Use and advances in the Active Grammar-based Modeling architecture

L.J. Manso, L.V. Calderita, P. Bustos, A. Bandera

***Abstract*—The choice of using a robotic architecture and one of its possible implementations is one of the most crucial design decisions when developing robots. Such decision affects the whole development process, the limitations of the robot, and changing minds can be prohibitively time consuming. This paper presents the redesign and the most relevant implementation issues of the Active Grammar-based Modeling architecture (AGM), as well as the latest developments thereof. AGM is flexible, modular and designed with computation distribution and scalability in mind. In addition to a continuous refactoring of the API library and planner, the most relevant improvements are an enhanced mission specification syntax, support for representations combining symbolic and metric properties, redesigned communication patterns, and extended middleware support. A few use examples are presented to demonstrate successful application of the architecture and why some of its features were needed.**

***Index Terms*—robotic architectures, artificial intelligence**

I. Introduction

Robotic architectures aim to define how the different software modules of a robot should interact. They are of crucial interest because multiple properties of the robots are affected by the architecture used. Most implementations of advanced architectures provide users with reusable domain-independent modules, such as executives or planners, to avoid forcing users reinventing the wheel for every robot. These implementations are important for the roboticists because they influence the development process, the range of middleware and programming languages supported. Modularity and scalability are also affected by the concrete implementation of the architecture.

Almost all advanced autonomous robots rely on some form of a three-tiered robotics architecture (see [4]) in which one can find a reactive layer, a plan execution layer and a deliberative layer that monitors the state of the robot's internal world model to update the plan. The main differences come from implementation issues such as the middleware used, how intermediate-level modules and high-level modules such as the planner and the executive communicate, or how is the robots' internal model represented. The Active Grammar-based architecture (AGM) is no exception in this vein: it proposes a detailed description of how the software modules of the robots' can interact and a world model structure.

In particular, AGM world models are multi-graph structures with typed nodes (symbols) and edges (predicates providing relationships between symbols), where the nodes can be attributed with metric properties. These attributes are supposed not to affect the plan, since the planner do not take them into account[1]. Formally, these graph models can be defined as tuples $G = (V, E)$ where V is the set of nodes and E the set of edges. Nodes are tuples $n = (i_n, t_n, a_n)$, where i_n is a string identifier, t_n is represents the type of the node and a_n is a string to string mapping used to store an arbitrary number of attributes for the symbols. Edges are tuples $e = (s_e, d_e, t_e)$, where s_e and d_e are the identifiers of the source and destination nodes of the edge and t_e is a string used to define a type for the edges. This kind of graph is used to represent the robots' knowledge, to describe how the world models can change and to specify missions. See figure 2.

The architecture is built around the representation. There is a deliberative module that, based on the robot's domain and goal, proposes a plan. The rest of the architecture is composed of a pool of semi-autonomous software modules –named *agents* in this context, as in M. Minsky's *Society of Mind* [10]– which are given the plan, can read and modify the representation, and are in charge of performing a subset of actions. In the simplest scenario each action is executed by a single agent, but actions can also be executed by multiple agents in collaboration. Agents can in turn be arbitrarily modularized, controlling their own software modules. The deliberative module is also in charge of managing the representation, accepting or rejecting the update proposals made by the agents. The diagram of the current architecture is shown in figure 1.

Depending on their purpose, the behavior of these agents ranges from blindly contributing to the execution of multiple actions of the plan to a pure reactive behavior[2]. Regardless of this, they can modify the model or read it in order to make the robot reach its goal. For example, the behavior of some perceptual agents can be purely reactive, behaving independently of the current plan (*e.g.*, an agent in charge of passively detecting persons). On the other hand, there can be agents in charge of modifying the model when necessary, not even performing physical actions.

Domain descriptions are sets of graph-rewriting rules describing how an action or percept can modify the robots' world models. Each rule in these sets is composed of a left-hand side pattern (LHS) and a right-hand side one (RHS), and states that, starting from any valid world model, the substitution of the pattern in the LHS by the pattern in the RHS yields a

L.J. Manso, L.V. Calderita and P. Bustos are with the Computer and Communication Technology Department of Universidad de Extremadura.
e-mail: {lmanso,lvcalderita,pbustos}@unex.es
A. Bandera is with the Electronic Technology Department of Universidad de Málaga.
e-mail: ajbandera@uma.es

[1]Metric information that can potentially affect the plan can be *symbolized* using regular predicates.

[2]Agents are given the full plan, not just the first action, so they can anticipate further actions (*e.g.*, robots can lift their arms to prepare for grasping tasks when walking towards the corresponding object).

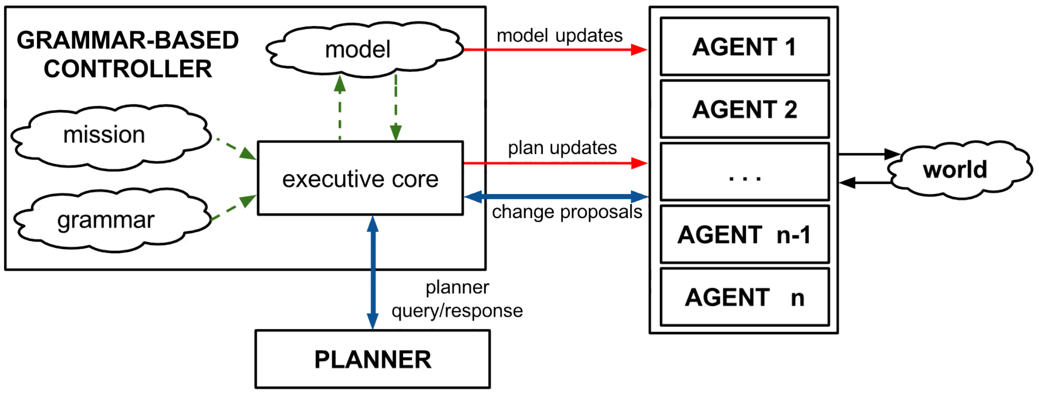


Fig. 1: Redesigned diagram of AGM. The grammar-based controller is composed of the *executive*, the *mission* specification, the *world grammar* (*i.e.*, domain description) and the *world model*. The *planner* is used by the *executive* to find plans and verify change proposals. The *agents* interact with the world perceiving or acting according to the plan, and propose to the executive model updates to acknowledge new information gathered from the environment or their actions. The executive then broadcasts to the agents those change-proposals that are found to be valid. The style of the arrow represents the nature of the information flow: dashed, thick and thin lines, mean direct access, RPC-like and publish/subscribe communication, respectively. The only modification in the diagram from the one in [8] is that change proposals are sent to the executive by RPC.

valid world model. This information is used by the executive to compute plans and to verify that the world models held by the robots are valid. See [6] for a deeper description of the used formalism.

The limitations detected in the earliest versions of the architecture and the new features implemented are described in section II. To illustrate how AGM can be used, a selection of concrete examples of use is presented in section III. The conclusions and future work are presented in section IV.

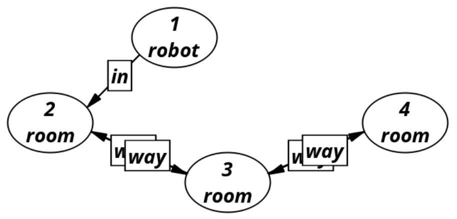
II. New features

Since it was first presented in [8] the Active Grammar-based architecture (AGM) has been used in multiple scenarios and robots [11], [9], especially in the CORTEX architecture [1], which is built on top of AGM. This extensive use has raised several limitations and feature requests. This section describes how the AGM architecture has evolved over time as the result of the experience gained through its use.

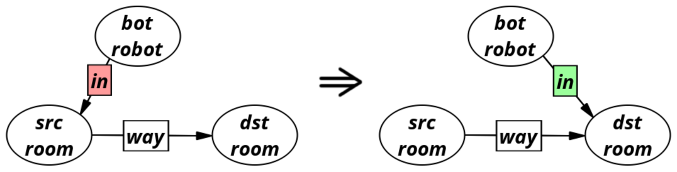
A. Flexible mission definition

AGM targets, as proposed in [8], were defined as concrete patterns sought to be found in the robot's world model representation. These patterns were specified using a graphical model. Despite this approach is moderately easy to understand and is enough to describe most missions (*e.g.*, bringing objects, going to places, even serving coffee [7]), these specifications could only use conditions that were explicitly represented. A simple example of this limitation can be seen in Table I, where a clean-the-table mission is specified using a) the initial approach and, b) the current approach. Using the former approach the ”table clean” condition had to be explicitly marked, whereas the current approach allows avoiding such restriction.

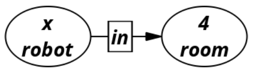
Graphical models are easy to read and share with domain-experts and have been demonstrated to be faster and less error-prone. With the idea that increasing the expressive power of the visual language with disjunctive and quantified formulas would keep domain reading easier than using a textual language, it was decided to add a new optional textual section to the mission definition. This section is applied in conjunction with the graphic section. Its syntax is similar to the one of PDDL [2]. The solution in Table I.a is clearly easier to read



(a) World model example describing three rooms connected in a row. The robot is in one of the extremes of the room sequence. The number in the top of each symbol is its identifier, the word in the bottom is its type.



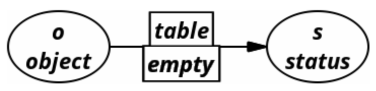
(b) Rule example (left and right hand side of the rule). It describes using a graph-grammar rule the preconditions and how the world model is modified when the robot moves from one room to another. Note that variable identifiers are used in rule definitions so that they can be used with any set of symbols satisfying the conditions.



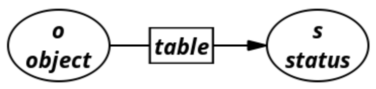
(c) Mission example in which a robot is required to be located in the room identified with number 4. Note that mission definitions can combine concrete identifiers (with numbers) and variable ones (the robot x in this case).

Fig. 2: Examples of how the AGM architecture uses graphs to describe a) world models, b) domain descriptions, and c) missions.

a) initial proposal:



b) current solution:



```
(forall something:object
   (not (in something o))
)
```

TABLE I: A clean-the-table mission specified using a) the initial approach, b) the current approach.

by domain experts than the one of Table I.b. However, this approach requires continuously monitoring and updating the world model to create and remove this kind of edges. Since using the textual section is not mandatory, users can decide which approach to use.

B. Native support for hybrid representations

The first robots using AGM used a single global reference frame for geometric information, so the pose of the objects could easily be specified in the attributes of their corresponding symbols[3]. However, as the number of robots using the architecture increased, the need for a methodical way to represent kinematic trees with different reference frames became clear. Being able to extract this kinematic information with an easy-to-use API was also deemed desirable.

Among all the possible approaches taken into account, the decision was to include special *transformation* edges describing the kinematic relationships. The rest of the options were discarded because they made kinematic structures harder to understand at first sight or because their impact in the models was bigger (regarding the number of additional symbols and edges to include). However, to implement these edges, which are identified by the *"RT"* label, it was necessary to enable edge attributes (a feature that the first AGM proposal missed). From a formal point of view, only the definition of edges is affected by this change. Edges are now defined as tuples $e = (s_e, d_e, t_e, a_e)$, where s_e and d_e are the source and destination nodes of the edge, respectively, t_e is a string used to define a type for the edges and a_e is a string to string map used to store an arbitrary number of attributes for the edges.

While the initial world models of the robots in AGM are defined using an XML format, their kinematic trees are usually defined using specialized file formats such as URDF [12] or InnerModel [5]. It would be time-consuming and error prone to manually include and update these kinematic structures in the robots' world models if desired. Therefore AGM provides two tools to automate the task:

[3]Keep in mind that these models were directed multi-graphs too, where nodes and edges were typed. Additionally, nodes (only nodes) could have optional attributes to store non-symbolic information.

- **agminner** is a tool provided to include the kinematic structure of the robot in it's internal model file.
- **libagm** is a C++ library which, among other purposes such as general access to the graph model (see section II-D), can be used to extract/update kinematics specific objects with geometry-oriented API from the model. Currently, libagm supports InnerModel kinematic definitions.

One of the advantages of using graph-like structures to represent the robots' knowledge is that they are easy to visualize. However, including their kinematic structures in the model makes these models too large to be visualized. To overcome this issue the AGM's model viewer was endowed with options that hide part of this information. See Figure 3.

C. Improvements in the communication strategy

Model modification proposals were initially published using one-way communication with the executive so agents did not get an answer when publishing new proposals. This occasionally made change proposals be silently overwritten if new ones arrived at a fast pace. The issue has been tackled by substituting the one-way publishing mechanism which raises an exception in case the executive is busy or the modification is not correct.

Since the support for hybrid models the frequency of model modifications increased dramatically. Publishing the whole model with every change increased the cost of maintaining complex hybrid models. For example, maintaining human models involved updating a high number of joints and their reference frames per second. The implemented solution was to allow updating edge and node attribute maps without the need of publishing the whole model: one at a time or –in order to decrease the network overhead– multiple maps per request.

D. libagm

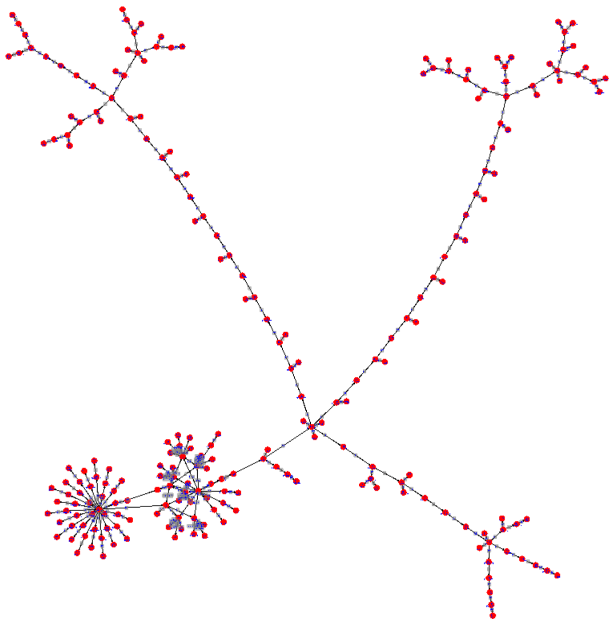
Agents receive the actions they have to perform as part of the plans sent by the executive. To perform their corresponding tasks they have to access the symbols included in the current action, probably other additional symbols and, eventually modify the graph accordingly. Programmatically implementing some of these tasks is time-consuming and error-prone, so libagm has included in its API new methods to ease:

- accessing the symbols in the current action
- accessing edges given the ending symbols
- iterating over the symbols connected to a specific symbol
- publishing modified models, making transparent the middleware-dependent serialization
- printing and drawing the models

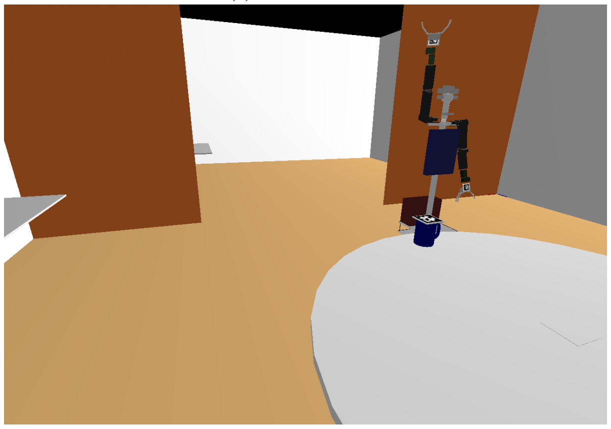
among other minor improvements and those commented in section II-B.

Additionally, a detailed description of the API and examples of its use have been written and published in the web[4].

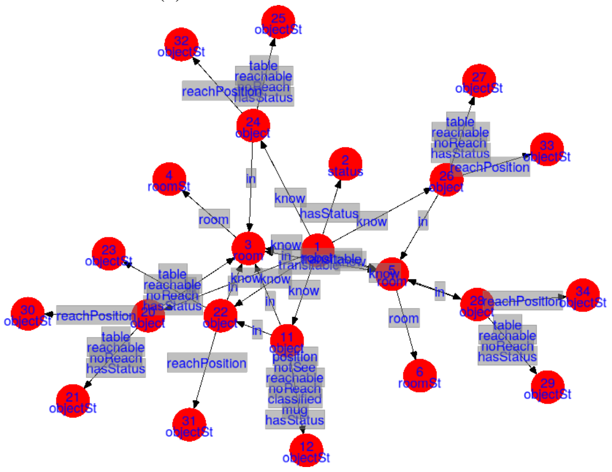
[4]http://www.grammarsandrobots.org



(a) Whole model.



(b) Geometric view of the model.



(c) Model after filtering geometry-only nodes.

Fig. 3: Example of a complex AGM model: a) as it is, b) its geometric view, c) filtering geometry-only nodes.

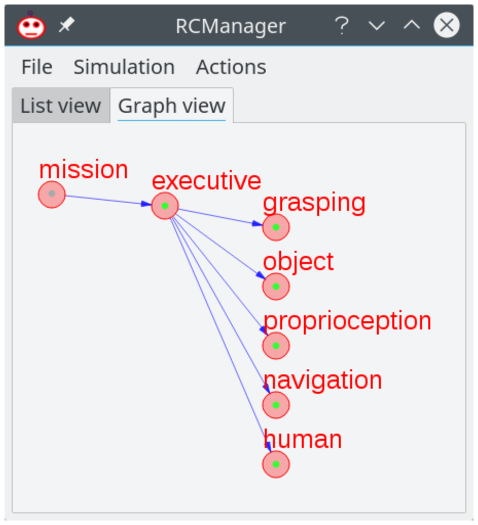


Fig. 4: Deployment network of the high level components used in these examples of use.

E. Middleware support

The first version of the architecture supported the RoboComp framework [3] which, despite of being a full-featured framework, its user base is quite limited. Learning to use a new framework or middleware is time-consuming, so roboticists may not be sufficiently motivated to change. To overcome this limitation, support for multiple frameworks is now underway. The current solution allows cooperation between agents implemented using different frameworks, specifically, RoboComp and ROS. The new executive implements services for both frameworks and, for each topic to subscribe to or publish, there is a RoboComp and a ROS version. Of course, since each of the agents can be programmed using these frameworks, they can in turn use other lower-level components implemented in the supported frameworks.

III. USE CASES

Instead of describing a full-featured robot domain we will introduce several common robot tasks and how they were solved in a real robot using the new features of the architecture. Figure 4 depicts the executive and the set of agents used in these examples along a mission viewer.

A. Changing rooms

For changing the room in which the robot is located it uses the rule described in figure 2b. The robot and the rooms are explicitly represented using *robot* and *room* symbols, respectively. The current semantic location of the robot is represented using a link labeled as *in* from the robot to the corresponding room symbol. The rule depicted in figure 2b describes how, given two rooms *src* and *dst* so that the robot (*bot*) is located in *src* and there is a link labeled as *way* from *src* to *dst*, the robot can change rooms. The result of such action is that the link from *bot* to *src* is removed and a new link from *bot* to *dst* is created. Note that, as in the previous versions of AGM, removed elements are highlighted in red

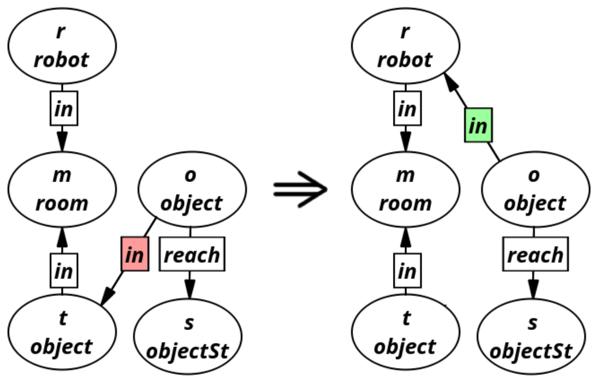


Fig. 5: Graph-grammar rule describing how object grasping affects the models.

whereas those created as a result of the rule are highlighted in green.

From the point of view of the execution of the rule, there are two independent processes involved in the navigation agent. First, when the executive requests the change room rule to be triggered, the agent sends the corresponding command to a lower level robot navigation component in charge of actually moving the platform. Concurrently, it continuously monitors the pose of the robot using another lower level localization component. The limits of the rooms are specified as an attribute of each room's symbol, so when the robot actually changes the room the *navigation* agent can update the model correspondingly, modifying the *in* edge as depicted in figure 2b.

B. Grasping

Grasping objects is slightly more complex than changing rooms. If there is a robot r and a table t containing an object o in a room m, if the o is reachable by the robot, such object can change from being in the table t to being in the robot.

For this task, we benefited from the hybrid models support combining symbols, relationships between the symbols, and geometric information. Since the object to grasp and the robot itself are represented in the model (as in figure 3a) we can extract an InnerModel object (an instance of a class used to support geometric calculations) to compute the necessary geometry-related computations.

As described in section II-A, continuously monitoring the model to highlight logical conditions that can be computed by the planner (based on symbols and/or edges) is not necessary anymore. However, geometric conditions are still necessary to be continuously monitored. It is the case of the *reach*/*noReach* edge that links each object and its status symbol. Depending on whether the robot can or cannot reach an object, the *grasping* agent, modifies the corresponding link. Only when it is reachable is that the robot can actually grasp an object. Therefore, before grasping an object an approaching action is requested by the executive if necessary (*i.e.*, if the object is not close to the area of the space where the robot can grasp objects).

If, for any reason, the robot platform starts moving, the grasping action is paused. To implement this the *grasping* agent needs to monitor the movement of the robot platform. However, access to the platform by the grasping agent is not recommended. Since it is included in the world model, the robot could just monitor the position of the robot in the world. However, the position of the robot in the world is given by a localization component and the given pose may slightly vary even if the robot is still. In order to make the *grasping* agent able to monitor the raw odometry without providing it with access to the platform, the *navigation* agent is also in charge of including and updating the amount of movement of the robot in the last seconds in a ”movedInLastSeconds” attribute of the *robot* symbol.

C. Human representation

As introduced in section II-C, maintaining geometric human models involves updating a high number of reference frames per second. In particular, assuming that for every human a total of 15 joints are tracked at $30Hz$, it would require 450 updates per second per person. If the agent *human*, the one in charge of this task would have to perform a remote call to the executive per joint, it would require 900 remote calls per second to track two humans in real time.

To ease this issue the new interface of the AGM's executive allows modifying multiple edges with just one call, and without publishing the whole model. This made the frequency of the agent go from 2Hz to 30Hz. Instead of publishing the edges it would have also been possible to perform structural change proposals sending the whole world model, but this kind of behavior would slow the rest of the agents down because they would have to extract their corresponding extracted InnerModel objects much frequently.

IV. Conclusions and future work

The improvements of the AGM architecture and the software provided were presented. Section III described how can different tasks be implemented using AGM.

There are two current lines of work to improve AGM. First, support for hierarchical reasoning is underway. It will allow the planner to avoid taking details into account until necessary. Second, efforts are being made toward a decentralized representation. Currently, the executive holds the *valid* world model, however, it would be interesting to distribute the issue. To this end, each agent proposing structural changes would have to perform the model-checking mechanisms that the executive currently performs.

Acknowledgment

This work has been partially supported by the Spanish Ministerio de Economa y Competitividad Project TIN2015-65686-C5-5-R, by the Extremaduran Goverment fund GR15120 ”Ayudas a Grupos” and FEDER funds.

References

[1] Luis Vicente Calderita Estévez. *Deep State Representation: an unified internal representation for the robotics cognitive architecture CORTEX*. PhD thesis, Universidad de Extremadura, 2016.

[2] D. McDermott et al. PDDL: the planning domain definition language. Technical Report DCS TR 1165, Yale Center for Vision and Control, 1998.

[3] L.J. Manso et al. RoboComp: a Tool-based Robotics Framework. In *Simulation, Modeling and Programming for Autonomous Robots*, pages 251–262. Springer, 2010.

[4] E. Gat. On three-layer architectures. *Artificial intelligence and mobile robots*, pages 195–210, 1998.

[5] Marco Antonio Gutierrez Giraldo. Progress in robocomp. *Journal of Physical Agents*, 7(1):38–47, 2013.

[6] L.J. Manso. *Perception as Stochastic Sampling on Dynamic Graph Spaces", school=Escuela Politécnica de Cáceres, Universidad de Extremadura year=2013*. PhD thesis.

[7] L.J. Manso, P. Bustos, R. Alami, G. Milliez, and P. Núñez. Planning human-robot interaction tasks using graph models. In *Proceedings of International Workshop on Recognition and Action for Scene Understanding (REACTS 2015)*, pages 15–27, 2015.

[8] 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):13, 2015.

[9] Jesús Martınez-Gómez, Rebeca Marfil, Luis V. Calderita, Juan P. Bandera, Luis J. Manso, Antonio Bandera, Adrián Romero-Garcés, and Pablo Bustos. Toward social cognition in robotics: Extracting and internalizing meaning from perception. In *XV Workshop of Physical Agents (WAF 2014), León, Spain*, pages 93–104, 2014.

[10] Marvin Minsky. *Society of mind*. Simon and Schuster, 1988.

[11] Adrián Romero-Garcés, Luis Vicente Calderita, Jesús Martínez-Gómez, Juan Pedro Bandera, Rebeca Marfil, Luis J. Manso, Pablo Bustos, and Antonio Bandera. The cognitive architecture of a robotic salesman. *environment*, 15(6):16, 2015.

[12] Martin Theobald, Mauro Sozio, Fabian Suchanek, and Ndapandula Nakashole. Urdf: Efficient reasoning in uncertain rdf knowledge bases with soft and hard rules. 2010.