International Journal of Advanced Robotic Systems

A Perception-aware Architecture for Autonomous Robots

Regular Paper

Luis J. Manso[1]*, Pablo Bustos[1], Pilar Bachiller[1] and Pedro Núñez[1]

1 University of Extremadura, Cáceres, Extremadura, Spain
*Corresponding author(s) E-mail: lmanso@unex.es

Received 07 November 2013; Accepted 08 October 2015

DOI: 10.5772/61742

Abstract

Service robots are required to operate in indoor environments to help humans in their daily lives. To achieve the tasks that they might be assigned, the robots must be able to autonomously model and interact with the elements in it. Even in homes, which are usually more predictable than outdoor scenarios, robot perception is an extremely challenging task. Clutter, distance and partial views complicate modelling the environment, making it essential for robots to approach the objects to perceive in order to gain favourable points of view. This article proposes a novel grammar-based distributed architecture, designed with reusability and scalability in mind, which enables robots not only to find and execute the perception-aware plans they need to achieve their goals, but also to verify that the world representation they build is valid according to a set of grammatical rules for the world model. Additionally, it describes a real-world example of use, providing qualitative results, in which a robot successfully models the room in which it is located and finds a coffee mug.

Keywords Perception-aware Planning, Robot Architectures, Active Perception, Semantic Perception

1. Introduction

Autonomous robot perception is extremely challenging. The scenarios in which domestic robots have to carry out their duties are far from ideal: distant and partial views, clutter and noise are some of the most significant issues that autonomous robots have to deal with. To overcome these factors, robots have to move so that they can achieve favourable points of view of the elements in their environment. These actions are seldom trivial and generally have to be planned, since robots might be asked to find objects which are out of sight or even in other rooms. Figure 1 depicts a classic example in which a robot is asked to find a mug: it will generally encounter scenarios like those in Figure 1(a), and it will have to find plans that include verifying that the obstacle ahead is a table, approaching it, and searching for possible objects that may eventually be modelled as mugs. Moreover, the individual elements perceived must be integrated into a representation of the world to be later used for planning purposes.

In household scenarios, most of the complexity of the activities that robots perform is still related to perception [1]. Pure symbolic models such as semantic networks [2] are not rich enough because they cannot represent the metric properties of the environment. On the other hand, planar metric-only representations lack semantic and topological information, though they are useful for human-robot interaction and improving planning efficiency. Hybrid models, encompassing low-level geometrical details along with symbolic predicates, are a promising research direction [3]. In brief, robots must be equipped with mechanisms allowing them to create perception-aware plans and turn these plans into actions that create hybrid models of the environment which can, in turn, be used to plan and execute the robots' missions.

To generate this kind of perception-aware resources, it is of interest to reuse existing state-of-the-art AI planners. Most of these planners are based on the Planning Domain Definition Language (PDDL, see [4]), which was designed to be the reference language for the International Planning Competition and has become the most widely-used language in automated planning. However, it was not designed with perceptual tasks in mind and has two drawbacks when these tasks are required: i) it does not support creating or deleting symbols nor modifying their types [5], and ii) it is not human-friendly for people who are not used to it. Online symbol creation and deletion are commonly required in perception-related tasks *e.g.*, when the goal is to perceive a new world element or it is required as part of a plan. This limitation of PDDL can be circumvented by including a list of unused symbols and additional code to handle them. Changing the types of the symbols as a consequence of active perception is also frequently necessary. For example, we might want robots to approach what, at a certain distance, they think is an *obstacle* to check if it is actually something that they know. Taking into account this kind of actions when planning requires a language that allows changing the types of the symbols dynamically. Again, this limitation can be circumvented by translating types to PDDL predicates, but these workarounds decrease efficiency and make the resulting PDDL code harder to read (see Listing 1). Although it might be seen as secondary, human-readability can be crucial because it influences the number of programming errors developers introduce and their overall efficiency. It also reduces the number of people that can understand robot planning domains, which is interesting when the domains must be supervised by multidisciplinary groups. Additionally, although some technologies such as hierarchical task networks [6] provide other types of advantages over PDDL, no alternative supporting dynamic typing and symbol creation is yet known.

To overcome the previously mentioned planning limitations and –most importantly– in order to enable robots to **execute** perception-aware plans and **verify** their outcome, this paper proposes using a novel architecture named *Active Grammar-based Modeling* (AGM) which uses a new domain definition language. AGM has been designed with reusability and scalability in mind and provides hybrid representations (combining symbolic and metric information). It is worth highlighting that AGM can be used to enable robots to verify that the models built are valid according to a set of world grammar rules (which helps in detecting and preventing invalid model modifications) as well as to demonstrate different perceptual phenomena such as bottom-up parsing, action selection (both for perception and mission completion), covert perception and the inclusion of context-dependent perceptual restrictions.

In AGM, perception and action are the result of the controlled interaction of a set of software modules that propose modifications to the robot's world model and, depending on the module, perform physical actions. Given the goal of the robot and the current world model, a *planner* provides the *executive* of the architecture with a sequence of valid world model modifications (some of them triggered by perceptual events, others by robot's actions) that would take the current world model to a state in which the goal of the robot is met. The executive forwards plan updates to the modules so that they can reconfigure themselves accordingly. When one of these modules proposes modifying the world model (*e.g.*, because an action has been achieved or because a new object is detected), the executive checks that such modification is valid and, if it is, broadcasts the new model and plan to all the modules, closing the control loop (see Figure 2).

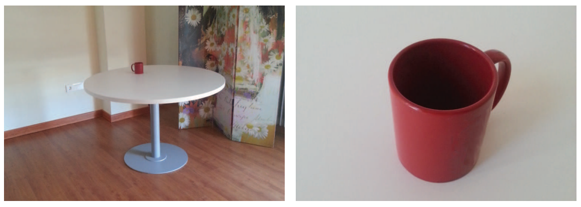

(a) Realistic initial scenario. (b) Favorable point of view.

Figure 1. Generally, segmentation and classification algorithms assume favourable points of view and scenarios in which objects are easy to segment (see Fig. 1(b)). Although domestic environments are structured to some extent, the case depicted in Figure 1(a) is much more likely. Getting from the situation of Fig. 1(a) to that of Fig. 1(b) usually requires an appropriate plan.

In AGM, the world model modification rules that are used for perception-aware planning and model checking are described using a visual grammar language named *Active Graph Grammar Language* (AGGL). AGGL was designed with perceptive tasks in mind (*i.e.*, it supports symbol creation, deletion and dynamic typing) and, thanks to the visual editor provided (see section 4.1), it is easier for humans to understand in comparison with PDDL (see example in Listing 1). Once defined, grammars can be used for planning, whether by translating them to PDDL (so that they can be used with the most common AI planners without dealing with the previously mentioned workarounds) or directly using a specific planner (*AGGLPlanner*, whose implementation details are out of the scope of this paper).

The remainder of the paper is organized as follows. Section 2 introduces the state of the art. Section 3 provides detailed information about the AGM architecture and how to design and implement systems based on it. Section 4 introduces Active Graph Grammar Language and how domain definitions can be defined using it. Section 5 highlights the different phenomena that can be achieved using AGM. Section 6 describes software-related issues. Section 7 provides a real-world example of use in which a domestic robot is commanded to find and model a coffee mug in a previously unknown environment. Because the advantag-

```
(: action  detectObstacle
  :parameters ( ? status  ?room ?gualzru  ?ListAGMInternal ? list0  )
  :precondition (and ( ISstatus  ? status ) (ISroom ?room) (ISrobot  ?gualzru ) (firstunknown  ? list0 ) (unknownorder ? list0  ?ListAGMInternal)
    (not(= ? status  ?room)) (not(= ? status  ?gualzru )) (not(= ? status  ?ListAGMInternal)) (not(= ? status  ? list0 )) (not(= ?room ?gualzru ))
    (not(= ?room ?ListAGMInternal)) (not(= ?room ? list0 )) (not(= ?gualzru  ?ListAGMInternal)) (not(= ?gualzru  ? list0 ))
    (not(= ?ListAGMInternal ? list0 )) (in  ?gualzru  ?room) (notFullyExplored  ?room ? status )  )
  : effect  (and (not (firstunknown  ? list0 )) (not (unknownorder ? list0  ?ListAGMInternal)) (firstunknown  ?ListAGMInternal)
    ( ISobstacle  ? list0 ) ( contains  ?room ? list0 )
)
```

Listing 1. Listing PDDL counterpart of the AGGL rule described in Figure 4

es of using AGM are qualitative, the results are also qualitative: it endows robots with abilities that they previously lacked (*i.e.*, perception-related planning and model verification). Section 8 summarizes the conclusions of the paper.

2. State of the art

A grammar can be defined as a theoretical tool used to describe the rules governing the formation of specific sets of structures. Graph grammars generalize the concept of string grammars (those generally used in natural and computer languages) so that they can also be applied to graphs with indefinitely complex connection patterns. In fact, strings can be seen as undirected graphs where all the nodes (characters), except those at sentence endings, have an edge linking them to an adjacent character on their right. They have been used in robotics and artificial vision for a wide range of applications. The work presented in [7] proposed an algorithm for graph grammar verification, *i.e.*, to verify the properties of the graphs generated by using the rules of a specific grammar. [8] describes a set of graph grammar rules over fixed-order graphs that can be used to achieve self-configuring adaptable software architectures. A similar approach for coordinating multi-robot systems where robots are represented by graph vertices is proposed in [9]. The graphs, which are shared by all the robots of the system, are also of fixed order. Coordination is achieved by modifying the linking pattern and the label (*i.e.*, role) of the robots. In [10, 11] and [12], the authors present a series of related works. In [10] and [12], an attributed graph grammar is designed in order to parse images taken from man-made scenes. It uses projected 3D rectangles as terminal nodes and different rectangle layouts as production rules. Bottom-up and top-down mechanisms are used in order to improve rectangle detection and parsing. Since different possible models can explain input images, the algorithm chooses the one that maximizes the posterior probability. The work presented in [11] follows the same approach for generic scenes. In [13], a similar approach to that described in [12] is used for object recognition. In this case, both the set of primitives and the production rules are wider but the foundations are the same. These approaches have two main differences from what is proposed in this paper: a) they use static input data, which is a very hard restriction for robotics; b) they are based on string grammars instead of graph grammars, reducing the number of problems that can be solved. In [14], grammars are used to describe plans and how can they be dynamically modified as sub-tasks are accomplished or as conditions change. In [15] *Spatial Random Tree Grammars* (SRTG) are proposed for image parsing. SRTG's are context-free grammars in which rules are labeled with information for determining the spatial distribution of their nodes (*i.e.*, vertically or horizontally distributed). While in string grammars this is not necessary (*i.e.*, productions are always horizontally distributed), it guarantees the unambiguous interpretation of parse trees from graph grammars. In this work, a probability distribution is also associated to production rules so that the probability of a specific parse can be estimated. The work presented in [16] deals with still-image understanding. Although it is not explicitly grammar-based, it uses a generative approach, and the result of the algorithm is a parse graph. The focus of this paper is to provide an image understanding algorithm consistent with physical laws that goes beyond unembodied image labeling.

Even though several works use grammars to generate representations of the contents of images ([10, 12, 13, 17]), all of them approach perception as a passive process where the input data is static and complete from the beginning. The grammars proposed in the present paper not only allow us to define how the world model can be transformed but also allow us to achieve other goals, such as deciding what robots should do to achieve their goals, incremental model checking (the architecture guarantees that the models generated using it are created according to the grammar), and decreasing programming errors (developers do not need to hard-code the grammar or describe what the robot should do in a general-purpose language, and can instead do so in a higher level language that can be graphically visualized).

From the point of view of integrated planning and execution, the dominant approach is to use three-tiered architectures [18], which are composed of a planner, a task sequencer and a skill layer. In [19], a comprehensive classification of many control architectures is provided, attending to the relative weight of the sequencer and the planning modules. There are remarkable early approaches that rely on strong executors, such as Subsumption [20], SOAR [21] and AuRa [22], as well as more advanced interleaving systems that actively gather information from the environment for the planners to generate better plans, such as IPEM [23], ASPIRE [24] and RETSINA [25], and a final category that gathers those systems that focus on a powerful deliberator component, such as DPLAN [26], CASPER [27] and Rogue [28]. Based on these ideas, there have been attempts to design generic architectures that provide a framework to integrate different planning, monitoring and learning algorithms using state-of-the-art component-oriented software techniques such as PELEA [29], MAPGEN [30] and IxTeT [31].

AGM can be seen as an alternative distributed generic solution to the construction of intelligent robots that introduces graph grammars as the formal representation of the dynamics of world states, and as a generative mechanism to build hybrid world models. The **main advantages** of AGM over these approaches is that it supports the planning of perception tasks and model checking and that it allows users to express domains using AGGL, with the advantages this entails.

3. Active Grammar-based Modeling

Active Grammar-based Modeling (AGM) is a robotics architecture with a strong focus on enabling active perception in non-trivial environments, where perception is something that needs to be explicitly planned. The architecture, depicted in Figure 2, is composed of three main elements: a) a *grammar-based controller*, b) an AI *planner*, and c) a set of software modules that can act and propose changes to the world representation —in short, these modules are referred to as ***agents*** from now on. The controller is in turn composed of an executive and three passive elements that are handled by it: the world's grammar, the goal of the robot and the current world model. Globally, AGM works as a loop as follows:

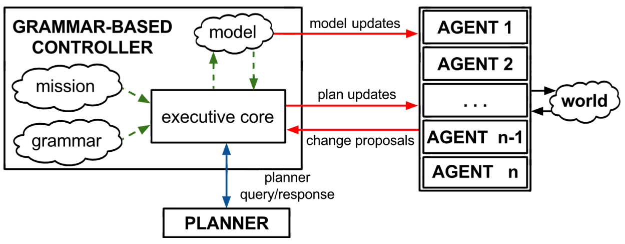


Figure 2. Design diagram of AGM. The *executive* along the *mission*, *grammar* and the *model* form the so-called *grammar-based controller*. The *planner* is used by the *executive* to find plans and verify change proposals. The set of modules that propose changes —referred to as *agents* in the context of AGM, in short — 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 rest of the agents those change-proposals that are found to be valid.

- **Planning** Given the current world state, the goal and the grammar of the world model, each cycle of the architecture starts with the executive asking the planner for the best plan to achieve the goal. Once the planner finds a plan, the executive propagates it to all the agents. In AGM, a plan is a sequence of graph transformations that would take the world model to a state in which the goals are met. Each particular transformation is specified providing the name of the rule to trigger and a map matching the symbols in the rule to those in the model (see Sect. 5.1).
- **Agent reconfiguration** After the plan is broadcast, the executive waits for model modification proposals from the agents. The agents, which have their own control loop, receive the latest plan and modify their behaviour according to it. Although only the first step of the plan (the most immediate modification to perform in the model) is generally taken into account, there are scenarios where the whole plan might be needed[1].
- **Modification proposal and verification** Once adapted to the latest plan, the agents continue operating autonomously until, at some point, one of them proposes a modification of the world model to the executive and it is accepted (*e.g.*, a human-robot interaction module might propose including a new human in the model). The verification of the modification proposals is solved by the executive by posing them as planning problems, whereby the world state is the current model and the goal state is the model proposed: if the planner can find a sequence of modifications that lead the current model to the one proposed, it is considered valid. Finally, when a new model proposal has been accepted, the executive broadcasts the new version to all the agents and a new cycle starts.

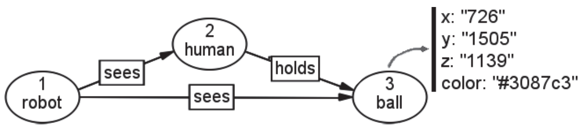


Figure 3. Simple example of an AGM model containing three nodes for representing a human, a robot and a ball. The picture also shows the attributes that a *ball* symbol could have: x, y and z coordinates, and its color.

The models resulting from this interaction among the modules of AGM are graphs with labeled links and attributed symbols, such as that depicted in figure 3.

As pointed out previously, there are tasks for which a pure symbolic model is useless or else inefficient (*e.g.*, object tracking, navigation and grasping). On the other hand, there is no known algorithm able to perform generic reasoning using metric data. To enable robots to take metric information into account when planning, we follow a very simple approach: all the planning done by the AGM executive is performed at a symbolic level (using symbols

1 For example, a human having breakfast in bed who commands the robot to fetch the butter. The plan could be as follows: i) go from the bedroom to the living room, ii) go from the living room to the kitchen, iii) open the fridge, iv) find the butter, v) fetch the butter, vi) close the fridge, vii) go from the kitchen to the living room, viii) go from the living room to the bedroom, and ix) deliver the butter. Let us also assume that the robot has a "butter detector" module. In this scenario, given that the robot could find the butter on its way to the kitchen and that activating the detector would not interfere with robot navigation, the "butter detector" module could be activated every time there is a "find butter" action in the plan.

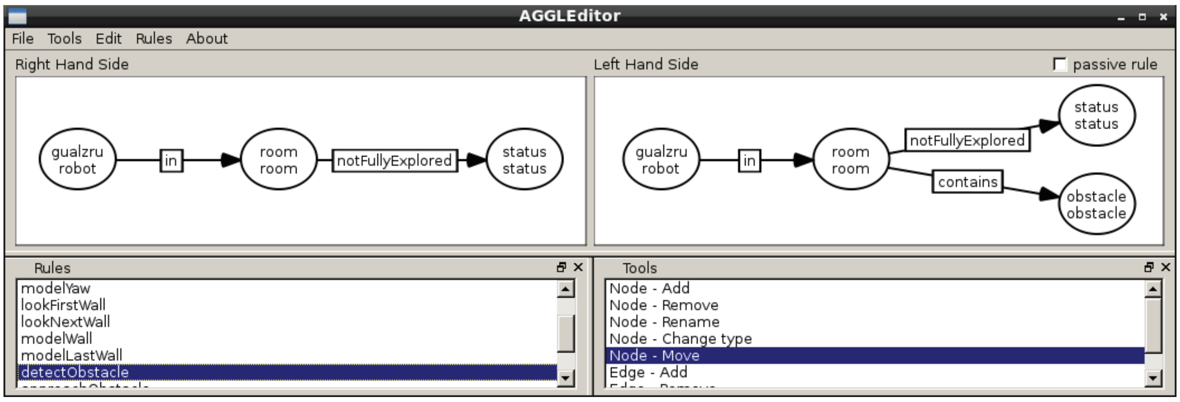


Figure 4. Screenshot of *AGGLEditor*, the tool used to describe AGGL grammars. In this example, the rule states that the robot can create new objects in the room in which it is located so long as the room is not considered "fully explored".

and the edges between them); any metric property supposed to affect the plans has to be *symbolized* by an agent. For example, if we want the robot to talk only to *close* people, we can make one of the agents mark humans as close or far by including or removing edges labeled as "interactionDistance". The attributes of the nodes are not considered in the planning process and are not covered by the grammar because they do not affect the structure of the model. Therefore, the updates for the attributes of the nodes are automatically broadcast (no model verification is involved). Although they are not used for planning, they can be used by other agents (*e.g.*, an agent implementing a tracker can update the coordinates of objects, while another agent that makes the robot grasp objects can use these coordinates).

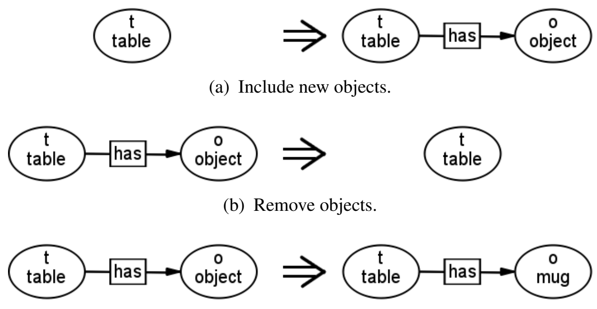
4. Active Graph Grammar Language

Active Graph Grammar Language (AGGL) is the visual language used in the AGM architecture to specify the transformation rules that describe how the world model of the robot can be created and modified. Grammars defined in AGGL can be directly used for planning using a special planner or translated into PDDL to use the grammars with PDDL planners after translation[2].

As opposed to PDDL, in AGGL developers do not describe actions and their effects explicitly. Instead, they describe the possible transformations that the world may incur. Such changes are expressed using graph-grammar rules, such as the one in Figure 5, which are similar to any other grammar rules but which are adapted to graphs. Using the AGGL editor tool (see Fig. 4), the design of AGGL grammars is straightforward –it provides the necessary tools to create transformation rules by including, removing or modifying nodes and edges.

4.1 AGGL rules

As with any other grammar, graph grammars are sets of grammar rules —also known as graph rewriting rules— whereby each one describes a valid transformation that the representation may incur. They are described using pairs of patterns $G_1 \Rightarrow G_2$, meaning that the pattern G_1 can be substituted with the pattern G_2 (see Figs. 4 and 5). The left-hand side of the pair is usually referred to as the LHS and the right-hand side as the RHS.



(a) Include new objects.

(b) Remove objects.

(c) Transform undefined objects into mugs.

Figure 5. Examples of simple graph-grammar rules

In AGGL, each symbol in the grammar rules is represented by a node with two strings: an identifier (used to match the symbols on the LHS with those on the RHS), and another string denoting the *type* of the symbol. Relationships between the elements of the model are represented by labeled links. There can be more than one link between two nodes so long as they have different labels.

As opposed to string grammars, graph grammars lack a generally accepted formalism for specifying their behaviour. We chose to use a slight modification of the *double push-out* formalism because of its simplicity and readability [32]. The behaviour of the rules in AGGL is as follows:

- A rule can only be applied if a match between the elements and the connection patterns of the LHS of the rule with those in the model is found.
- In order to apply a rule, the nodes and edges that are present on the LHS but not on the RHS are removed;

2 Even when using PDDL planners, developers using AGGL benefit from avoiding dealing with the necessary PDDL workarounds commented on the introduction.

those present on the RHS but not on the LHS are created; those appearing on both or on no side remain.

- If the origin or end of an edge correspond to a node that will be removed, such edge will also be removed.
- If the type of a symbol in the RHS of a rule differs from the type specified in the LHS, the type of the symbol is changed as a result of executing rule (see Fig. 5(c)).

For example, rules can be used to express the possibility that tables can be associated with new objects in the robot world model (Fig. 5(a)), that objects can disappear from tables (Fig. 5(b)), or that objects on tables can be converted into mugs (Fig. 5(c)).

Additionally, the rules can be set as *passive* or *active*. Active rules are used for planning and model verification while passive rules are only used for model verification. Passive rules are explicitly specified in the grammar to avoid using rules describing *exogenous events* for planning (*e.g.*, the battery discharging or the sun rising).

5. Using Active Grammar-based Modeling

Active Grammar-based Modeling facilitates achieving different perception and reasoning skills. This section describes how some of the most remarkable ones can be implemented.

5.1 Perception-aware planning

As described in Sect. 3, AGGL can be used to describe rules that can later be used to find plans. AGGL allows creating, deleting and modifying the types of the symbols. These operations are extremely common when performing perception-related tasks, such as finding or classifying objects. Using AGM, robots can achieve any kind of task regardless of whether there are perceptive (sub)tasks involved or not. The requirement is that all actions that are desired to be planned using the grammar must have consequences in the symbolic representation of the world (nodes and edges) so that a planner can reason about these actions and their consequences.

The goals of the robot are defined in terms of the graph patterns that are sought in the representation. The symbols in the target patterns can be constants or variables. Constants are represented by symbols with the numeric identifier of any of the symbols in the current model as an identifier, and they are forced to match such a symbol. Variables are denoted by symbols with variable names as identifiers and can match any of the rest of the symbols or else unknown ones. If the robot aims to find a new mug, a valid goal would be a graph containing all known mug symbols (as constants) and a non-existing mug symbol (using a variable). If the mug must be found in the current room, the goal pattern must contain the current room symbol and a non-existing mug symbol connected to it. Any goal pattern is valid so long as it can be achieved after applying a finite sequence of grammar rules to the current model. Therefore, if a robot in the situation depicted in figure 3 is commanded to touch the ball, its *target sub-pattern* (goal) would be that in figure 6(b). Note that, in the example, the symbol corresponding to the human does not appear: this means that (in this specific case) the robot is not interested in its relationship with the human, only in touching the ball; symbols not appearing in the target pattern are not taken into account. Making a symbol constant in the target pattern forces the planner to search for a world model in which such a symbol corresponds to the one with the same identifier in the current model. Goal patterns can be static or set by additional modules.

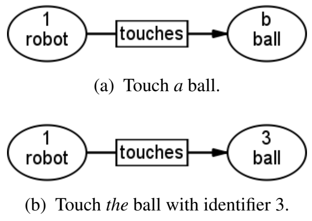


Figure 6. Target patterns that could be used to make a robot touch a ball

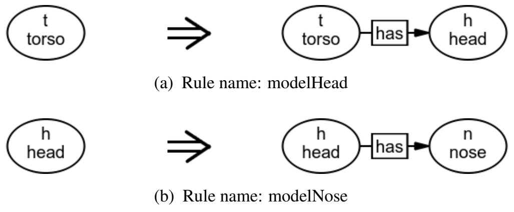


Figure 7. Simple rules that could be used for modelling the head and nose of a human

The result of the planning is a list of transformations that would take the current world model to the target state (the one that defines the mission). Each transformation is specified with the name of the rule and a map matching the variables on the LHS of the rule with the numeric identifiers of the symbols involved in the current model. The following are two examples of possible executions of the rules defined in Fig. 5(a) andFig. 5(c), respectively:

$$\begin{aligned} findObject &: \; t \to 4 \\ modelMug &: \; t \to 4, \; o \to 8 \end{aligned} \quad (1)$$

Execution is achieved by forwarding the computed plan to the agents in each execution cycle. The robot's behaviour is the result of the coordinated actions of all the agents given the current plan.

5.2 Context-aware perception

Although suggesting that robot perception should be adapted to the context is a rather evident thought, the path to achieve context-aware perception appropriately remains an open question. In AGGL, rules describe the context in which a transformation can be applied (on their LHS), and it can be used to define several context-specific rules. For example, if we want robots to behave differently when they are looking for new objects, depending on the location the robot is inspecting (the floor and a table for this example), the grammar would have two different rules differing according to what the robot looks at: one in which the robot appears looking at the floor, and one in which the robot appears looking at a table. Therefore, the first rule could only be applied when inspecting the floor, and the second one could only be applied when inspecting a table. Since the agents are provided with the plan of the robot (the sequence of rules to be triggered), their behaviour can be adapted depending on the name of the first rule to trigger (*e.g.*, *findObjectInFloor* or *findObjectInTable*).

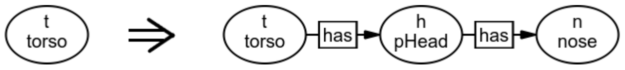
5.3 Context-aware perception restrictions and model verification

The limitations imposed by the grammar on the context can help detecting (and therefore reducing) perceptual errors because some of them will probably not comply with the rules of the grammar and this situation can be detected. To illustrate this, consider the rules in Figure 7:

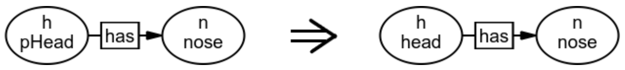
These rules would avoid, for example, attaching a nose to a torso. Since there is no rule (or rule sequence) that would do such thing, the planner used by the executive to verify changes would report that there is no plan that would directly link a nose to a torso. Using this grammar, the only way to perceive a nose would be to detect first the head of the human and only then detect its nose. By making these checks for every possible model-change we can guarantee that the generated models comply with the grammar.

5.4 Covert perception

Covert perception is another interesting application of AGM. In poor conditions, such as when parts of the objects to be detected are occluded or where there is too much noise, bottom-up object detectors tend to decrease their effectiveness dramatically. Grammars can also be used to diminish this issue by enforcing the detection of specific parts of the model (also referred to as covert perception [33]). This means that AGM can not only be used to reduce false positives in object detection but also to reduce false negatives. Consider the rules shown in Figure 8 as an extension of those in Figure 7.



(a) Additional rule allowing to *imagine* a head without actually seeing it (*pHead* is a short name for *proto-head*).



(b) Additional rule allowing to transforming *pHead* symbols to *head* symbols

Figure 8. Simple extension of the rules shown in Figure 7

Introducing these rules, we actually enable the robot to model the head using two different strategies: a) by regular detection using an ad hoc detector/modeller (see rule in Figure 7(a) and b) by detecting a nose, which makes it evident that there is a head supporting it that should be modelled (rules in Figures 8(a) and 8(b)). It should be noted that, in the second case, the information provided by the modelled nose can be used by a "head detector" to improve its effectiveness by providing a good *a priori* based on the information provided by the *nose* symbol.

5.5 Dealing with qualitative metric information

As introduced in Sect. 3, in AGM, metric-only changes are automatically broadcast to all the agents in the system because pure-metric information does not have grammatical meaning and, therefore, does not affect the plans (and it cannot be grammatically checked). If pure-metric information is desired to affect the plans of the robot, one of the agents should introduce structural changes in the model when required. For example, in the case where a robot may collide with another object, one of the agents should be in charge of denoting it by including a new link labeled with an appropriate name, such as "mayCollideWith", between the symbol of the robot and that of the obstacle.

6. Software-related issues

In the absence of proper tools, robots tend to be programmed using hard-coded if-then-else constructs (which are considerably error-prone when the complexity increases) or, at most, fixed plans embedded in state machines (see [34, 35]). Using state machines to embed plans makes the code more structured, easier to understand and less-likely to contain programming errors in comparison with hard-coded logic. However, to use them it is necessary for roboticists to know and include in the state machines all the necessary states and transitions that the robots may need. This is practical for moderately simple robots but, as they perform a few different tasks and their world representations become richer, developing and understanding these state machines becomes difficult. AGM makes use of AI planning to provide a principled, structured and distributed way to enable robots to generate complex representations of the environment and act according to them and the robots' goals.

AGM may be of greater interest if the distributed nature of the concept behind AGM is taken to an implementation level using Component-Oriented Programming (COP) [36, 37]. Implementing each of the active elements of AGM depicted in Figure 2 as actual independent software components provides several desirable features:

- *Computation distribution:* COP is a simple way to distribute computational tasks in different cores and computers.
- *Decoupling:* Undesirable side-effects of hard dependencies between the subsystems of monolithic software are also reduced (*e.g.*, blocked waits).

- *Robustness:* In a distributed system, the failure of a module does not imply the failure of the whole system; any remaining modules can re-initiate faulty ones.
- *Flexibility:* COP makes it easier to prototype, substitute or include new modules (the executive, planner and agents in AGM) and enables roboticists to implement them in different programming languages.

Even using an advanced robotics framework, the development of the rules of grammar-based systems can be error-prone and time consuming if they have to be manually written or specified in textual languages. To make it user-friendly, a graphical editor named **AGGLEditor** was developed (see Fig. 4). The following software is also provided: *a)* a library providing the functionality of the AGM executive (**libAGM**), *b)* a planner based on AGGL rules[3] (**AGGLPlanner**), and *c)* an AGGL-to-PDDL compiler that enables the use of AGGL grammars with the most common PDDL planners (**aggl2pddl**). The library implementing the AGM executive is distributed as a framework-agnostic C++ shared library with Python bindings and an open license (GPLv3).

7. Indoor modelling experiment

This section describes an experiment in which a robot equipped with a differential drive system and an orientable RGBD sensor is requested to model the room in which it is located and find a coffee mug. The goal is complex enough to bring up several challenging issues that force the robot to demonstrate sophisticated perceptual skills. Since the robot is enclosed by a room, it cannot be modelled from a fixed point of view; therefore, the robot must direct its gaze towards different points of interest to be able to provide valid room models. Even when perceiving elements lying in the field of view of the robot's sensors, it can be too close or too far, or else there might be an obstacle between the sensor and what is to be perceived. The robot must actively ensure that the elements will be properly perceived. Additionally, to find objects lying outside the field of view, the robot will frequently need to move not only its head but also its whole body, and these actions are sometimes necessarily planned.

For this experiment, we propose a grammar that makes the robot act in a two-staged fashion. First, the robot must model the room, enabling it to classify input 3D points as belonging to the room or the objects inside it. Afterwards, the robot will enter a loop in which it looks for unknown obstacles, approaches them for inspection and, when a table is found, looks for mugs in the table. In the first stage, the robot has to model its angular orientation (relative to an arbitrary wall) and direct its gaze to different walls to model them. In the second stage, the robot detects unknown obstacles, differentiating them from the walls of the room, and approaches them for a better point of view according to their position. Those obstacles that turn out to be tables are represented using a high-order model for tables, while the rest are modelled as generic obstacles using a bounding cylinder model. Once the robot finds and models a table, depending on its goal, it can try to find and model the objects in the table: mugs are modelled as *mug* symbols using a cylinder model and the rest are modelled as generic objects using bounding sphere models. Although the robot automatically calibrates its camera's height and its joints' offsets using AGM, we assume that the robot is provided with this information to shorten the experiment to an adequate length. The robot is initialized looking forwards with the model described in Figure 9.

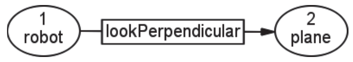


Figure 9. Initial model for the experiment

The remainder of the section describes the symbol-types, rules and agents used in the experiment.

7.1 Symbols

The following list describes the symbol-types that are used in the world model of the robot. Although the attributes of the symbols are ignored at the grammar (reasoning) level, they are also specified in order to make the text easier to understand (note that the symbols with no attributes still have syntactic meaning):

- **robot** Represents the robot itself. It has no attributes.
- **plane** Symbol temporarily used for the ground plane. Attributes: *pitch* angle, *roll* angle, *d* (distance).
- **floor** Symbol temporarily used for the ground plane and the angle to an arbitrarily chosen *zero wall* used as *north* while modelling the room. Attributes: *yaw* angle, *pitch* angle, *roll* angle and distance *d*.
- **notWall** Temporary symbol used to denote walls that are known to exist but have not been modelled yet. No attributes.
- **wall** Symbol used for walls that have been modelled. Attributes: *d* (distance).
- **room** The final symbol for the room. It has the same attributes as the *floor* symbol (*pitch*, *roll*, *yaw*, and *distance*), but *room* symbols can only exist once the four walls have been modelled. The size of the room can be computed using the distance of the robot to each of the walls.
- **obstacle** Used for models of unknown obstacles. Attributes: bounding cylinder (geometric *center*, *radius* and *height*).

3 The implementation details of the planner are beyond the scope of this paper but they are already available in http://grammarsandrobots.org.

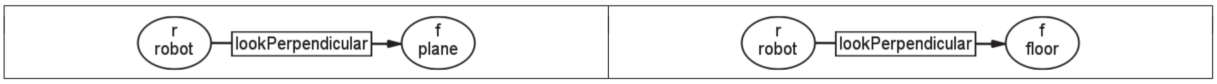


Table 1. Rule 1 is used to denote that the bearing angle of the robot has been modelled. Rule name:“**modelYaw**”.

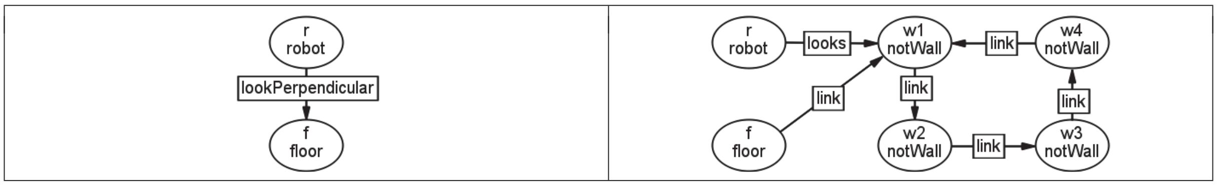


Table 2. Rule 2 is used to look to the first wall. Rule name: “**lookFirstWall**”.

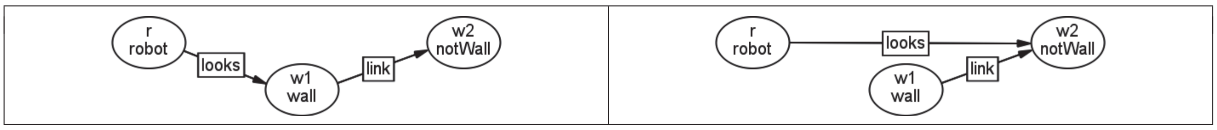


Table 3. Rule 3 is used to denote that the robot has switched its gaze to the next not-modelled wall. Rule name: “**lookNextWall**”.

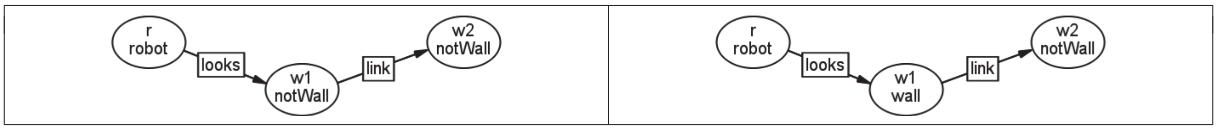


Table 4. Rule 4 is used to denote that the robot has modelled a wall. Rule name: “**modelWall**”.

- **table** Used for models of modelled tables (which, for this experiment, are assumed to be circular). Attributes: geometric *center*, *radius* and *height*.
- **object** Used for models of unknown objects on tables. Attributes: bounding sphere (geometric *center*, *radius*).
- **mug** Used to represent mugs. Attributes: extremes of the cylinder associated to the mug (*a* and *b*) and *radius*.

7.2 Grammar rules

Designing the rules of the grammar is a central task in the development of an AGM -based system. They define what the robot will be able to perceive and do. This section describes the rules defined for the experiment (shown in Tables 1–10).

Rule 1, named *modelYaw* (see table 1), modifies the type of the node *f*, from “plane” to “floor”. *Planes* represent regular planes, while *floors* represent “oriented planes” (*i.e.*, a plane with a *north*). Thus, it is used to denote that the bearing angle of the robot has been modelled.

Rule 2, named *lookFirstWall* (see table 2), performs two interesting modifications in the model: first, it creates a sequence of four “notWall” symbols —not-modelled walls — and links the sequence to the existing floor symbol, and second, it removes the link connecting the robot and the floor labeled “lookPerpendicular” and includes a new one from the robot to the first of the not-modelled walls, labeled as “looks”. It is used to represent that the robot looks towards the wall at *zero* angle and that there are four walls yet to model.

Rule 3, named *lookNextWall* (see table 3), moves the head of the arrow from a modelled wall to the next not-modelled wall. It is used to denote that the robot does not look towards the last modelled wall any more but to the next not-modelled wall (*i.e.*, the robot moved its gaze towards the next wall to model).

Rules 4 and 5, respectively named *modelWall* and *modelLastWall* (see tables 4 and 5), modify the type of walls from “notWall” to “wall”. They are used to indicate that a wall has been modelled. Additionally, rule 5 also changes the type of the floor from “*floor*” to “*room*”, denoting that the room has been completely modelled. The difference is that rule 4 is used to model all the walls but the last one, which is modelled using rule 5.

Rule 6, named *detectObstacle* (shown in table 6), creates and links new *obstacle* symbols to the *room* symbol associated with the robot. It is used to include obstacles (unknown objects located in the floor) in the representation.

Rule 7 is named *approachObstacle* (shown in table 7). It creates a new link labeled “closeLook” from the robot to

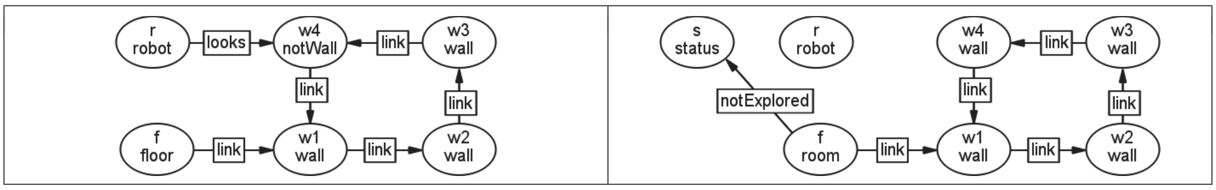


Table 5. Rule 5 is used to denote that the robot has modelled the last wall. Rule name: "**modelLastWall**".

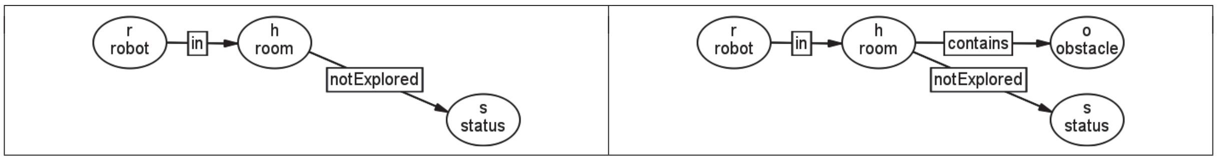


Table 6. Rule 6 is used to include obstacles in the representation. Rule name: "**detectObstacle**".

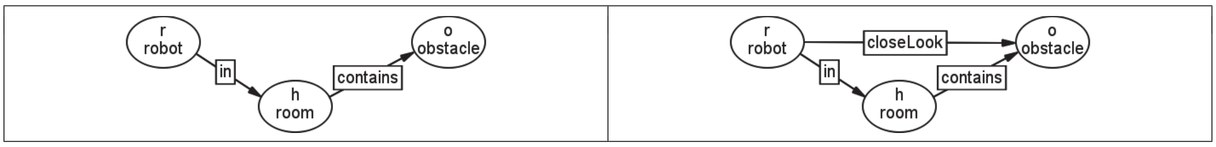


Table 7. Rule 7 is used to denote that an obstacle is being seen by the robot. Rule name: "**approachObstacle**".

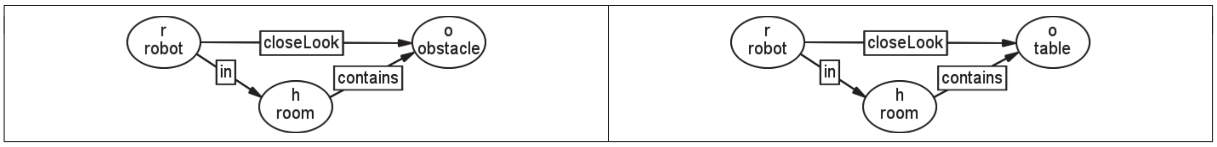


Table 8. Rule 8 is used to denote that an obstacle has been successfully modelled as a table. Rule name: "**modelTable**".

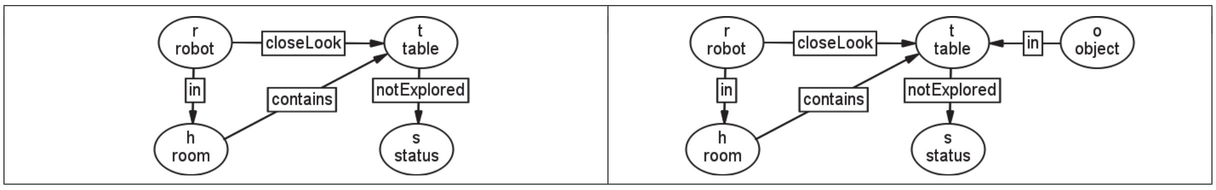


Table 9. Rule 9 is used to include new objects in tables. Rule name: "**findObjectsInTables**".

obstacle symbols. It is used to denote that the robot is looking towards a specific obstacle from a close position.

Rule 8, named *modelTable* (shown in table 8), modifies the type of *obstacle* symbols to *table* symbols. It is used to denote that an obstacle has been successfully modelled as a table.

Rule 9, named *findObjectsInTables* (shown in table 9), creates and links new *object* symbols to tables that are being looked at. It is used to denote that an obstacle has been successfully modelled as a table.

Rule 10 is named *modelMugInTable* (shown in table 10). It modifies the type of *object* symbols to *mug* symbols. It is used to denote that an object has been successfully modelled as a mug.

7.3 Agents and behaviours

AGM rules do not model or interact with the environment by themselves –they are only used to plan actions and verify model modifications. Such actions are implemented in agents, as described in section 3, which are also in charge of updating the model correspondingly. For this experiment, we developed the following agents:

- **yaw:** In charge of modelling and tracking the yaw angle of the camera of the robot. Implements rule 1.
- **saccade:** In charge of performing camera saccades and platform movements. Implements rules 2, 3 and 7.

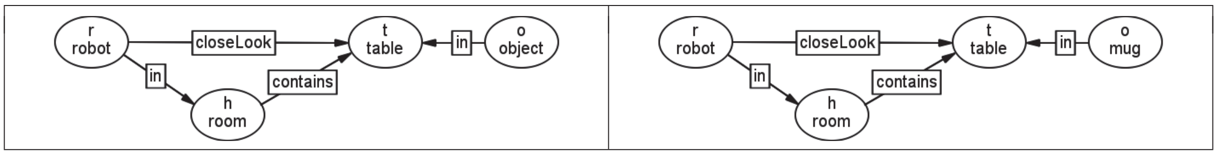


Table 10. Rule 10 is used to classify objects in tables as mugs. Rule name: "**modelMugInTable**".

- **distance:** In charge of modelling and maintaining floor and wall distances. Implements rules 4 and 5.
- **unknown:** Detects and introduces in the world representation obstacles (surfaces that were not expected according to the previous world model). It can also provide models for obstacles and objects (tables and mugs). It implements rules 6, 8, 9 and 10 and updates the attributes of the *obstacle*, *table*, *object* and *mug* symbols.

7.4 Implementation and software components

The whole system was implemented using COP (see section 6) and the RoboComp framework [37]. The executive –which is domain-independent– and each of the agents were implemented as separate components. Additional components for hardware access, the processing of the acquired point cloud, and the execution of the platform and camera-coordinated saccades, were also used.

7.5 Experiment execution

The experiment described in the previous section was tested in a real robot in the environment shown in figure 11. It is a room with two tables and several obstacles: three chairs, a coat stand, a radiator and a litter bin. The goal provided to the executive is shown in figure 10. As with any other goal pattern, it only contains the symbols required to specify the goal: "the robot must model the room in which it is located and find an obstacle".

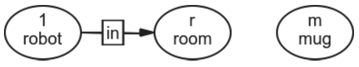


Figure 10. The goal provided to the executive for the experiment

The plan returned by the planner to achieve the robot's goal consists of the sequence executions of the following rules: 1) modelYaw, 2) lookFirstWall, 3) modelWall, 4) lookNextWall, 5) modelWall, 6) lookNextWall, 7) modelWall, 8) lookNextWall, 9) modelLastWall, 10) findObstacle, 11) approachObstacle, 12) modelTable, 13) findObjectInTable and 14) modelMugInTable. The plans in AGM are monitored and updated with every model-change event in case something does not go as expected. However, to avoid using unnecessary space and to improve readability, we describe the execution of an experiment in which everything goes as planned. Table 11 shows the events that occur in the execution of the experiment according to the plan.

source of the event	**event description**
executive	The experiment begins. The executive analyses the mission and uses the planner to get the first action of the plan with the lowest cost that would get the robot to the target state. As a result, the *yaw* agent is activated.
yaw	The agent *yaw* models the yaw of the robot with respect to an arbitrary wall of the room and proposes to update the current model. After the modification is verified, the executive broadcasts the new model and removes the first action of the plan (Rule 1).
saccade	The agent *saccade* makes the robot look towards the closest wall (Rule 2). Again, as in the following events, the agent proposes a modification and, after the model is verified and broadcast, the executive updates the plan, removing the first action.
distance	The agent *distance* models the distance to the wall (Rule 4).
saccade	The agent *saccade* makes the robot look towards the next wall (Rule 3). These two last events are repeated three times.
distance	The agent *distance* models the last wall (Rule 5).
unknown	The agent *unknown* detects an obstacle (Rule 6).
saccade	The agent *saccade* makes the robot closer to the obstacle (which is actually a table) and triggers Rule 7.
table	The agent *table* models the obstacle as a table (Rule 8).
unknown	The agent *unknown* detects an object on the table (Rule 9).
mug	The agent *mug* models the object as a mug (Rule 10). The robot accomplishes the mission.

Table 11. Events and actions during the experiment

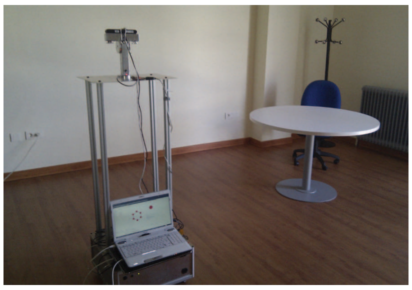


Figure 11. Picture of the environment in which the system was tested

The software developed for the experiment benefited from using the architecture because AGM enforces implementing modules as loosely coupled components: the executive, the planner and, more importantly, the agents. This made it easier to include new agents and grammar rules, and reduced undesirable inter-module interactions (*e.g.*, deadlocks and shared-memory side-effects). The fact that the executive works with high-level descriptions helps in detecting and reducing errors and diminishes the time spent designing and integrating the grammar.

The mean execution time was 94.4 seconds with a standard deviation of 4.3. The memory overhead derived from the use of the component-oriented framework was less than 10 Mb per component. Given that we used a total of eight components, the overall memory overhead was a total of 80 Mb, which is negligible for any modern computer. Additionally, we benefit from the advantages derived from COP (see section 6). The maximum planning time measured was 1.45 s using the FastDownward planner on an i5 laptop [38].

8. Conclusions

Perception is still the main limitation of domestic robots [1]. The paper presented the AGM architecture and the features that it provides to improve robot perception. It makes use of AGGL, a high-level visual language which enables the easy creation of domain definitions and the obtaining of plans that explicitly express how can robots actively find and classify objects. Instead of an active process which is automatically planned, and to the knowledge of the authors, all previous architectures approach perception whether as a passive process or as part of an already-given task plan. Moreover, AGM verifies that the model held by the robot is grammatically sound by accepting only those modifications that are valid according the grammar of the world, a feature which any other known architecture lacks.

The AGM architecture has a distributed nature and was designed with reusability and scalability in mind: the executive is domain-independent, and the same agents can be reused for different applications with little or no modification.

To demonstrate how the architecture and the language AGGL can be used, the paper presented a moderately complex experiment in which a robot uses these technologies to autonomously model the room in which it is located and find and model a mug on a table. Robots using AGM can be assigned more complex goals dealing with additional types of objects and additional rules dealing with them, see http://grammarsandrobots.org for additional examples.

9. Acknowledgements

This paper has been partially supported by the Spanish Ministerio de Economía y Competitividad TIN2012-38079, Junta de Extremadura fund GR15120 "Ayudas a Grupos" and FEDER funds.

10. References

[1] S. Srinivasa et al. Herb: a home exploring robotic butler. *Autonomous Robots*, 28(1):5–20, 2010.

[2] M. Ross Quillian. The teachable language comprehender: A simulation program and theory of language. *Communications of the ACM*, 12(8):459–476, 1969.

[3] A. Müller, A. Kirsch, and M. Beetz. Transformational planning for everyday activity. In *ICAPS*, pages 248–255, 2007.

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

[5] S. Edelkamp. Limits and possibilities of PDDL for model checking software. In *Int. Conf. on Automated Planning and Scheduling*, pages 53–63, 2003.

[6] K. Erol, J. Hendler, and D.S. Nau. HTN planning: Complexity and expressivity. In *AAAI*, volume 94, pages 1123–1128, 1994.

[7] J. Bauer and R. Wilhelm. Abstract interpretation of graph grammars. In *Proceedings of Simulation and Verification of Dynamic Systems Seminar*, 2006.

[8] I. Bouassida, C. Chassot, and M. Jmaiel. Graph grammar-based transformation for context-aware architectures supporting group communication. *Nouvelles Technologies de l'Information*, L(19):29–42, 2010.

[9] B. Smith, A. Howard, J.M. McNew, J. Wang, and M. Egerstedt. Multi-robot deployment and coordination with embedded graph grammars. *Autonomous Robots*, 26(1):79–98, 2009.

[10] S.C. Zhu, R. Zhang, and Z. Tu. Integrating bottom-up/top-down for object recognition by data driven markov chain monte carlo. In *Computer Vision and Pattern Recognition Conf.*, volume 1, pages 738–745. IEEE, 2000.

[11] Z. Tu, X. Chen, A.L. Yuille, and S.C. Zhu. Image parsing: Unifying segmentation, detection, and recognition. *Int. Journal of Computer Vision*, 63(2): 113–140, 2005.

[12] F. Han and S.C. Zhu. Bottom-up/top-down image parsing with attribute grammar. *Transactions on Pattern Analysis and Machine Intelligence*, 31(1):59–73, 2009.

[13] L. Lin, T. Wu, J. Porway, and Z. Xu. A stochastic graph grammar for compositional object representation and recognition. *Pattern Recognition*, 42(7): 1297–1307, 2009.

[14] J. Hasemann. A robot control architecture based on graph grammars and fuzzy logic. In *Proc. of Int. Conf. on Intelligent Robots and Systems*, volume 3, pages 2123–2130. IEEE, 1994.

[15] J. Siskind, J. Sherman, I. Pollak, M. Harper, and C. Bouman. Spatial random tree grammars for model-

ing hierarchal structure in images with regions of arbitrary shape. *Transactions on Pattern Analysis and Machine Intelligence*, 29(9):1504–1519, 2007.

[16] A. Gupta, A. Efros, and M. Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *European Conference on Computer Vision*, pages 482–496. Springer, 2010.

[17] Z. Tu and S.C. Zhu. Image segmentation by data-driven markov chain monte carlo. *Transactions on Pattern Analysis and Machine Intelligence*, 24(5):657–673, 2002.

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

[19] R. Knight, F. Fisher, T. Estlin, B. Engelhardt, and S. Chien. Balancing deliberation and reaction, planning and execution for space robotic applications. In *Int. Conf. on Intelligent Robots and Systems*, volume 4, pages 2131–2139. IEEE, 2001.

[20] R. Brooks. A robust layered control system for a mobile robot. *Journal of Robotics and Automation*, 2(1): 14–23, 1986.

[21] J.E. Laird, A. Newell, and P.S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial intelligence*, 33(1):1–64, 1987.

[22] R.C. Arkin and T. Balch. AuRA: Principles and practice in review. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3):175–189, 1997.

[23] J.A. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. In *AAAI*, volume 88, pages 21–26, 1988.

[24] N. Yamasaki and Y. Anzai. Active interface for human-robot interaction. In *Proc. of Int. Conf. on Robotics and Automation*, volume 3, pages 3103–3109. IEEE, 1995.

[25] K.P. Sycara. Multiagent systems. *AI magazine*, 19(2): 79, 1998.

[26] L. Zhang, X. Huang, Y. Kim, and D. Manocha. D-plan: Efficient collision-free path computation for part removal and disassembly. *Journal of Computer-Aided Design and Applications*, 5(6):774–786, 2008.

[27] S.A. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Using iterative repair to improve the responsiveness of planning and scheduling. In *AIPS*, pages 300–307, 2000.

[28] K. Haigh and M. Veloso. Interleaving planning and robot execution for asynchronous user requests. In *Autonomous agents*, pages 79–95. Springer, 1998.

[29] V. Alcázar, C. Guzmán, D. Prior, D. Borrajo, and L. Castillo. PELEA: Planning, learning and execution architecture. In *PlanSIG'10*, pages 17–24, 2010.

[30] M. Ai-Chang et al. Mapgen: mixed-initiative planning and scheduling for the mars exploration rover mission. *Intelligent Systems*, 19(1):8–12, 2004.

[31] P. Laborie and M. Ghallab. IxTeT: an integrated approach for plan generation and scheduling. In *Proc. of Symp. on Emerging Technologies and Factory Automation*, volume 1, pages 485–495. IEEE, 1995.

[32] R. Heckel. Graph transformation in a nutshell. *Electronic notes in theoretical computer sci.*, 148(1):187–198, 2006.

[33] H. Svensson, A.F. Morse, and T. Ziemke. Neural pathways of embodied simulation. In *Anticipatory Behavior in Adaptive Learning Systems*, pages 95–114. Springer, 2009.

[34] R. Cintas et al. Robust Behavior and Perception using Hierarchical State Machines: a Pallet Manipulation Experiment. *Journal of Physical Agents*, 5(1): 35–44, 2011.

[35] J. Bohren et al. Towards autonomous robotic butlers: Lessons learned with the PR2. In *Int. Conf. on Robotics and Automation*, pages 5568–5575. IEEE, 2011.

[36] M. Quigley et al. ROS: an open-source Robot Operating System. In *Proc. of ICRA Workshop on Open Source Software*, 2009.

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

[38] Malte Helmert. The fast downward planning system. Journal of Artificial Intelligence Research, 26:191–246, 2006.