective of robotics, this coupling generally emphasizes a tightly integrated perceptuomotor system, which is then loosely connected to some limited form of cognitive system such as a planner. At the other end, from the perspective of automated planning, the emphasis is on a highly functional system that, taken to its extreme, calls perceptual and motor modules as independent functions. This paper proposes to join both perspectives through a unique representation where the responses of all modules on the software architecture (percepts or actions) are grounded using the same set of symbols. This allows to generalize the signal-to-symbol divide that separates classic perceptuomotor and automated planning systems, being the result a software architecture where all software modules interact using the same tokens.**

***Index Terms*—cognitive robotics, inner representations, symbol grounding**

I. Introduction

ABSTRACT reasoning about phenomena from the outer world is intimately tied with the existence of an internal representation of this external reality. From a robotic perspective, this implies the establishment and maintenance of a connection between what the robot reasons about and what it can sense [5]. The Physical Symbol Grounding is defined as the problem of how to ground symbol tokens to real world entities, i.e. to percepts that can have a high dimensionality and, unfortunately, that can vary under different conditions. Furthermore, the dynamic essence of the outer reality can impose that these percepts continuously change over time. Of course, this is a challenging problem, approached from very different points of view by many researchers in recent decades (e.g. see some examples on the brief survey by Coradeshi et al. [5]). Among these proposals, recent contributions [2] are pointing towards the use of a shared, unique internal representation. This representation is fed with the symbol tokens generated by all the software components in charge of solving the grounding problem.

R. Marfil, J.P. Bandera, A. Romero-Garcés and A. Bandera are with University of Málaga. E-mail: {rebeca, jpbandera, argarces, ajbandera}@uma.es

L.J. Manso, P. Bustos, and L.V. Calderita are with University of Extremadura. E-mail: {lmanso, pbustos, lvcalderita}@unex.es

J.C. González, A. García-Olaya, R. Fuentetaja and F. Fernández are with University Carlos III of Madrid. E-mail: {josgonza, agolaya, rfuentet, ffernand}@inf.uc3m.es

Figure 1(left) shows a schematic view of RoboCog [4], [3], [11], a software architecture where this premise of maintaining a shared representation is hold. The figure depicts how two different agents interact through this representation for unfolding a 'go to person' behavior provided by the deliberative agent (in this specific case, the PELEA module [1]). Domain-dependent agents, in charge of performing the necessary physical and perceptual actions, use this plan and the shared world model to perform their activities. In this case, the Person agent detects the pose of the person and provides these data to the representation, and the Navigation agent takes these data and moves the robot. The shared representation is cooperatively built and kept updated by all the modules.

The use of the state of the world as the best mechanism to communicate software components was pointed out by Flynn and Brooks [6], as a way for reducing the large and close dependence of the components within the subsumption architecture. Paradoxically, this was considered more a problem than an advantage by Hartley [8], as *similar states of the world could mean different things depending on the context*. Thus, *this would result in a behavior being activated when another behavior accidentally allowed the world to satisfy its preconditions*. Substituting the real world by this inner representation, the problem can be minimized as symbolic information can disambiguate these situations. In fact, in Figure 1(left) the task to accomplish by all agents is clear as it is commanded by the deliberative agent.

As Fig. 1(left) depicts, the execution of the high-level action emanated from PELEA is controlled by the Executive module, a component that also provides the interface to the representation. The Executive partially assumes the functionality of the Sequencer module of classical three-layer architectures [7]. It interfaces PELEA, from which it receives the plan to execute and to which it reports changes on the representation, through asynchronous events. The Executive module publishes the representation to all agents (blank arrows in Figure 1(left)) and it is also the only module in charge of checking if the changes on the representation, coming from the agents, are valid or not. More details can be read in [10], but here this property is reflected by connecting the Executive core with a Grammar data set. Significantly, this scheme implies that the agents will execute the required subtask as they receive a direct order from the Executive. Hence, the internalized state of the world does not guide its behavior and it will be only used, as described before for the simple 'go to

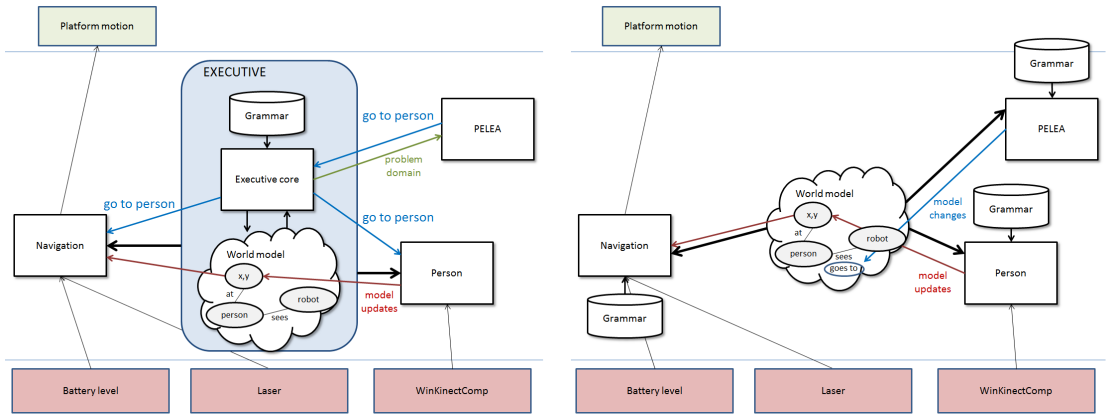

Fig. 1. (left) A brief scheme of the RoboCog architecture, showing its three main components: PELEA (a high level module for planning, monitoring, and learning), an Executive in charge of redirecting the plans from the planning module to the corresponding low-level modules and managing the representation of the world, and a set of domain-dependent agents (in this case represented by Navigation and Person). Modules with red background (Battery level, Laser...) provide inputs to the agents, meanwhile those with green background (Platform motion) receive the results from the agents. Both sets constitute the Hardware Abstraction Layer (HAL) of the system; and (right) the scheme of the new proposal for dealing with this same task. There is not an Executive module and PELEA needs to change the representation to achieve the correct response from the Person and Navigation agents.

person' example, to intercommunicate the agents. This paper proposes to change this scheme by removing the Executive module and forcing the agents to encode the action using the same set of symbols. Figure 1(right) schematizes how PELEA should ask the agents to perform the 'go to person' behavior. Briefly, as it occurs with perceptions, actions will also be thought of as changes to the world. It is in this new proposal where the shared representation truly becomes the core of the architecture, storing all data that is required for the agents to perform their activities. This simplifies the architecture as a whole, as no further modules are required to understand the state of the robot and its context. Furthermore, fewer connections eases intercommunication and generalization. As a major disadvantage, this scheme implies that the modules use a more complex logic to infer their activities from the state, without specific action commands. However, they are also more easily modified, added or removed without affecting the rest of the architecture. This approach also eases the deployment of different behavioral schemes such as stigmergic collaboration or competitive approaches (e.g. using more than one planner).

The rest of the paper is organized as follows: Section II briefly presents the unified framework for representing geometric and symbolic information. The proposal has been currently endowing within CLARC[1], a robot in charge of performing different tests to geriatric patients. Sections III and IV show the unfolding of the proposal for performing the Barthel test and the preliminary results from this work. An open discussion about the pros or cons of this new scheme is sketched at Section V.

[1]http://echord.eu/essential_grid/clark/

II. The Deep State Representation

The Deep State Representation (DSR) is a multi-labeled directed graph which holds symbolic and geometric information within the same structure. Symbolic tokens are stated as logic attributes related by predicates that, within the graph, are stored in nodes and edges respectively. Geometric information is stored as predefined object types linked by 4×4 homogeneous matrices. Again, they are respectively stored as nodes and edges of the graph. Figure 2 shows one simple example. The **person** and **robot** nodes are geometrical entities, both linked to the **room** (a specific anchor providing the origin of coordinates) by a rigid transformation. But, at the same time that we can compute the geometrical relationship between both nodes ($RT^{-1} \times RT'$), the **person** can be located (is_with) close to the **robot**. Furthermore, an agent can annotate that currently the **robot** is_not **speaking**.

A. Data structure

As a hybrid representation that stores information at both geometric and symbolic levels, the nodes of the DSR store concepts that can be symbolic, geometric or a mix of them. Metric concepts describe numeric quantities of objects in the world that can be structures like a three-dimensional mesh, scalars like the mass of a link, or lists like revision dates. Edges represent relationships among symbols. Two symbols may have several kinds of relationships but only one of them can be geometric. The geometric relationship is expressed with a fixed label called RT. This label stores the transformation matrix (expressed as a Rotation-Translation) between them.

Then, the DSR can be described as the union of two *quivers*: the one associated to the symbolic part of the representation, $\Gamma_s = (V, E_s, s_s, r_s)$, and the one related to the geometric part, $\Gamma_g = (V_g, E_g, s_g, r_g)$. A quiver is a quadruple, consisting of a set V of nodes, a set E of edges, and two maps $s, r : E \to V$.

These maps associate with each edge $e \in E$ its starting node $\mathbf{u} = s(e)$ and ending node $\mathbf{v} = r(e)$. Sometimes we denote by $e = \mathbf{uv} : \mathbf{u} \rightarrow \mathbf{v}$ an edge with $\mathbf{u} = s(e)$ and $\mathbf{v} = r(e)$. Within the DSR, both quivers will be finite, as both sets of nodes and edges are finite sets. A *path* of length m is a finite sequence $\{e_1, ...e_m\}$ of edges such that $r(e_k) = s(e_{k+1})$ for $k = 1...m-1$. A path of length $m \geq 1$ is called a *cycle* if $s(e_1)$ and $r(e_m)$ coincide.

According to its nature, the properties of the symbolic quiver Γ_s are:

1) The set of symbolic nodes V contains the geometric set V_g (i.e. $V_g \in V$)
2) Within Γ_s there are no cycles of length one. That is, there are no *loops*
3) Given a symbolic edge $e = \mathbf{uv} \in E_s$, we cannot infer the inverse $e^{-1} = \mathbf{vu}$
4) The symbolic edge $e = \mathbf{uv}$ can store multiple values

On the other hand, according to its geometric nature and the properties of the transformation matrix RT, the characteristics of the geometric quiver Γ_g are:

1) Within Γ_g there are no cycles (acyclic quiver)
2) For each pair of geometric nodes $\mathbf{u}$ and $\mathbf{v}$, the geometric edge $e = \mathbf{uv} \in E_g$ is unique
3) Any two nodes $\mathbf{u}, \mathbf{v} \in V_g$ can be connected by a unique simple path
4) For each geometric edge $e = \mathbf{uv} = RT$, we can define the inverse of e as $e^{-1} = \mathbf{vu} = RT^{-1}$

Thus, the quiver Γ_g defines a directed rooted tree or rooted tree quiver [9]. The *kinematic chain* $C(\mathbf{u}, \mathbf{v})$ is defined as the path between the nodes $\mathbf{u}$ and $\mathbf{v}$. The equivalent transformation RT of $C(\mathbf{u}, \mathbf{v})$ can be computed by multiplying all RT transformations associated to the edges on the paths from nodes $\mathbf{u}$ and $\mathbf{v}$ to their closest common ancestor $\mathbf{w}$. Note that the values from $\mathbf{u}$ to the common ancestor $\mathbf{w}$ will be obtained multiplying the inverse transformations. One example of computing a kinematic chain is shown in Figure 2.

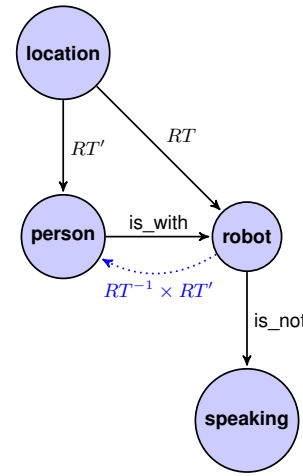

Fig. 2. Unified representation as a multi-labeled directed graph. Edges labeled as is_with and is_not denote logic predicates between nodes and they belong to Γ_s. Edges starting at **room** and ending at **person** and **robot** are geometric and they encodes a rigid transformation (RT' and RT respectively) between them. Geometric transformations can be chained or inverted to compute changes in coordinate systems.

Input: *action* from the Executive core
while *(1)* **do**
  subscribe to DSR updates;
  process { *action* };
  if *DSR_changes* **then**
    publish new DSR;
  end
end

Algorithm 1: Procedure of an agent within RoboCog

B. Internalizing the outer world within the DSR

The complexity of the domain-dependent modules typically implies that they will be internally organized as networks of software components (*compoNets*). Within each compoNet, the connection with the DSR is achieved through a specific component, the so-called *agent*. These agents are present in the two schemes drawn at Figure 1, but its degree of complexity has dramatically changed when we move from one scheme to the other. Within RoboCog (Figure 1(left)), the internal execution of an agent can be summarized by the Algorithm 1. With the removing of the Executive, the agents need to search for those changes on the DSR that launch the specific problem-solving skills of the compoNets they represent (e.g. detect the pose of a person). The new agents should then modify its internal data flow, as it is briefly outlined at Algorithm 2.

The **search_for_changes** skill depends on each agent and the behaviors that the compoNet can solve. Within the algorithm, it is stated that this function returns the *action* to perform. This is the most significant difference between Algorithms 1 and 2: in the first case the action is imposed by an external module, but in the second one, the action is determined by the agent. As we will briefly discuss at Section V this opens new ways for dealing with the top-down and bottom-up mechanisms for determining what the next action to perform will be or for implementing reflexive behaviors. The whole execution of the compoNet is conditioned by its inherent Grammar, i.e. the database storing triplets with the states of the DSR after, during and before the compoNet executes a specific action. Figure 3 shows one example, stored at the Grammar of the Speech agent (see Section III). Figure 3(left) shows the state of the DSR before PELEA states that the robot should say the sentence **yyy**. When PELEA changes the DSR (changing the attribute **xxx** to **yyy**, between **test** and **test_part**), and the agent Speech receives the new state, Speech uploads the DSR to inform all agents that the **robot** is **speaking**. When the sentence ends, Speech changes the DSR to **robot** finish **speaking**. Contrary to the way we work within RoboCog, where the Grammars were only defined by a initial state of the DSR (before an agent executes the action) and an ending state (after an agent executes the action), the agents must now to inform that they *are executing* the action. This constitutes the current way to avoid that other agent on the architecture

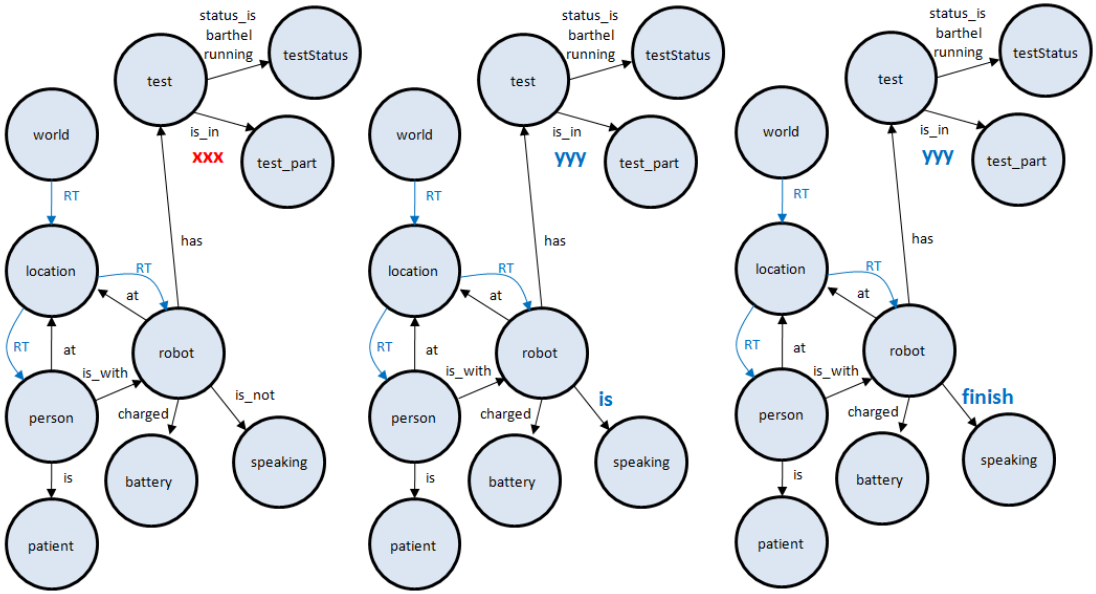

Fig. 3. (Left) The state of the DSR before PELEA states that the robot should say a sentence **yyy**; (center) PELEA changes the text to speech (from **xxx** to **yyy**) and then, when the Speech agent reads the DSR, the robot starts to speech (**robot** is **speaking**); and (right) the sentence has been said and the Speech agent informs to the rest of the agents through the DSR (**robot** finish **speaking**).

while *(1)* **do**
  subscribe to DSR updates;
  search_for_changes { output: *action* };
  process { *action* };
  if *DSR_changes* **then**
    update DSR;
  end
end

Algorithm 2: Procedure of an agent within the new proposal

modifies that part of the DSR meanwhile an action is being executed.

III. Building a whole architecture around the DSR

A. Our use case: the Barthel test

CLARC is waiting on Room 1 for its first patient. When Dr. Cesar presses the Start button on his mobile phone, CLARC wakes up and looks for Carlos, his patient, who should be sitting in front of it. When CLARC sees him, he greets him and presents itself as the responsible for conducting a small test, which will help the doctor to know how he is. It also briefly describes him what the test will be: basically a collection of questions that must be answered by selecting one of the 3-4 options described. And then the test starts. Sometimes, CLARC hear words that it does not understand. Sometimes, it just hears nothing. However, these situations are expected... and planned!. It is patient and can repeat the phrase several times, also suggest to leave it and go to the next question, and also always offer the option to answer using the touch screen on its chest. After 10-15 minutes, the test ends. It is time to say goodbye to Carlos and to send an internal message to Dr. Cesar indicating that the result of the test is stored on the CGAmed server for being validated.

This brief summary describes how the CLARC robot should address the Barthel Test. For performing the required actions, it is endowed with a software architecture that is showed at Figure 4. Out of the figure is the CGAmed server, where the application that Dr. Cesar used for launching the test is set. Once the Deliberative module (PELEA) receives this command, it wakes up the activity of the system, translates the high level action into a set of low level commands, and introduces the first of these commands within the DSR as a structural change on the DSR. Each of these changes provokes the response of one or several agents, which will try to modify the DSR towards a new state. Certain commands are single questions, that the agent of PELEA can ask examining the DSR. For instance, the command SearchPatient is answered as Yes, if the robot is seeing the patient seated in front of it, or as No, otherwise.

B. Overview of the architecture

Figure 4 shows an overview of the whole architecture in charge of performing the Barthel test within the CLARK

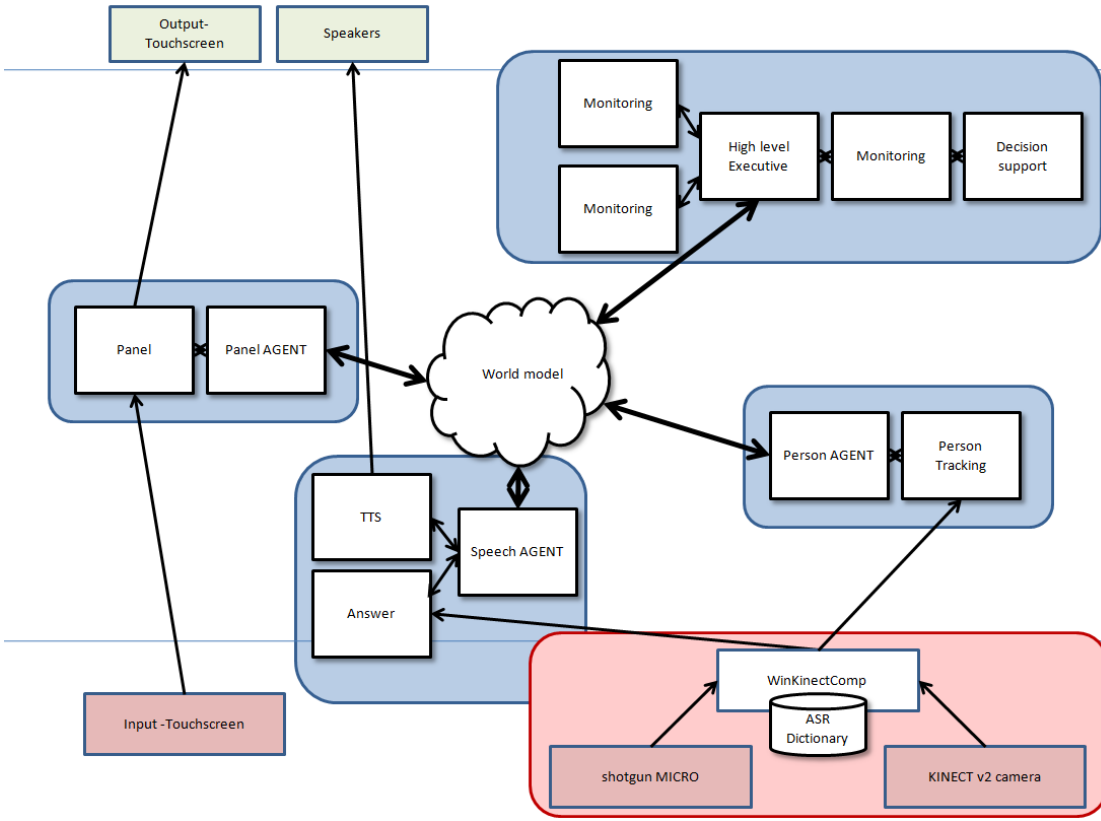

Fig. 4. Overview of the architecture within the CLARC robot. There are currently four compoNets: PELEA, Person, Panel and Speech. The Grammars that drive the behavior of these agents is encoded within the agents.

project. Surrounding the World model provided by the DSR there are four agents: PELEA, Speech, Panel and Person. The first one is in charge of providing the deliberative skills to the architecture, but it interacts with the rest of agents using the same procedure (i.e. changing the DSR). The Speech agent is the responsible of understanding the answers of the patient or guardian and of translating the text into speech, generating the voice of CLARC. This agent manages a specific grammar for dealing with the answers. The Panel agent manages the touch screen, which provides a non-verbal channel for interacting with the patient that complements the verbal one. Finally, the Person agent is the responsible of detecting and tracking the face of the interviewed person. It should be noted that, for this stage of the project, it is not needed that the robot moves. A fifth module is the WinKinectComp. It runs on a second PC and is in charge of capturing the preprocessed data provided by a rgbd Kinect sensor (i.e. joints of the person) and a shotgun microphone (automatic speech recognition). This data is provided to the Person and Speech compoNets.

PELEA is the most complex agent within the architecture. Within this project, it includes the following components

- The *High Level Executive* (HLE) module manages the whole compoNet. It receives the global goals and invokes the Monitoring module to get a plan achieving them. Then it takes the first action of the plan and invokes the HighToLow module to decompose it into low-level actions. These actions are then inserted into the DSR as changes on the model. The HLE looks at the changes in the DSR and, after a conversion to high level knowledge performed by LowToHigh, sends them to Monitoring that checks whether the plan is executing conveniently.
- The *Monitoring* module is in charge of maintaining a high level model of the environment and of invoking the Decision Support module when any deviation in the execution of the plan arises. It detects for example that the user has not answered a question or is not facing the robot and tries to find alternate plans to solve the problem found.
- The *Decision Support* module creates a plan starting from the current state, the goals to be achieved, the possible states of the world and the description of the changes the actions produce in the world state. To create the plan it

invokes an automated planner that returns the sequence of actions achieving the goals.
- The *HighToLow* module converts the high level actions of the plan created by the Decision support module into low level actions that can be included into the Inner Model.
- The *LowToHigh* module converts the information contained in the Inner Model, which represents knowledge in the form of binary predicates into n-ary predicates that the Monitoring module uses to reason about the correctness of the execution of the plan.

C. Encoding the grammar rules within the agents

With the removal of the central Executive module, we can consider that its role is now encoded within the agents on the architecture (see Figure 4). Thus, as Figure 1(right) shows, each compoNet has now its own Grammar. This Grammar is local to the compoNet and is currently encoded within the code of the agent. In an architecture that is mainly driven by how the internalized world changes, the coherence on the encoding of each change and its global synchronization are basic aspects. Although we have briefly described how the world is internalized at the DSR at Section II-B, we will provide here a more detailed description of how a Grammar is encoded within an agent.

Algorithm 3 illustrates how the specific grammar of the Speech compoNet is encoded in the agent. The Speech compoNet is in charge of translating the chosen sentence from text to speech and of receiving the responses from the patient (via the Automatic Speech Recognition (ASR) set on the WinKinectComp). There are three main modules within the Algorithm 3. The finishSpeaking is launched by the TTS module to the Speech agent to inform this that a sentence has been said. In the DSR, this implies that the **robot** is **speaking** must change to **robot** finish **speaking**. On the other hand, the setAnswer is launched by the Answer module to the agent to inform that a response has been captured. The DSR is changed from **person** waiting **answer** to **person** got **answer**. It also provides the specific $answer$. Thus, these functions encode the final state of the two rules driven the responsibilities of the Speech compoNet (e.g. Figure 3(right) shows the result of launching finishSpeaking). Within the main loop of the agent (compute), it is encoded the searching for changes aforementioned at algorithm 2. In this case, we document the two situations that launch the waiting of a new response from the patient (waitingAnswer) and the saying of a new sentence (startSpeaking). In the first case, the change on the DSR is done without evaluating any additional constraint. On the second case, we will evaluate if the $label$, i.e. the sentence to say, has been changed (see Figure 3). The procedures to address (i.e. the **process** { $action$ } of algorithm 2) are also launched (**canAnswer**() and **setText**($label$), respectively). But, as described at Section II-B, and just before launch one of these procedures, the agent notifies to the rest of agents that the process is under execution (publishing the new DSR model).

```
finishSpeaking
if getEdge(robot,speaking) == is then
    removeEdge(robot,speaking, is);
    addEdge(robot,speaking, finish);
    publishModification();
end
setAnswer
if getEdge(person,answer) == waiting then
    removeEdge(person,answer, waiting);
    addEdge(person,answer, got [answer]);
    publishModification();
end
compute
if worldChanged then
    waitingAnswer
    if getEdge(person,answer) == can then
        removeEdge(person,answer, can);
        addEdge(person,answer, waiting);
        canAnswer();
        model_modified = true;
    end
    startSpeaking
    if getEdge(test,test_part) == is_in then
        {q, label} = getEdgeAttribute(test,test_part);
        if label != label_back then
            removeEdge(robot,speaking, is_not);
            addEdge(robot,speaking, is);
            setText(label);
            model_modified = true;
        end
    end
    changeConfig
    if ... then
        ...
    end
    if model_modified then
        publishModification();
    end
end
```

Algorithm 3: Example of the Grammar encoded within the Speech agent

IV. Experimental results

The new scheme has been applied for implementing the Barthel test within the aforementioned CLARK project. The Barthel test consists of ten items that measure a person's daily functioning; specifically the activities of daily living and mobility: Bladder and bowel function, transfers and mobility, grooming and dressing, bathing and toilet use, and feeding and stairs use. Being actions and perceptions internalized within the DSR, the monitoring of the evolution of this state representation allows to track the whole execution of the use case. Fortunately, we have at our disposal in RoboComp the graphical tools for addressing this monitoring (see Figure 5).

The Barthel test needs that the robot can speech and show on a touchscreen specific sentences, for asking or helping. The robot must also hear or capture from the touchscreen the answers from the patient or relative. Finally, it is needed to track the presence of the person. The use case includes a first introduction, where the test is explained to the user, and then the robot asks the user for ten items. Each item implies that

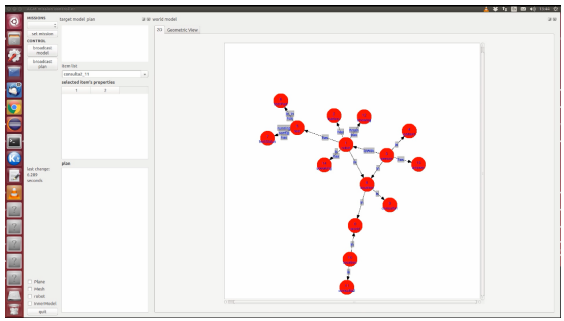

Fig. 5. Monitoring the evolution of the DSR.

the user chooses the option that best fits its state from a list. The presentation of each item follows the same scheme:

1) The robot introduces the item through voice and a message on the touchscreen
2) The robot describes each possible option (voice and message)
3) The robot asks the user to give an answer (voice or tactile on the touchscreen)
4) If after an specific time there is no response
   a) The robot describes each possible option via voice
   b) The robot shows all options on the touchscreen

If after this second try the patient does not answer, the robot will go to the next item on the test. Two non-answered items will provoke that the robot asks the clinician to attend. Figure 6 shows the evolution of the DSR during the Barthel test. It shows how the state evolves when the person is lost and then is detected again. It can be noted that the DSR practically provides a semantically annotated view of the scene. We have not measured response times but currently all the test can be run online and without remarkable latencies. The people from the Hospital Universitario Virgen del Rocío has checked the test and it is now ready for be evaluated by real users.

V. Open Discussion

It is really hard to draw a Conclusion Section as this is a preliminary work. In any case, we are currently able to run a complete Barthel test, following the correct course of actions but also acting against exogenous events such as the suddenly absence of the interviewed person (as Figure 6 shows). This allows us to open this Section and sketch what the main pros or cons of this proposal could be.

Pros. On one hand, we have been able to encode using the same scheme and collection of tokens the commands originated from a deliberative module and the perceptions, more or less elaborated by the domain-dependent modules, but always coming from the sensors. It is interesting to note that this provides a natural mechanism for merging top-down and bottom-up procedures for determining the course of action. In this proposal, the domain-dependent modules can respond in the same way to both kinds of processes. For instance, when the robot looses the person, the Speech component can immediately stop talking and PELEA can change the global course of action. The local behavior of the Speech module does not need to wait for receiving a specific command from PELEA. Thus, these modules can learn how to react to specific stimulus.

Cons. It is clear that the complexity of those software components in charge of monitoring the evolution of the DSR (our agents) is now very more complex. Although the architecture could provide the impression of being modular, this is not a reality in its current version: the definition of the agents demands a high degree of coupling among all researchers on charge of the implementation of each one of them. The reason of this dependence can be on the need of working with the same collection of symbols/tokens, which is not currently defined in advance. Thus, each change requires that all agents know how it will be 'written' on the DSR. Agents will also manage with care how they advise to the rest of agents that they are working over a specific part of the DSR. The current messages on the DSR (such as the **robot** is **speaking**) must be correctly interpreted by other agents that could be interested on using the speakers. We need again a very close interaction among the whole team of programmers.

Obviously, future work will focus on dealing with the main problems detected after these trials. We need to develop a mechanism that allows the agents to access to specific parts of the DSR (e.g. the person) and that sets and manages priority levels for using the resources. The aim should be to mimic the main guidelines of active vision perception, avoiding that other agents, with lower priority levels, can change this part of the DSR meanwhile an action is being executed. It is also mandatory to change the current way of encoding the agents, providing a mechanism that can allow a easier encoding. Finally, the collection of symbols must be *generalized*, as a way for achieving the desired modularity of the compoNets. The current encoding, which allows to *read* the state of the outer world by a simple observation of the DSR, can be a good option for achieving this generalization. However, we must also be open to the possibility that this encoding can ease our monitoring of the evolution of the DSR, but could not be the best option for achieving autonomous learning of the task-dependent modules. Other encodings (e.g. low-level features or outcomes generated/employed by these modules) could be employed at the graph items that are closer to the sensors. This will increase the data volume represented on the DSR but also open the architecture for techniques such as deep learning, which could be applied to generate the current symbols encoded using natural language. We are currently working on other tests, such as the Minimental or the GetUp & Go ones. This last one requires a high coupling of the DSR with the geometrical reality of the motion of the patient's joints during a short sequence of movements. It will be then required that the annotating of low-level features on the DSR is also increased.

Acknowledgments

This paper has been partially supported by the Spanish Ministerio de Economía y Competitividad TIN2015-65686-C5 and FEDER funds and by the FP7 EU project ECHORD++ grant 601116 (CLARK project).

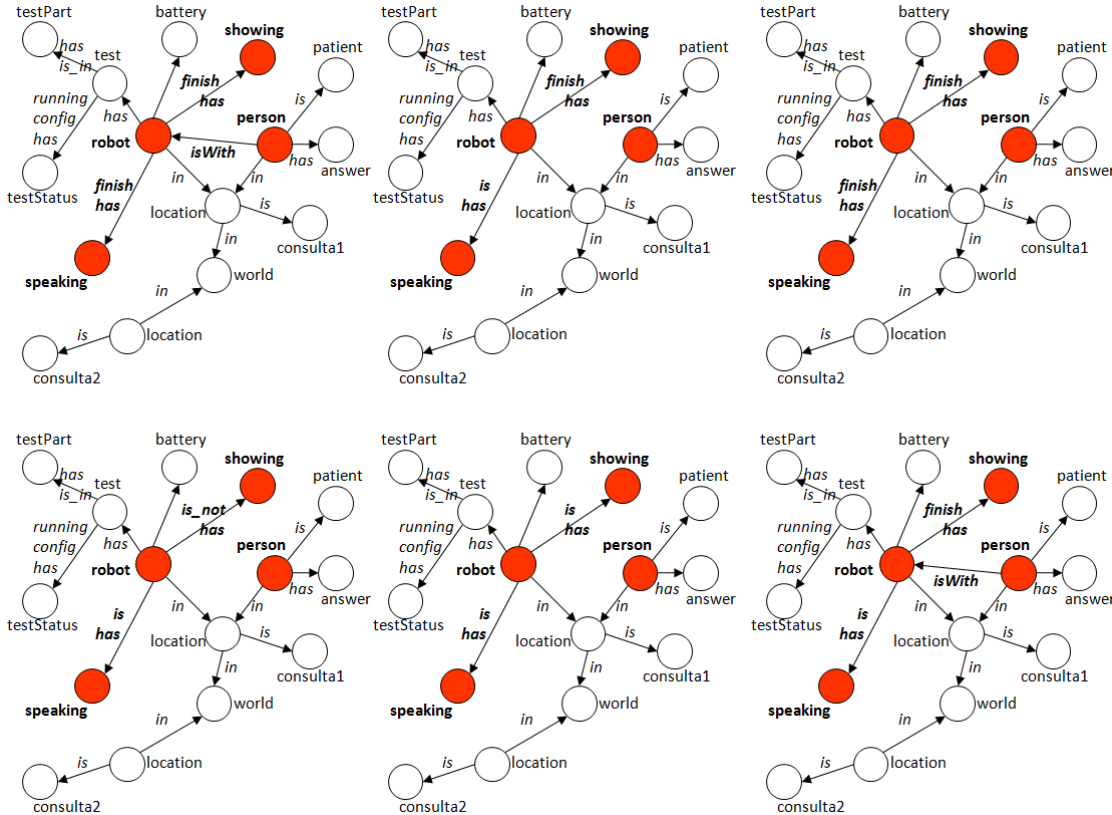

Fig. 6. Evolution of the DSR: During the execution of the Barthel test, the robot loses the person for several frames and then asks the patient to sit again in front of it. More related nodes and edges are colored and in bold, respectively. The figure shows a sequence of views of the DSR: (top-left) before losing the person, the robot has just finished on asking a question and showing a message on the touchscreen; (top-center) the **robot** is **speaking** a new sentence but then it loses the person. The situation is reported to PELEA; (top-right) but the Speech agent immediately stops speaking; (down-left) the Speech and Panel returns the DSR to its original state (**robot** is_not **speaking** and **robot** is_not **showing**) and PELEA changes the world proposing to speech a specific sentence ('Please, sit down in front of me'). This sentence is encoded as a label on the is_in attribute between **test** and **testPart**; (down-center) the Panel also shows this sentence on the touchscreen; and (down-right) the **person** isWith **robot** again and PELEA determines that the test can continue

REFERENCES

[1] V. Alcázar, C. Guzmán, D. Prior, D. Borrajo, L. Castillo and E. Onaindia, Pelea: Planning, learning and execution architecture, *PlanSIG10*, 17-24, 2010

[2] S. Blumenthal, H. Bruyninckx, W. Nowak and E. Prassler, A scene graph based shared 3d world model for robotic applications. *IEEE International Conference on Robotics and Automation*, 453-460, 2013

[3] L.V. Calderita, L.J. Manso, P. Bustos. C. Suárez-Mejías, F. Fernández, A. Bandera, THERAPIST: Towards an Autonomous Socially Interactive Robot for Motor and Neurorehabilitation Therapies for Children, *JMIR Rehabil Assist Technol.* **1**(1), DOI: 10.2196/rehab.3151, 2014

[4] L.J. Manso, P. Bustos, P. Bachiller and P. Núñez. A Perception-aware Architecture for Autonomous Robots. *International Journal of Advanced Robotic Systems* 12(174), pp.13. DOI: 10.5772/61742. 2015.

[5] S. Coradeschi, A. Loutfi and B. Wrede, A Short Review of Symbol Grounding in Robotic and Intelligent Systems, *Künstliche Intelligenz* **27** (2), 129–136, 2013

[6] A.M. Flynn and R.A. Brooks, Battling Reality, *MIT AI Memo* 1148, 1989

[7] E. Gat, On Three-Layer Architectures, *Artificial Intelligence and Mobile Robots*, 195-210, Cambridge: MIT Press, 1998

[8] R. Hartley and F. Pipitone, Experiments with the subsumption architecture. *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 1991

[9] V. Katter and N. Mahrt. Reduced representations of rooted trees. *Journal of Algebra* 412, 41-49, 2014

[10] L.J. Manso, L.V. Calderita, P. Bustos, J. García, M. Martínez, F. Fernández, A. Romero-Garcés and A. Bandera, A genera-purpose architecture for control mobile robots. *Workshop on Physical Agents (WAF)*, 105-116, 2014

[11] A. Romero-Garcés, L.V. Calderita, J. Martínez-Gómez, J.P. Bandera, R. Marfil, L.J. Manso, A. Bandera and P. Bustos, Testing a fully autonomous robotic salesman in real scenarios. *IEEE International Conference on Autonomous Robots Systems and Competitions*, 2015