Available online at www.sciencedirect.com

ScienceDirect

Cognitive Systems Research 59 (2020) 179–197

Cognitive Systems RESEARCH

www.elsevier.com/locate/cogsys

The TROCA Project: An autonomous transportation robot controlled by a cognitive architecture

Ricardo Gudwin [a,*], Eric Rohmer [a], André Paraense [a], Eduardo Fróes [a], Wandemberg Gibaut [a], Ian Oliveira [a], Sender Rocha [a], Klaus Raizer [b], Aneta Vulgarakis Feljan [c]

[a] *University of Campinas (UNICAMP), Campinas, SP, Brazil*
[b] *Ericsson Research Brazil, Ericsson Telecomunicações S.A., Indaiatuba, Brazil*
[c] *Ericsson Research, Ericsson AB, Stockholm, Sweden*

Received 26 June 2019; accepted 12 September 2019
Available online 17 September 2019

Abstract

Autonomous mobile robots emerged as an important kind of transportation system in warehouses and factories. In this work, we present the use of MECA cognitive architecture in the development of an artificial mind for an autonomous robot responsible for multiple tasks, including transportation of packages along a factory floor, environment exploration, warehouse inventory, its internal energy management, self-monitoring and dealing with human operators and other robots. The present text provides a detailed specification for the architecture and its software implementation. Future work will present the simulation results under different configurations, together with a detailed analysis of the architecture performance and its generalization for autonomous robot control.
© 2019 Elsevier B.V. All rights reserved.

Keywords: Cognitive architecture; Transportation robot; Dual-process theory; Dynamic subsumption; MECA

1. Introduction

According to Adinandra, Caarls, Kostić, Verriet, and Nijmeijer (2012), autonomous mobile robots (AMR) have emerged as a means of transportation system in warehouses. AMRs, sometimes also addressed as AGVs (autonomous guided vehicles), provide a scalable solution, providing transportation in a reliable and flexible way. A large collection of AMRs can be responsible for the transportation of different kinds of goods within a warehouse, with robustness and flexibility.

The full transportation system might use a centralized or decentralized solution, providing lower or higher autonomy in the operation of such AMRs. Modern transportation systems with AMRs are slowly giving more autonomy to them. As technology evolves, systems are moving increasingly towards autonomy, perceiving, deciding, learning, etc. often without human engagement. According to Lacher, Grabowski, and Cook (2014), the definition of autonomy or an autonomous system is not a simple matter. Autonomous systems decide for themselves what to do and when to do it.

The field of Cognitive Robotics emerged as a confluence of Artificial Intelligence and Robotics (Clark & Grush,

* Corresponding author.
E-mail addresses: gudwin@dca.fee.unicamp.br (R. Gudwin), eric@dca.fee.unicamp.br (E. Rohmer), paraense@dca.fee.unicamp.br (A. Paraense), senderrocha@yahoo.com.br (S. Rocha), klaus.raizer@ericsson.com (K. Raizer), aneta.vulgarakis@ericsson.com (A. Vulgarakis Feljan).

https://doi.org/10.1016/j.cogsys.2019.09.011
1389-0417/© 2019 Elsevier B.V. All rights reserved.

1999; Christaller, 1999) as the study of the knowledge representation and reasoning problems faced by an autonomous robot (or agent) in a dynamic and incompletely known world (Levesque & Lakemeyer, 2008). According to Moreno, Umerez, and Ibañez (1997), cognition is a major enabler for autonomy in biological systems, which means that giving cognitive abilities to robots might improve their capacity for autonomy.

Cognitive Architectures (CAs) have been employed in many different kinds of applications, from the control of robots to decision-making processes in intelligent agents. Cognitive Architectures are general-purpose control systems' architectures inspired by scientific theories developed to explain cognition in animals and humans (Raizer, Paraense, & Gudwin, 2012).

Usually, a cognitive architecture is decomposed based on its cognitive capabilities, like perception, attention, memory, reasoning, learning, behavior generation, etc. Cognitive Architectures are, at the same time, theoretical models for how many different cognitive processes interact with each other in order to sense, reason and act, and also software frameworks which can be reused through different applications.

In 2010, Samsonovich (2010) conducted a broad study resulting in a comparative table, presenting a comprehensive review of the most important implemented CAs in the literature. More recently, Kotseruba, Gonzalez, and Tsotsos (2016) performed a detailed analysis of how this field of research developed in the last 40 years. Our research group has recently contributed to the field with the proposition of MECA – the Multipurpose Enhanced Cognitive Architecture (Gudwin et al., 2017; Gudwin et al., 2018).

According to Kurup and Lebiere (2012), robotics research has traditionally been focused on finding optimal solutions via the use of specialized techniques that involve carefully crafted representations and processes. In contrast, AI and Cognitive Science are concerned with problems that are, in many ways, diametrically opposed to robotics, namely, open-ended tasks that span longer intervals of time in discrete domains, are knowledge-intensive and make assumptions about the existence of modules and tools that simplify interaction with the environment. Examples of such tasks include high-level planning and scheduling problems, language understanding, instruction following, diagnosis, and domain-independent execution monitoring and recovery. Cognitive architectures capture the underlying commonality between different intelligent agents and provide a framework from which intelligent behavior arises. The architectural approach emphasizes the role of memory in the cognitive process, i.e., cognition is centered on the notion of rapidly identifying previous experiences that relate to the current situation, appropriately modifying previous responses and applying this modified response to the current situation. There are many reports in the literature promoting the use of cognitive architectures with robotics (Benjamin, Lyons, & Lonsdale, 2004; Burghart et al., 2005; Avery, Kelley, & Davani, 2006; Kelley, 2006; Ziemke & Lowe, 2009; Lemaignan, Ros, Mösenlechner, Alami, & Beetz, 2010; Trafton et al., 2013). Particularly, the use of Cognitive Architectures can be quite powerful in providing autonomy to robotic systems (Thórisson & Helgasson, 2012).

In this paper, we propose the use of MECA in order to build a Transportation Robotic System for a simulated factory. A simulated environment was constructed with the aid of the V-REP robotics simulator (Rohmer, Singh, & Freese, 2013), and MECA was used to build TROCA, an intelligent agent controlling a transportation robot within the factory.

2. MECA – The multipurpose enhanced cognitive architecture

The development of MECA was a first attempt to compose a large generic-purpose cognitive architecture with many features inspired in popular ones like SOAR (Laird, 2012), Clarion (Sun, 2003) and LIDA (Franklin, Madl, D'mello, & Snaider, 2014), using our Cognitive Systems Toolkit (CST) (Paraense, Raizer, de Paula, Rohmer, & Gudwin, 2016) as a core.

During the design of MECA, we tried to integrate many lessons acquired from our study of other cognitive architectures. Among these lessons, the use of codelets (just like in LIDA) as the building blocks of processing, and a mechanism inspired on Global Workspace Theory (Baars, 1988) to implement a machine consciousness cognitive capability (similarly but not exactly equals to the one in LIDA), the requirement to have both explicit and implicit knowledge representations (just like in Clarion), considering rule-based processing (just like in SOAR) as an explicit processing modality, together with a dynamic subsumption-like processing (Brooks, 1986; Nakashima & Noda, 1998; Hamadi, Jabbour, & Saïs, 2010; Heckel & Youngblood, 2010) as an implicit processing modality, with the potential to include also neural networks like Hierarchical Temporal Memory (HTM) (George & Hawkins, 2009) within some of its modules. We got inspiration from hybrid cognitive architectures, like SAL (Synthesis of ACT-R and Leabra) (Jilk, Lebiere, O'Reilly, & Anderson, 2008), which combined both a rule-based architecture (ACT-R) with a neural network one (Leabra) in order to compose a more powerful architecture and mostly a strong inspiration on dual process theory, which recently is being explored under the context of cognitive architectures (Faghihi, Estey, McCall, & Franklin, 2015; Lieto, Chella, & Frixione, 2017; Lieto, Radicioni, & Rho, 2017; Augello et al., 2016). We also relied on important theories regarding knowledge representation, including Grounded Cognition (Barsalou, 2010), Conceptual Spaces (Gärdenfors, 2014) and Computational Semiotics (Gudwin, 2015), extending some recent work (Lieto et al., 2017; Lieto et al., 2017) indicating the relevance of extended kinds of representation beyond the traditional symbolic rule-based processing and multi-layered backpropagation neural networks.

According to Dual Process Theory (Osman, 2004), the human mind can be described by the interaction of two different systems, named System 1 and System 2, which assume two functionally distinct roles that integrate with each other, in order to account for the different facets of the mind phenomena. The exact characteristics of System 1 and System 2 varies depending on the theory proposers.

System 1 is generally described as a form of universal cognition shared between humans and animals. It is not actually just a single system, but a a set of sub-systems operating with some level of autonomy. System 1 includes instinctive behaviors that might be innately programmed and also automatic learned behaviors evolved during the system interaction with its environment. System 1 processes are rapid, parallel and automatic in nature: only their final product is posted in consciousness (Evans, 2003).

System 2 is believed to have evolved much more recently and is considered by many to be uniquely human. System 2 thinking is slow and sequential in nature, and makes use of the central working memory system, intensively studied in psychology. Despite its limited capacity and slower speed of operation, System 2 permits abstract hypothetical thinking that cannot be achieved by System 1, as e.g. decision-making using past experiences to abstract new behaviors and the construction of mental models or simulations of future possibilities, in order to predict future events and behave accordingly to reach desirable situations or prescribed goals (Evans, 2003).

Despite their intrinsic autonomy, System 1 and System 2 interact with each other in order to build the overall system behavior. System 1 implements a kind of fast, automatic reactive behavior which provides a default response to system input, aligned with a possible set of general system goals. System 2 has a kind of inhibitory role in suppressing this default response, emphasizing specific time-based goals, which are characteristic of exceptional situations, generating as a result a complex and refined interleaved overall behavior.

To implement System 1, we designed a Dynamic Subsumption Architecture, implemented on top of CST. The inputs to this Dynamic Subsumption Architecture might come directly from Sensory Memory, but usually there is some kind of Perception processing in between. The role of Perception is to generate more elaborate Percepts, abstractions of sensory data, which are then used as input to the Behavioral Codelets. These percepts can also be tracked by Attention Codelets in order to detect special situations and send information upstream to System 2. These attention codelets are responsible for generating the Current Perception at the Working Memory, where a selected subset of the Perception Memory is made available for System 2 subsystems in a representation suitable to be processed within System 2. Among the behavioral codelets, there is a special sub-set (Motivational Behavioral Codelets) comprising the System 1 Motivational Subsystem, which is responsible for implementing a kind of instinct mechanism in the architecture. This Motivational Subsystem also includes some sort of emotional processing.

According to Dual Process Theory, System 2 is responsible for the slow conscious process of deliberative reasoning. It is mainly a sequential rule-based process, operating on symbols, and considering not just the present, like in System 1, but also the past and the future. This is the place where imagination and planning occurs. This is also the place where the many unconscious perceptions performed at System 1 enter into a process of competition to integrate the agent's present experience, where the most important percepts are payed attention to and other less relevant are discarded. This leads to the formation of the conscious perception which is usually called experience by many philosophers of mind, and which is integrated into episodes, and then stored in an episodic memory to be recovered later for many purposes.

System 2 Specification in MECA includes the definition of an Episodic Subsystem, responsible for higher-level perception with the tracking of time along Perceptual Memory. With the aid of Attention codelets it discover and detects the formation of episodes, and performs the storage and recovering of these episodes in the Episodic Memory. It also includes a Planning Subsystem, responsible for simulating the future and making plans of action in order to reach possible Goals. The Planning Subsystem is also responsible for the process of Imagination, which is used as an aid for testing possible courses of action and evaluating the best action to take. MECA's implementation of System 2 also includes a High-Level Motivational Subsystem, responsible for generating Goals for the Planning Subsystem; an Expectation Subsystem which tries to foresee the short-term future and learn from the possible inconsistencies; and a Consciousness Subsystem, responsible for filtering the information available for the Planning Subsystem.

The Planning Subsystem relies on SOAR, a rule-based Cognitive Architecture developed by John Laird at the University of Michigan, in USA (Laird, 2012). The decision to use SOAR for rule-based processing in order to perform imagination and planning was aligned with recent tendencies in the literature. The cognitive architecture SAL (Synthesis of ACT-R and Leabra) performs a similar exercise by binding two different cognitive architectures (ACT-R and Leabra), where ACT-R is a rule-based cognitive architecture, similar to SOAR, and Leabra is a Neural-network based cognitive architecture. One motivation for choosing SOAR instead of ACT-R was that there is an open source implementation of SOAR in Java (ACT-R is in LISP), which is compatible with our CST toolkit. Another motivation is the fact that SOAR is a mature technology, with good documentation and a solid repertoire of use cases in the community, it therefore appears to be the right choice. Of course, other rule-based systems and cognitive architectures, e.g. ACT-R, could also be used.

The problem of building a control system for an autonomous transportation robot is a specially interesting one for exploring MECA's capability. Due to its intrinsic complexity, the scenario provides good opportunities to explore MECA's cognitive abilities, such as perception, planning, opportunistic behavior, etc.

3. TROCA – A transportation robot controlled by a cognitive architecture

In order to explore the potential use of a Cognitive Architecture in robotics, we used the V-REP simulation framework (Rohmer et al., 2013) to simulate an industrial environment, where a set of transportation robots are required to move packages around in an autonomous way. The overall experiment concerns an industrial scenario, holding both processing cells, a warehouse, and robots for providing the transportation of packages from cell to cell, or from and to the warehouse. The factory floor is split into different regions: Processing Cells, Warehouses and Open spaces.

A *processing cell* is a region of the factory floor where robots are not allowed to enter (the same happens to the *warehouse*[1]), and some sort of processing is realized. This processing might be of any type, including receiving materials from outside of the factory, handling materials and storing them on packages, assembling or mounting pieces or devices, performing maintenance, testing or service on parts, applying some industrial process, like turning, milling, drilling, molding, painting or other machine operations and boxing/shipping packages to outside the factory. The common role of a processing cell is that this service is performed by either humans or robots, and there will be particular spots where materials are entering or leaving the processing cell. Transportation robots are allowed to run only on *open space*. Robots might be able, though, to pick packages from processing cells and from the warehouse.

Each process cell might have a *pickup/delivery spot*, where packages might be put, in order for transportation robots to pick them and transport them to the appropriate place. The same spot might be used for either pickup, delivery, or both. Pick-up/delivery spots are marked with an AR-Tag for the sake of identification. Also, every package is marked with an AR-Tag. AR-Tags are *Augmented Reality Tags*, a special kind of QR-code which uniquely identifies either a pickup-delivery spot or a package. Besides the normal kind of information available in a QR-code, the AR Tags design is used to extract a positional reference frame using a computer vision algorithm. This reference frame serves to define where to handle a package, or where a package can be placed or delivered. The use of AR-Tags makes it easier to develop a perception system for the robot to pick and deliver packages.

[1] The *warehouse* includes only the space where goods are stored. Corridors next to storage shelves are not considered as being a part of it.

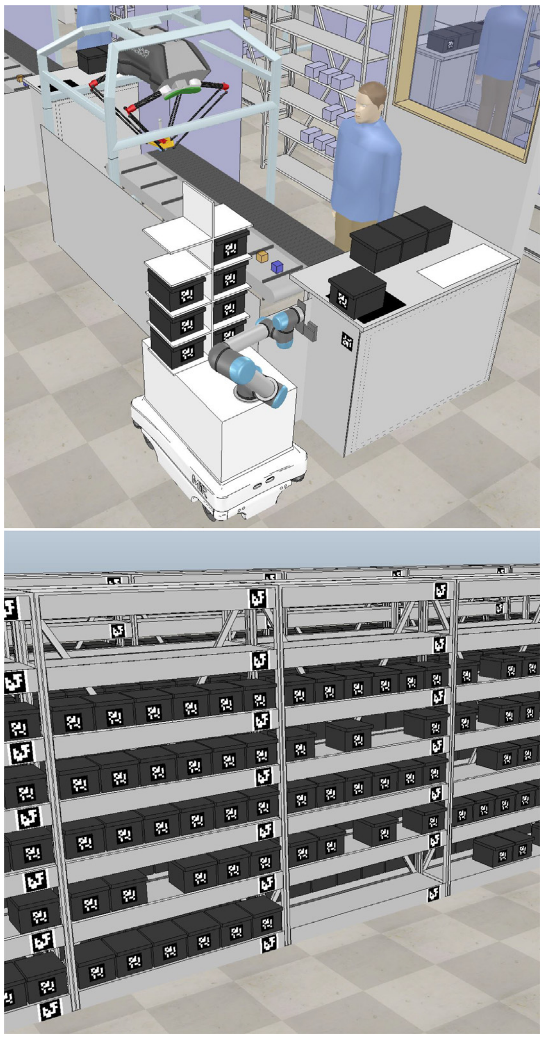
Fig. 1. Example of processing cell and warehouse.

Besides processing cells, packages can be stored in the *warehouse*. From our technical perspective, a *warehouse* is a place inside the factory whose sole purpose is to store items. Without loss of generality, we represent the *warehouse* as a sequence of shelves, where packages can be delivered and picked-up, either by robots or by human beings. These shelves are identified by AR-Tags, in a similar way to those on the picking-delivery spots. The robots might pick and deliver packages from these shelves, in the same way they do in the processing cells. One additional feature expected from the robotic systems working nearby the warehouse shelves is to manage the inventory of items currently stored there. Whenever passing through the warehouse shelves, the robots should check if the packages expected to be stored in the shelves are really there, as they may be mistakenly placed in a wrong shelf, or taken by a human operator. Examples of a Processing Cell and the Warehouse are illustrated in Fig. 1.

Our configuration for the transportation robot includes a MiR100 base (from Mobile Industrial Robotics[2]), with

[2] https://www.mobile-industrial-robots.com/en/products/mir100/.

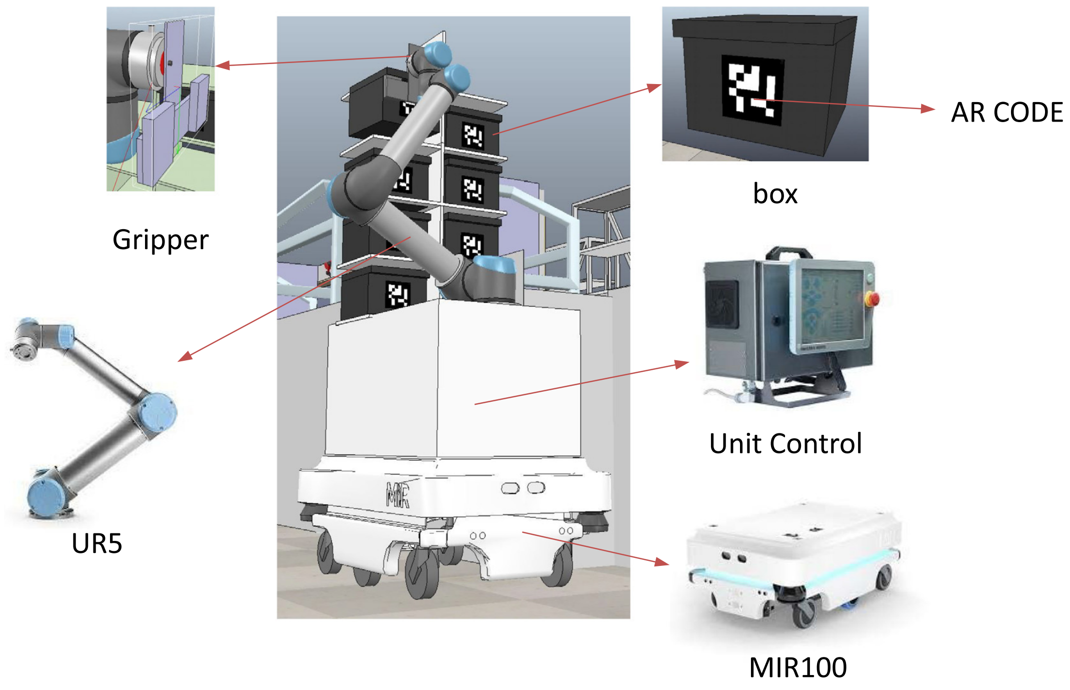

Fig. 2. Detail of the robot.

an upper UR5 mechanical arm (from Universal Robots[3]), and a set of slots where delivery items can be stored. Other commercially available solutions could also be used to build a similar robot configuration. It is, therefore, capable of picking up many packages while also delivering some of them, making it possible to explore many routing techniques and thus reach optimal performance through flexible and adaptive behavior. Fig. 2 provides a better view of the transportation robot and its parts.

Completing the full simulation architecture, the *General Manager* is a software agent, also controlled by a cognitive architecture, which does not have a physical instance, like the transportation agents, but is a software-only kind of agent. The General Manager has full access to a database, where there is a common registry of all known packages in the factory, and their contents, all known pickup-delivery spots in all processing cells, all known storage spots in all warehouse locations. The General Manager is reachable by all transportation robots by means of network sockets, and they can exchange information this way. The General Manager is also reachable by human employees by means of smartphone applications, where they are able to grab information about the common registry, introduce new information in this registry, as new packages are introduced in the system, and delete information in the registry, as packages are dismantled or re-organized within processing cells. As soon as a package is elaborated in a processing cell, a human operator is supposed to contact the General Manager, by means of his/her smartphone application and point out what is the destination of the package, which might be a storage spot in a warehouse or a pickup-delivery spot in another processing cell. In the same way, as soon as a package is collected in a processing cell, a human employee is supposed to acknowledge the package input to the processing cell, registering that information in the common register with the General Manager.

Transportation agents are able to communicate with the General Manager, and with each other, always using network sockets for this communication. Human operators are also able to communicate both to the General Manager and to the Transportation Agents, using their smartphone application. The General Manager is also able to start a communication to either Transportation Agents or to human employees. Human employees are uniquely identified by their smartphone application. Each employee has a unique pair user/password, which is used to log-in into the smartphone application. As soon as it logs intp the application, the General Manager becomes aware that they are available within the factory. If the General Manager, for some reason, must contact a given employee, it might generate a notification in the smartphone application where the employee is logged. The smartphone application is constantly “pinging” the General Manager, to indicate they are available in factory space. If, for some reason, this “pinging” does not succeed (as e.g. the employee goes out of the factory, or runs out of battery), the General Manager assumes the employee is not available, and awaits for a new pinging event to assume that the employee is back to service. The contact from the General Manager might be to give orders or instructions or to ask for advice in some decision-making situations it cannot solve by itself.

Transportation agents might need to communicate with other transportation agents. For example, we might

[3] https://www.universal-robots.com/products/ur5-robot/.

conceive a situation where one transportation agent is blocking the way for another transportation agent to pass. In this case, the one which is intending to pass should contact the other and let it know that they are blocking the way. In this case, some sort of negotiation should go on in order to solve the situation.

The following general tasks are specified to be performed by our transportation agent:

- Package Transportation
  - Pick-up of packages in processing cells and its transportation to their final destination, which can be another pick-up-delivery slot or a warehouse shelf
- Environment Exploration
  - Generation/update of maps, including human beings, obstacles, pick-up/delivery slots and packages (metric, topological and semantic SLAM).
- Warehouse Inventory
  - Identification of slots in warehouse shelves
  - Situation/occupation of warehouse shelves
- Energy Management
  - Verify energy (battery) level, recharging when it is necessary
- Dealing with Human Operators
  - Humans cross the same environment as the robots, and pick packages from the warehouse – safety considerations should be respected.
  - Communication with human operators in order to solve problems (using smartphone application).
  - Visual robot-human interface in order to express information for humans without a smartphone interface.
- Dealing with other robots
  - Robots should be able to communicate with each other in order to solve deadlocks, e.g. dispute for passageway, and perform local coordination of activities.
- Self-Monitoring and Benchmarking
  - Different means of benchmarking (min/ average/max time to attend requests, distance-to-identification of AR-Tags, etc).

4. TROCA agent cognitive model

We designed a cognitive agent to control the transportation robot, assigning to it multiple responsibilities, which might be attended in an autonomous way. This agent was constructed using the MECA Cognitive Architecture.

Differently from other cognitive architectures like SOAR or LIDA, to be effectively used in an application, MECA requires a process of instantiation, i.e. the MECA user might need to define and implement a set of different codelets and memory objects in order to use the cognitive architecture. So, in order to use MECA to control the transportation robot, it is necessary, first, to specify the different kinds of codelets prescribed for the task of controlling the robot. A customization of MECA, designed to control our agent is shown in Fig. 3.

Following the directives from dual process theories, the main architecture is split in two major sub-systems: *System 1*, in the bottom and *System 2* in the top of the figure. *System 1* is designed to accommodate the major automatic behaviors, tracking a chain of processes starting in sensors and delivering commands to actuators. The main sensors in TROCA are ROS (Robot Operating System) topics (Joseph, 2015) being written by the Transportation Robot into the ROS system and sockets from the Operational System. Also, the actuators are ROS topics being written by TROCA, and ROS services, as we describe in Section 4.1.3 and also sockets from the Operational Systems. Sockets are used for network communication among the many TROCA agents and also to and from the General Manager, also an agent, responsible for providing general information about transportation requests and other information. This means that for TROCA, it doesn't matter if the Transportation Robot is being simulated using V-REP (as we will be doing in our experiments) or is a real robot using ROS (Joseph, 2015).

System 2 was designed to generate *plans*, multi-step sequences of actions allowing the decrease of drives in the motivational sub-system after their conclusion. The idea is that *System 2* and *System 1* should work in an integrated way. In some situations, described in Section 4.1.6, *System 1* requests that a new plan for obtaining a certain goal should be developed. In this case, this request is delivered to *System 2*, which should elaborate the plan and send it back to *System 1* to be used in its behaviors. The next two sections provide an overall description of *System 1* and *System 2*.

4.1. The System 1 specification

System 1 can be seen at the bottom part of Fig. 3. On the left, we can see in dark green the sensory codelets feeding the Sensory Memory. This Sensory Memory is then integrated into a single *Percept* by the Perceptual Codelet. This Percept is then distributed to the many Motivational Codelets, Behavioral Codelets, the Activity Tracking Codelet (on the top right of the diagram), and up to *System 2*.

Next, in the diagram, we can find the many Motivational Codelets, in light green, responsible for modeling the many *needs* associated to the MECA motivational sub-system. For each Motivational Codelet, there is a corresponding *drive* in the *Drives Memory*. According to MECA's principles, *needs* are the main source of motivation for our agent. In MECA's motivational system, *drives* measure the degree of dissatisfaction of a need, which are computed by a corresponding Motivational Codelet. The

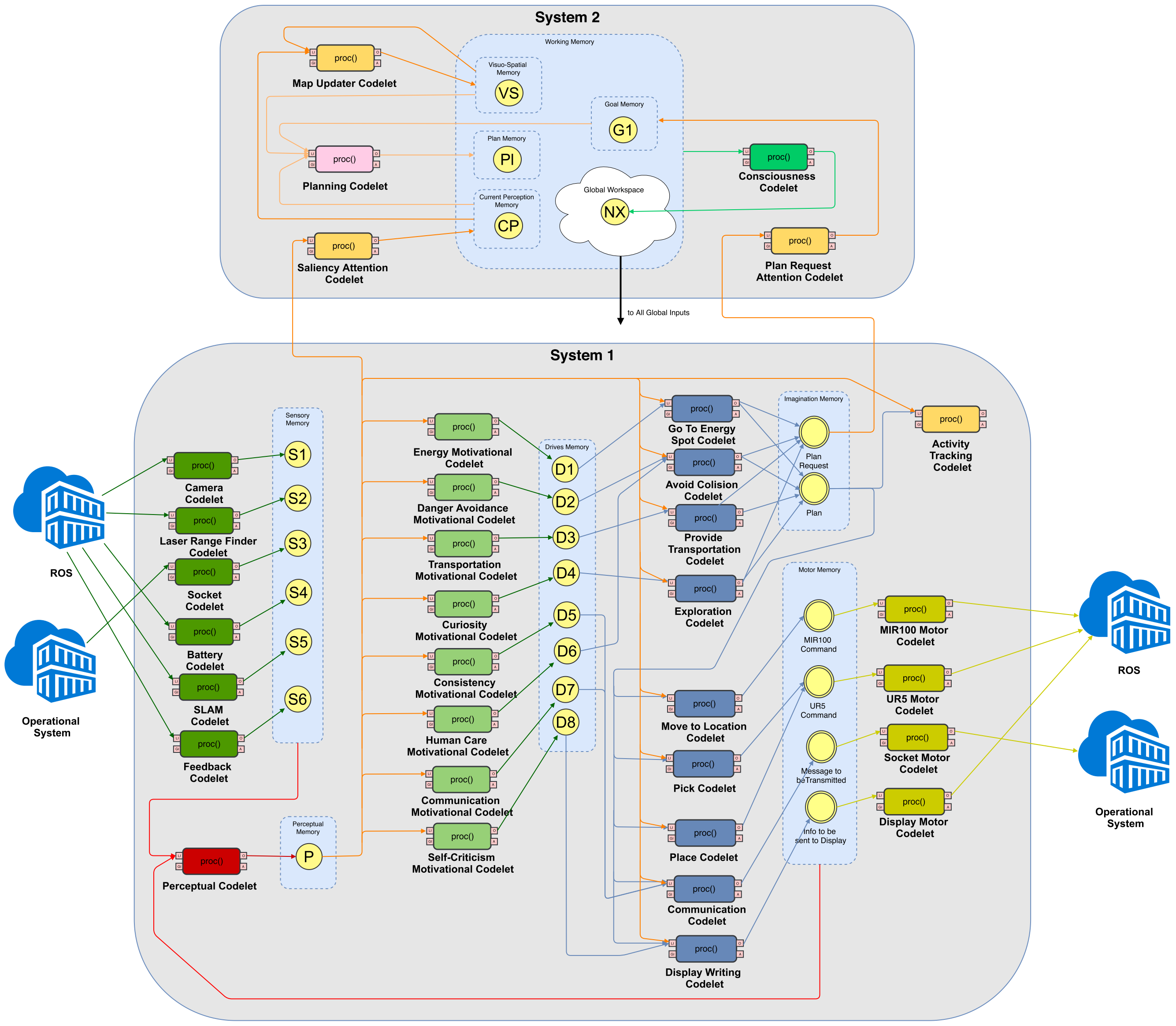

Fig. 3. Cognitive model of the TROCA Agent.

many *needs* attributed to our Cognitive Agent are related to the many assignments which are expected to be accomplished by the Cognitive Agent.

Next in the diagram we can see the many Behavioral Codelets, in dark blue, which are responsible for behavior generation in the TROCA Cognitive Agent. The Behavioral Codelets are split into two sub-groups, a smaller one on the top, with 4 codelets, and a larger one on the bottom, with 5 codelets. This division of Behavioral Codelets in two groups, is one of the innovations developed during the development of TROCA, which required an enhancement of MECA's specification. The original MECA specification (Gudwin, 2016; Gudwin et al., 2017) was extended to allow the execution of sequences of actions by *System 1*. Also, the Behavioral Codelets are feeding two different kinds of memory: *Imagination Memory* and *Motor Memory*. In the Imagination Memory, the Multi-step Behavior Codelets generate plans and plan requests. In the *Motor Memory*, the Action Behavior Codelets generate actuation signals, which will be later collected from the Motor Codelets to transform these signals into actions.

Finally, on the right, we can see the *Motor Codelets*, which are responsible for collecting actuation signals from *Motor Memory* and transforming these into actions. In our case, we do this by using ROS topics, ROS services and writing to network sockets.

In the next sections, we describe the many codelets necessary for fulfilling these specifications.

4.1.1. The Sensory Codelets

The Sensory Codelets are those responsible for collecting data from environment, in order to start the causal chain which might result in the fulfilling of the system's goals. In our case, most of these systems will be collected from ROS, because even though we are running a simulated experiment, we are connecting the simulated environment with ROS topics and services. This means that, in the future, we might replace the simulated environment with real transportation robots and we expect the system to require just a minimum adaptation in order to operate. Besides ROS, we are also using *sockets* from the operational system to implement network communication. The transportation robots are expected to communicate among each other, and with the General Manager, a central repository of information with a collective knowledge about all transportation requests in the factory, the localization of all known packages, a map of the environment, with the localization of known shelves and processing cells and all the identified AR-Tags and their meaning.

The following Codelets are specified to collect the necessary information from the environment:

- Camera Codelet
- Laser Range Finder Codelet
- Socket Codelet
- Battery Codelet
- SLAM Codelet
- Feedback Codelet

A detailed view of the Sensory Codelets and its connections to environment and Sensory Memory is available in Fig. 4.

The **Camera Codelet** is responsible for capturing information from the robot's cameras. The transportation robot has two RGB cameras, each one pointed to a different side

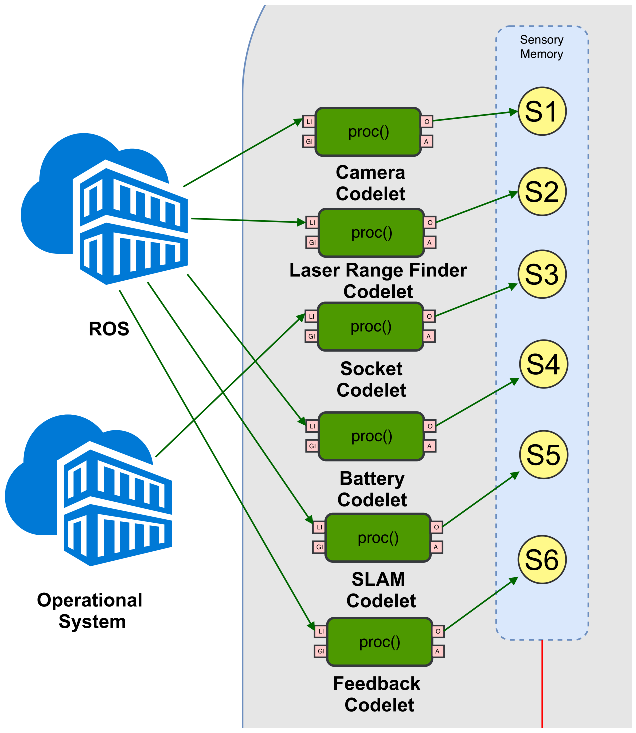

Fig. 4. A detail of the Sensory Codelets.

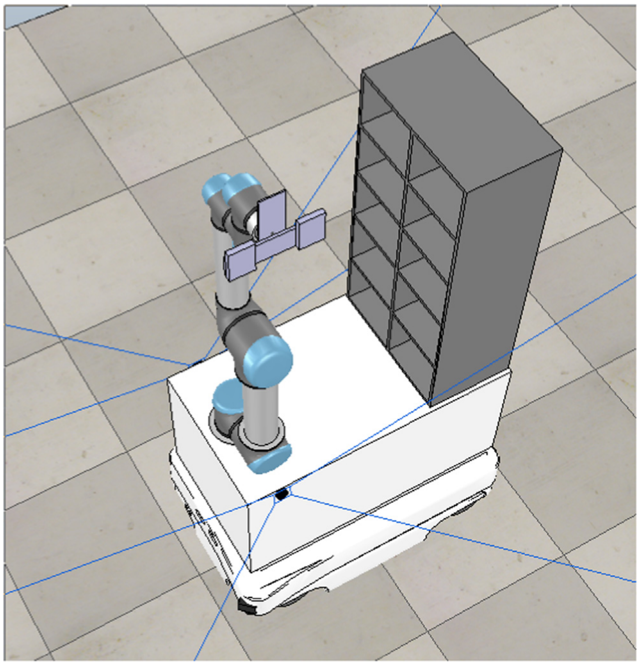

Fig. 5. A detail of the transportation robot cameras.

of the robot (see Fig. 5). The purpose of these cameras is to feed a module using the ARUCO library[4], such that it is able to detect the AR-Tags to extract their ID and reference frame's pose and posture in the environment. The Camera Codelet is not reading the image of these cameras, though.

The pipeline in Fig. 6 details the flow of information until the Camera Codelet can process its information. The images from both left and right cameras of each Transportation Robot is captured by V-REP in each simulation cycle, and grabbed by the Simulation Manager (see Section 5), using the V-REP RemoteAPI. The Simulation Manager then publishes these images as ROS nodes. The ARUCO library collects both images and detects all the AR-Tags appearing in those images. It then publish new ROS topics, with a list of detected AR-Tags in both of its sides. The output of ARUCO is a String ROS topic, with a JSON[5] array containing two lists: the list of detected AR-Tags in the left side and the list of AR-Tags in the right side. Each AR-Tag is represented by a Pose object and a numeric identifier (id) decoded from the AR-Tag. A *Pose* is a 7-tuple array, containing 3 float values representing the position (x, y, z) of the tag, in coordinates relative to the camera position, plus 4 more float values with the quaternion describing the rotation of the AR-Tag relative to the camera position. So, each Pose is a 7-tuple of floating point numbers describing the position and orientation

[4] The ARUCO library is a package within OpenCV, an open source library of programming functions mainly aimed at real-time computer vision.

[5] JSON (JavaScript Object Notation) is a lightweight data-interchange format.

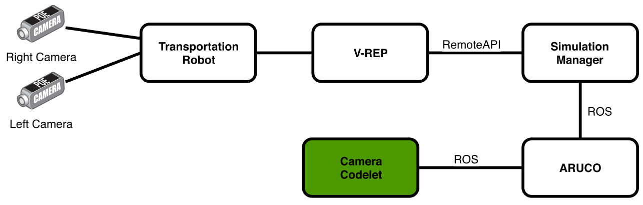

Fig. 6. The pipeline of information for the Camera Codelet.

of a detected AR-Tag. Each numeric id is unique to a different object in the whole system. We have numeric ids for every important object in the environment which needs to be identified: packages, shelves, processing stations, charging stations and robots.

This JSON string is the input for the Camera Codelet. The output of the Camera Codelet is then a proper Java representation of this list of poses and tag ids.

The **Laser Range Finder Codelet** is responsible for capturing information from the robot's laser scanners. The transportation robot counts with two Sick S300 laser scanners, one on the front of robot and other on the back. These range finders are positioned at 45° on the corners of the MIR100 and are able to scan 270° around. A schematic view of the sensor reading can be seen in Fig. 7.

The input of the Laser Range Finder codelet is a representation in ROS with the measurings from each laser scanner. Each laser scanner receives a vector with 541 elements, where each element represents the distance between the laser scanner and an obstacle at the environment, in a sweeping interval of 0.5° between each measuring. The maximum measurable distance is 30 meters.

All the important objects are expected to be detected with the Camera Codelet. The goal of the Laser Range Finder Codelet is to provide additional information allowing the detection of objects which do not have assigned AR-Tags, like obstacles in the environment, human beings, and possibly important object where the AR-Tag is not visible, as e.g. shelves or robots in particular positions where their AR-Tags are occluded or partially occluded, avoiding their decoding.

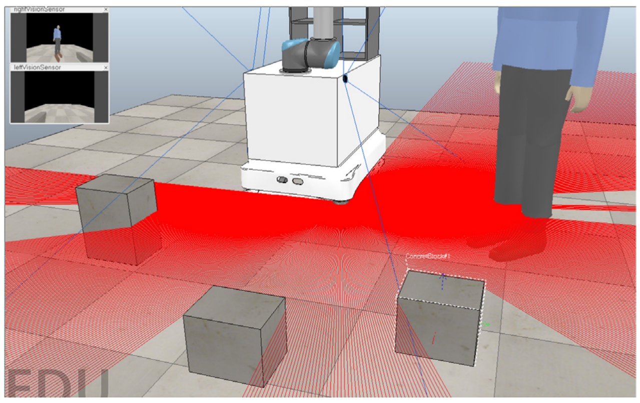

Fig. 7. Schematic of the laser range finder readings in the transportation robot.

The goal of the **Socket Codelet** is to provide communication between the transportation robot and the General Manager, and also among the transportation robots. There is also a Socket Motor Codelet, with the same purpose. The idea is to be able to send requests to the General Manager or to other robots, and be able to receive the response to these requests, or either to receive requests from the General Manager or from other robots.

The **Battery Codelet** captures from ROS messages the current level of the batteries, in terms of a percentage value.

The **SLAM Codelet** is responsible for providing a map of the known environment and the current robot position. It relies on ROS SLAM packages which, based on the Laser Range Finder information and odometry information, returns an environment map and a position, in terms of (x, y, θ) which is supposed to be the current robot position in 2D absolute coordinates. In a real robot, the odometry data comes from motion sensor (encoders and Inertial Measurement Unit), to estimate the change in position over time. In our simulation, we are able to get the robot position directly from V-REP. The real process of odometry is sensitive to errors due to the integration of velocity measurements over time to give position estimates. Rapid and accurate data collection, instrument calibration, and processing are required in most cases for odometry to be used effectively. Additional methods might use landmarks at the environment and the aid of laser range-finder data and vision from cameras. Because in this study our goal is not to focus on these problems, we are making a simplification and just using the V-REP position as if we already have a precise enough system of odometry.

The **Feedback Codelet** is another codelet which needs to be understood together with some of the Motor Codelets. The whole idea is that some of the commands sent through motor codelets might require some time to be processed. For example, the MIR100 Motor Codelet might command the robot to move to a given position, or the UR5 Motor Codelet might command the robot to pick a package at a given location. The Feedback Codelet is responsible for providing the feedback after these commands are concluded. This information is stored in ROS Messages, which are decoded by the Feedback Codelet and made available for the architecture. The following feedback is available:

- `pickCommand` concluded
- `placeCommand` concluded
- `moveToPosition` concluded
- Internal Slots Occupation: for each robot internal slot, the codelet returns the package occupying it. If the slot is free, a null package is said to be occupying the slot.

4.1.2. The Perception Codelet

The Perception Codelet unifies all the sensory information, abstracting part of this information and deriving high-level concepts, unified into a single object, called the

`Percept`. The `Percept` is then used as a source of information in many codelets in *System 1* and *System 2*.

4.1.3. The Motor Codelets

The Motor Codelets are the codelets responsible for commanding action at the environment. The following codelets are available:

- The MIR100 Codelet
- The UR5 Codelet
- The Socket Motor Codelet
- The Display Motor Codelet

A detailed view of the Motor codelets can be seen in Fig. 8.

The **MIR100 Motor Codelet** uses a ROS service available in the ROS system, to command the movement of the robot to a given pose (x, y, θ). The codelet simply executes the `moveToPosition` ROS service. The ROS service is responsible for controlling the MIR100 velocity on both wheels until the robot reaches the assigned position. As soon as the robot reaches the position, the ROS service sends a `moveToPosition` concluded ROS message, which is captured by the Feedback Codelet.

The **UR5 Motor Codelet** uses two ROS services available in the ROS system:

- The `pickCommand(pose,slot)` command
- The `placeCommand(pose,slot)` command

The `pickCommand` service tries to pick a package at the `pose` location, and store the package internally at its internal slot number `slot`. The `placeCommand` service tries to pick the package at the internal slot number `slot` and place it at the `pose` location. The location `pose` is defined relatively to the robot position, and not in world coordinates. This position usually will be provided by the AR-Tag of a package perceived by the robot.

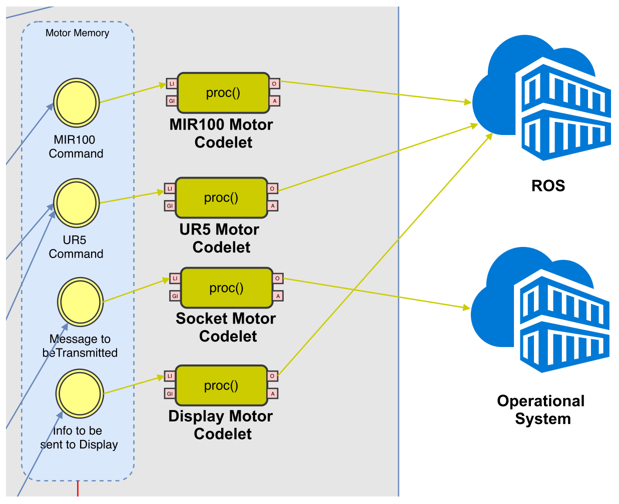

Fig. 8. A detail of the Motor Codelets.

The **Socket Motor Codelet** is the dual of the Sensory Socket Codelet, for providing socket communication between the robot and the General Manager, or to other robots.

The **Display Motor Codelet** is previewed for communication with human users. The idea is to have a display on the top of the robot where general information regarding the robot is made available for information purposes. Among the information we have:

- The current destination
- The list of future destinations already scheduled
- The list of packages being handled
- The current motivation affecting the robot
- The name of the robot, if the human user wants to communicate with the robot using its smart application

4.1.4. The Motivational Codelets

The Motivational Codelets are the main source of motivational behavior in the system. Each motivational codelet corresponds to one of the needs assigned to the system. The needs are related to the different tasks which can be assigned to the robot, and are expected to be weighted in order to generate the final behavior to be executed by the robot. The robot have autonomy to decide what to do at each given instant, and this autonomy is essentially attached to how much these needs are satisfied or not, depending on the current situation. The measure of the dissatisfaction of each need is said to be a *drive*. Each motivational codelet is responsible for generating this drive, based on the robots perception of the situation. A detailed view of all the motivational codelets can be seen in Fig. 9.

The **Energy Motivational Codelet** is responsible for evaluating if the need of energy of the robot is under a satisfying level. If the energy level is enough for the robot to provide service, the calculated drive is low. If the energy level is too low for providing service, the drive starts to increase. If this drive is too high, the robot might opt to stop everything and move to an energy docker to recharge its batteries.

The **Danger Avoidance Motivational Codelet** is responsible for calculating the danger avoidance drive. The purpose of this drive is to provide some aid to the “Avoid Colisions” Behavioral Codelet, such that it can integrate with all the other behaviors involving the robot’s motion around the environment. This policy is necessary, because all the other drives involving the robot’s motion are blind regarding possible collisions with objects from environment. This drive, together with the “Avoid Colisions” Behavioral Codelet, provides means for making small changes in the trajectory of the robot, whatever is this trajectory, to maintain a necessary distance from possible obstacles at the environment. The drive value is computed based on the information from the laser rangefinder and the current velocity vector. If the distance, in the direction of the velocity vector is beyond a given threshold, the drive value is

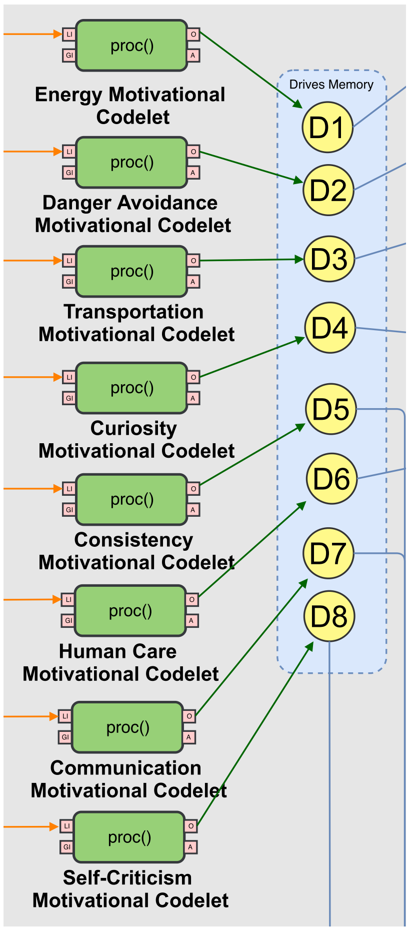
proc()
LI
GI
O
A
Energy Motivational Codelet
Drives Memory
D1
D2
D3
D4
D5
D6
D7
D8
Danger Avoidance Motivational Codelet
Transportation Motivational Codelet
Curiosity Motivational Codelet
Consistency Motivational Codelet
Human Care Motivational Codelet
Communication Motivational Codelet
Self-Criticism Motivational Codelet

Fig. 9. A detail of the Motivational Codelets.

low. Its value starts increasing while this distance starts to become smaller and is maximum when an imminent collision is detected.

The **Transportation Motivational Codelet** provides the main source of motivation in the system. The main need expected for the robot is to provide package transportation. But because the robot might have also other duties, as environment exploration, warehouse inventory, energy management, dealing with human operators and self-monitoring and benchmarking, the transportation drive is calculated based on the number of open requests for transportation reported by the General Manager. If the number of open requests is low, then this drive is low. As soon as this number starts to increase, the transportation drive also starts to increase.

The **Curiosity Motivational Codelet** is related to the robot's need to explore the environment. This need is basically set up by two different reasons. The first reason is due to a still incomplete map of the environment. The second reason is due to a long time since the robot visited particular areas of the factory. The Curiosity Motivational Codelet basically defines the curiosity drive, which will feed the Exploration Codelet, making the robot to go to regions of the factory where it has never been, or regions where there is a long time since its last visit to it. The rationale for this second reason for curiosity is to make the robot to visit regions of the factory, where the storage of packages might have been changed since its last visit. This provides support for the Warehouse Inventory task, in collaboration with the Consistency Motivational Codelet below. Basically, going to these regions, there is a greater chance to detect changes in the package storage, feeding the Communication Behavioral codelet to report these changes to the General Manager.

The **Consistency Motivational Codelet** is responsible for generating the Consistency Drive. This drive is associated to the warehouse inventory task, which should be performed all along other tasks. This means that, while traveling around the factory, the robot should be alert to things which are not where they were supposed to be. The consistency drive measures if things are as expected. This drive is calculated while the robot detects AR-Tags of both Locations and Packages and verifies if the Packages are stored in the correct Locations, as informed from the General Manager, or empty. Particularly the codelet tries to detect the following situations:

- A Location which should be occupied by a package is empty
- A Location which should be occupied by a given package is occupied by another package
- A Location which should be empty is occupied by a package

If any of these situations occur, the consistency drive raises its level, signalizing the Communication Behavioral Codelet the necessity to send a message to the General Manager informing the situation.

The **Human Care Motivational Codelet** is a second level security measure to the danger avoidance mechanism. The Danger Avoidance Motivational Codelet is measuring the distance between the robot and any possible obstacle. The Human Care Motivational Codelet estimates if this obstacle is due to a human being. If the robot detects the presence of human beings which could be harmed due to the robot's action, this drive should increase, following the distance detected to the closest human presence. Differently from the case with the Danger Avoidance drive, the expected behavior to the human care drive is for the robot to stop and wait until the distance to human beings goes beyond a security limit.

The **Communication Motivational Codelet** is responsible for the Communication Drive, i.e., the drive to respond to particular messages received by the Socket Sensor Codelet. If a received message requires the robot to provide a response, then this drive will be increased, being equal to 0 if there is no need to communicate.

The **Self-Criticism Motivational Codelet** is responsible for generating the self-criticism drive, which is responsible for the behavior necessary to fulfill the Self-Monitoring and Benchmarking task. This drive will basically motivate the Display Writing Codelet to elaborate performance statistics and send them to the robot display.

4.1.5. The Behavioral Codelets

The Behavioral Codelets are responsible for generating the behavior performed by the transportation robot. An overview of all the Behavioral Codelets can be seen in Fig. 10. In the TROCA behavioral system, we are introducing a new contribution to the MECA architecture. In the original MECA architecture (see (Gudwin et al., 2017)), all the behavioral codelets were feeding Memory Containers in the Motor Memory. This means that two or more behavioral codelets feeding the same Memory Container, would be competing for generating their prescribed action, using a dynamic subsumption scheme (Nakashima & Noda, 1998; Hamadi et al., 2010; Heckel & Youngblood, 2010).

Using the old scheme in MECA, the possible sequences of different behaviors are always due to a change in activation of a given behavior. As soon as the activation of a behavior becomes the highest one, this behavior wins the competition, and the behavior is changed to the new one. This process is always dynamic and unpredictable in nature, does not allowing the creation of habitual sequences of behavior which repeats itself, or which are coordinated in order to achieve a desirable state. With the current scheme, we introduced the possibility of having automatic sequences of behavior, which might be executed in a predictable way, leading the system to a desired situation. This is equivalent of having high level behaviors, which might be decomposed into a sequence of low level behaviors. These high-level behaviors might compete among them, in order to drive the final behavior of the robot.

Thus, our new contribution to MECA splits the behavioral codelets into two different categories:

- Multistep Behaviors
- Action Behaviors

The difference between multi-step behaviors and actions is that multi-step behaviors do not directly feed actuators in the Motor Memory. Instead, they feed objects in an Imagination Memory, which can then be used to trigger further Action Behaviors. Action Behaviors, from their side, work just like the old Behavioral Codelets in MECA, directly generating commands to actuators in the Motor Memory.

4.1.6. Multistep behaviors

The multi-step behaviors in the TROCA architecture can be seen in detail in Fig. 11.

The overall scheme for multistep behaviors requires two different Memory Container as output: the PlanRequest and the Plan. The idea is that each multi-step behavior is composed of a sequence of Action Behaviors. This sequence is represented into a Plan. We have then two different possible strategies for generating these Plans. These plans might be static, i.e. they might be defined “a priori”, and inserted in the code of the Multi-step behavioral codelets. Alternatively, they might be dynamic. In this case, we might require the collaboration of System 2 to generate the plans. At the beginning, the multi-step behavioral codelets do not have yet a plan. In this case, it generates a PlanRequest, which will be sent to System 2, requesting that it generates a plan for the multi-step behavior. System 2 will

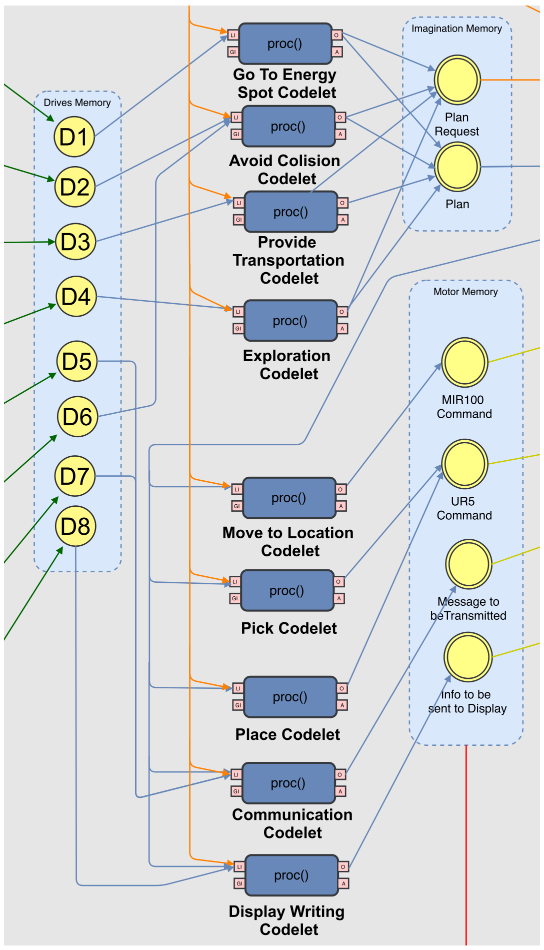

Fig. 10. A detail of the Behavioral Codelets.

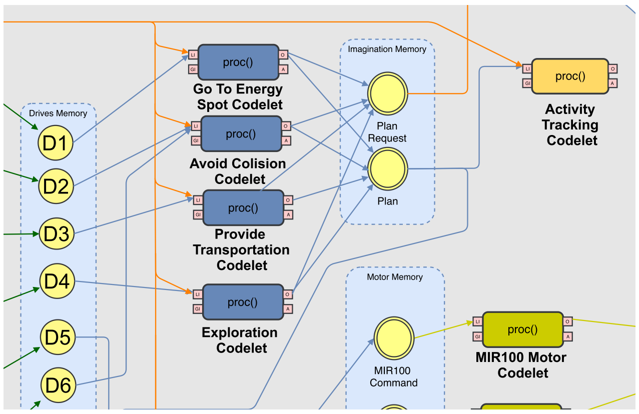

Fig. 11. A detail of the multistep Behavioral Codelets.

generate this plan, and would send it through the Global Workspace, being received by the multi-step behavioral codelet in its Global Input port. As soon as the multi-step behavior has a plan, it is now able to deliver the plan to the Plan Container. The many Plans, provided by all the multi-step behavioral codelets, will compete to each other, and the output of Plan Container will be the Plan with the highest activity level, being the local winner. The winner Plan will feed the Action Behavior codelets, making them act on the final Motor codelets, finally affecting the system.

The structure of a Plan can be seen in detail in Fig. 12. As can be seen in Fig. 12, a `Plan` is a sequence of `PlanSteps`, and each `PlanStep` is defined by a `BehaviorName` and a set of `BehaviorParameters`. A `Plan` also have a `CurrentStep`, which controls the execution of the `Plan`. The execution of the selected `Plan` is tracked by the **Activity Tracking Codelet** (see Fig. 11), which based on the information given by the `Percept`, defines if the `terminationCondition` of a given step already concluded, and if the `preCondition` of the next step is already valid, increments the `CurrentStep`, marking the next step in the plan as the current one, as suggested in the bottom of Fig. 12. The `Plan`, together with its `CurrentStep` which defines the current `PlanStep`, will be now the input of the many Action Behavior Codelets being commanded by the Plan. These Action Behavior Codelets will identify which is the current `PlanStep` and will check if the `BehaviorName` is their own. In the case there is a match, they will use the available `BehaviorParameters` for the `PlanStep` and provide a command in the Motor Memory.

In TROCA, we have 4 Multistep Behavioral Codelets, which compete to each other in order to command the robot:

- The Go To Energy Spot Codelet
- The Avoid Colision Codelet
- The Provide Transportation Codelet
- The Exploration Codelet

The **Go To Energy Spot Codelet** is responsible for creating a plan of actions moving the robot from its current position to the nearest reachable Recharging Station and staying there until the batteries are recharged. The `Plan` generated by this codelet is basically a sequence of poses (x, y, θ) which will be parameters for the Move To Location Codelet, remembering that the Move To Location Codelet is blind regarding possible obstacles, and providing the robot moving to the indicated pose in a straight line. Depending on the activity value of the energy drive, the activity level for this `Plan` should be higher or lower. In the case the battery level is too low, the activity level of this `Plan` should be high, and possibly this plan will be the winner.

The **Avoid Collision Codelet** is responsible for creating small corrections in the route when `Plans` coming from the Go To Energy Spot, Provide Transportation and Exploration Codelets puts the robot, for some reason, in a route of a possible collision. Usually a `Plan` created by this codelet is made of a single `PlanStep`, moving the robot to a slightly different position in order to avoid a possible collision. Whenever a plan created by this codelet is created, it usually have the highest activity level, in order to subsume all other plans, and avoid an imminent collision.

The **Provide Transportation Codelet** is responsible for creating plans suitable to provide the transportation of a package, attending a transportation request. These plans are the most complex among all other plans, because they involve moving the robot to the pick station, picking the package, storing the package internally within the robot, moving to the destination station, picking the package from its internal location within the robot and placing the package in its destination slot.

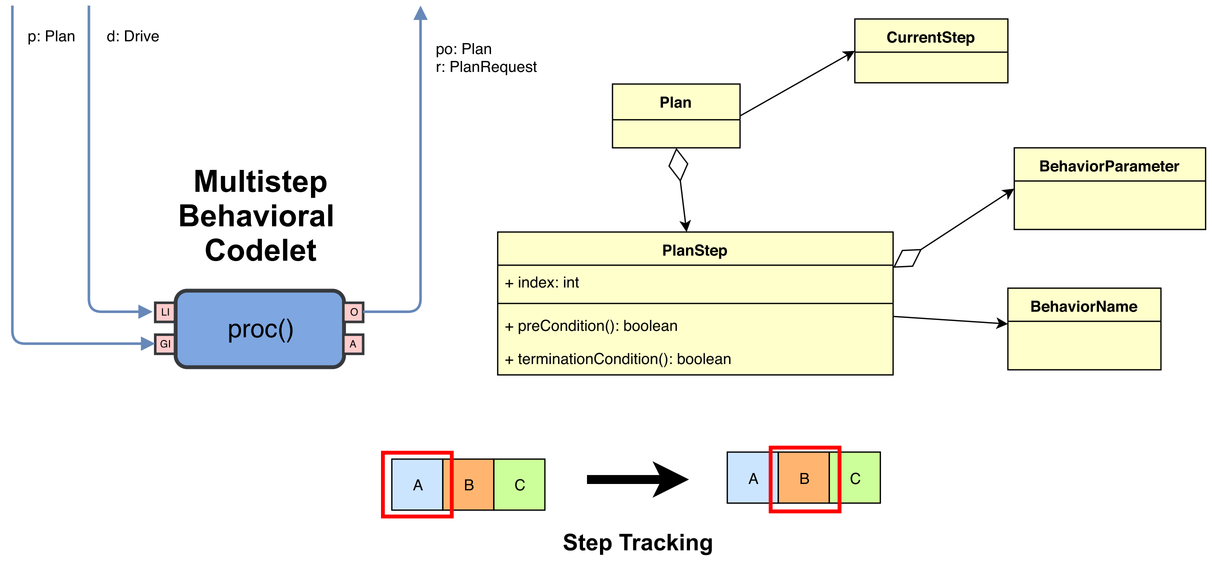

Fig. 12. The structure of a plan in a multistep Behavioral Codelet.

The **Exploration Codelet** is responsible for creating plans moving the robot to different locations in the factory, with the purpose of either exploring parts of the factory which are still unknown, or going to parts of the factory where there is a long time since the last visit. In this last case, the purpose is to perform warehouse inventory, allowing the discovery of packages stored in wrong locations or missing packages. The Exploration Codelet usually creates simple `Plans`, usually only moving the robot to specific locations, using a sequence of Move To Location actions.

4.1.7. Action behaviors

Differently from the Multistep Behavioral Codelets, Action Behavioral Codelets are responsible for controlling motor variables in the Motor Memory. Action behaviors corresponds directly to the older Behavior Codelets specified in the MECA Cognitive Architecture. Differently from the old MECA configuration, though, Action Behaviors might have as inputs not only drives from the Drives Memory, but also `Plans` stored in the Imagination Memory. A detail of the Action Behaviors used in the TROCA architecture can be seen in Fig. 13.

It is interesting to notice that all Action Behaviors receive as input the `Plan` coming from the Multi-step Behavioral Codelets. More than that, the Move to Location, Pick and Place Action Codelets do not receive any input from drives. Only the Communication Codelet and the Display Writing Codelet receive inputs from drives. All the Action Behavioral Codelets are described in the sequence.

The **Move To Location Codelet** is responsible for determining the commands to be sent to the MIR100 component of our Transportation Robot. This codelet uses as input the `Plan` output from the Multi-step Behavioral Codelets, and if some `BehaviorStep` has as a `BehaviorName` a symbol specifically addressing the Move To Location codelet, than it collect the `BehaviorParameters` identifying the location to move the robot to, and creates a request for the `moveToPosition` ROS service to be delivered by the MIR100 Motor Codelet.

The **Pick Codelet**, similarly, receives as input the `Plan` output from the Multi-step Behavioral Codelets, and if some `BehaviorStep` has as a `BehaviorName` a symbol specifically addressing the Pick codelet, than it collects the `BehaviorParameters` identifying both the pose from which a package should be picked up and the internal slot where the package should be stored, creating a request for the `pickCommand(pose,slot)` ROS service to be delivered to the UR5 Motor Codelet.

Also, the **Place Codelet**, receives as input the `Plan` output from the Multi-step Behavioral Codelets, and if some `BehaviorStep` has as a `BehaviorName` a symbol specifically addressing the Place codelet, than it collects the `BehaviorParameters` identifying the internal slot where the package should be picked and the pose to which a package should be placed, and creates a request for the `placeCommand(pose,slot)` ROS service to be delivered to the UR5 Motor Codelet.

The **Communication Codelet** is responsible for delivering messages through sockets for other robots and for the General Manager. It uses both the drives computed by the Consistency Motivational Codelet and the Communication Motivational Codelet for preparing and sending these messages.

Finally, the **Display Writing Codelet** receives the drive created by the Self-Criticism Motivational Codelet and uses it to send messages to be printed in the Robot Display.

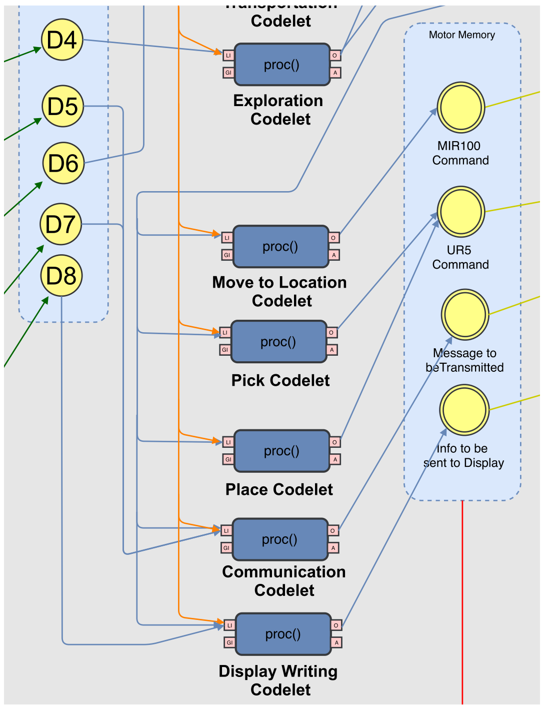

Fig. 13. A detail of the action Behavioral Codelets.

4.2. *The System 2 specification*

In the current version of the Cognitive Agent, *System 2* is dedicated to generating plans to fulfill requests coming from *System 1*. An overview of *System 2* can be seen in Fig. 14. Basically, these requests come from an Attention Codelet on the right side of the diagram, creating a *goal* in the Goal Memory. From the left bottom, another

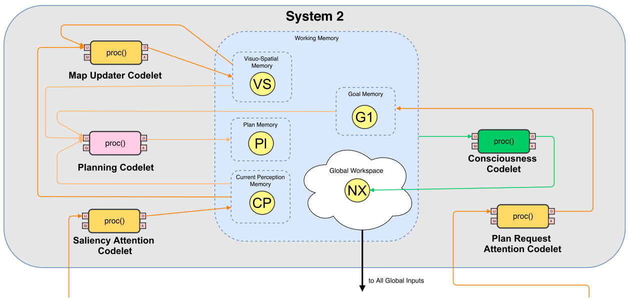

Fig. 14. A detail of the System 2 Subsystem.

Attention Codelet creates a Current Perception at the Current Perception Memory. Then, the Planning Codelet is able to create a plan and store it in the Plan Memory. After that, the Consciousness Codelet picks this plan and broadcasts it back to *System 1*.

It is important for the reader to compare TROCA's *System 2* with the *System 2* available in MECA (see (Gudwin et al., 2017)). We are not using here many of the resources available in MECA, like the Episodic Memory and the Expectation subsystems. We intend to use these, in future versions of TROCA, though. They are not being included here, because the first version of TROCA is meant to be simpler, attending the specifications. Further enhancements will extend the current architecture to include these features.

Even though *System 2* processes are completely independent of *System 1* processes, they are connected in a symbiotic way, in which *System 1* provides the information for *System 2* to start working, and *System 2* provides the information required for *System 1* to continue its operation. In the sequence, we detail the overall process, by describing the performance of each of the *System 2* codelets.

The **Saliency Attention Codelet** is one of the input ports to *System 2*, selecting from the Percept generated by the Perception Codelet, the information necessary for the Planning Codelet to effectively generate its plans. A detail of the Saliency Attention Codelet can be seen in Fig. 15.

The second input port to *System 2* is the **Plan Request Attention Codelet**. A detail of this codelet can be seen in Fig. 16. As described previously, whenever one of the Multistep Behavior Codelets has an invalid or outdated plan, it generates a plan request in the Imagination Memory, which is detected by the Plan Request Attention Codelet.

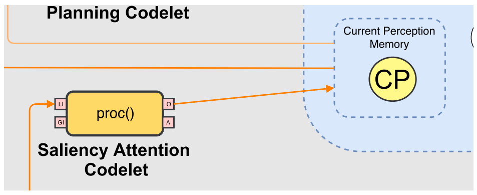

Fig. 15. A detail of the saliency Attention Codelet Codelet.

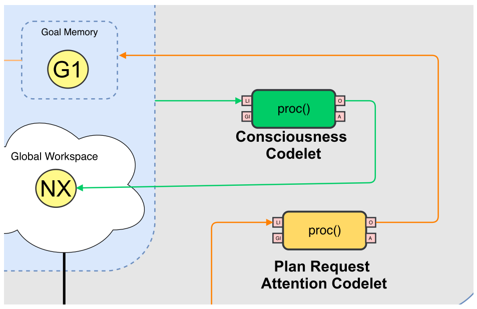

Fig. 16. A detail of the plan request Attention Codelet Codelet.

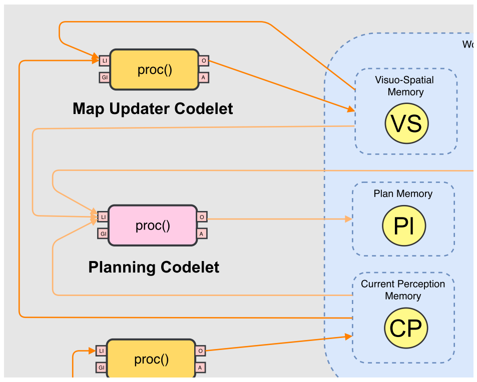

Fig. 17. A detail of the map updater and Planning Codelets.

From the `PlanRequest`, the Plan Request Attention Codelet identifies the `Goal` to be reached, which is necessarily different, depending on the Multistep Behavior Codelet it originates from. For example, for the Go To Energy Spot Codelet, the `Goal` is to have the robot's battery level completely replenished. For the Avoid Collision Codelet, the `Goal` is to be sufficiently far from the identified collision object, such that a collision is avoided. For the Provide Transportation Codelet, the `Goal` is to have the identified package located in its destination Location. And for the Exploration Codelet, the `Goal` is to be located in a defined position of the factory.

The Plan Request Attention Codelet then represent this Goal in a way it can be used for the planning system and stores it in the Goal Memory.

A detail of the **Map Updater Codelet** can be seen in Fig. 17.

The Map Updater Codelet is responsible for picking up the SLAM material available in the Current Perception Memory, which provides a part of World Map (as Perceived by the Perception Codelet and made available in *System 2* by the Saliency Attention Codelet), and updating the known Environment Map in the Visuo-Spatial Memory. This Map is a list of all known objects at the environment, defined by their type, position and pose. The Map is important for the Planning Codelet to generate its plans.

The **Planning Codelet** is responsible for generating the `Plans` requested by the Multistep Behavioral Codelets. A detail of the Planning Codelet can be seen in Fig. 17. This `Plan` is constructed using SOAR (Laird, 2012), as the standard planning strategy in MECA (Gudwin et al., 2017). Before calling SOAR, though, the Planning Codelet first grabs the information coming from the Current Perception Memory, the Visuo-Spatial Memory and the Goal Memory, transforming all of them in WMEs[6], such that they can be inserted in SOAR's input link. After SOAR halts, the Planning Codelet incorporates SOAR's output link information in order to build a `Plan` object, which is inserted in the Plan Memory.

[6] WMEs, or Working Memory Elements, are the canonical representation structure used within SOAR.

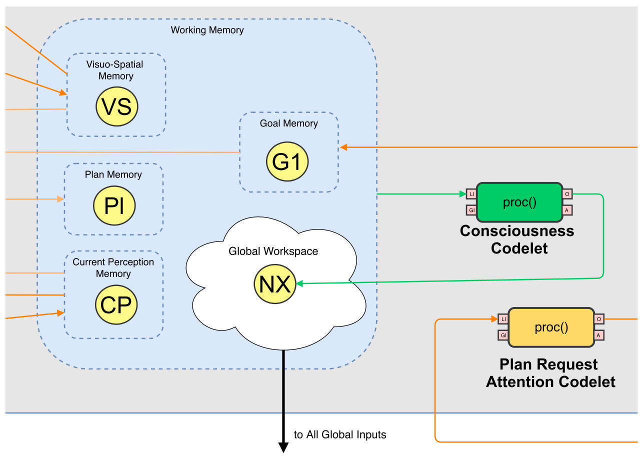

Fig. 18. A detail of the Consciousness Codelet.

The **Consciousness Codelet** is responsible for scanning all the Working Memory and finding suitable information to be broadcasted to the Global Workspace, and from there to all Global Inputs of all Codelets in the system, following the consciousness mechanism provided in Baar's Global Workspace Theory (Baars, 1988). A detail of the Consciousness Codelet can be seen in Fig. 18.

Most of the times, the Consciousness Codelet will be selecting Plans from the Plan Memory and broadcasting it such that the requester Multistep Behavioral Codelet is able to identify a response to its request, picking the plan and preparing it for execution.

5. The software implementation

In Section 4 we developed a detailed specification for the TROCA Cognitive Agent, controlled by an instance of the MECA Cognitive Architecture. In order to run our experiments, binding our TROCA agent to the simulated Transportation Robots in V-REP, we had to develop a quite sophisticated software implementation, in order to attend the multiple specifications provided in Sections 3 and 4. In this section, we detail the software infrastructure we had to develop in order to run the simulations.

Fig. 19 shows how the different components of the architecture should communicate to each other. All the Transportation Robots should communicate to the General Manager, and also to each other. From the other side, all the Human Operators should communicate to the General Manager.

Using only this perspective, the overall distributed system should contain, at least, the following software components:

- The General Manager
- The TROCA Agent
- The Smartphone Application

But there were many other specifications that had to be considered. First, we were not running a real robotic scenario, but a simulated one, using the V-REP simulator. Because our factory scenario is rather complex, we envisioned that a single V-REP instance would not have enough computational power to simulate the whole factory. With this in mind, we envisioned a distributed simulation configuration, where multiple V-REP instances should be running in different machines. For managing these multiple V-REP instances, we decided to create a Simulation Manager software, responsible for integrating all V-REP instances into a unified scenario. Besides that, we had the specification to use ROS (Robot Operating System) to communicate between the TROCA Agent controller and its hardware (real or simulated). By using ROS, we were anticipating a future possibility of having the TROCA Agent controlling real versions of the robot, with a minimum refactoring of the code. But the use of ROS comes with a cost. Even though, in principle ROS should be available for multiple kinds of operational systems, in reality the versions of ROS are very much tied to Linux Ubuntu specific versions. The version of ROS we decided to use (Melodic Morenia) is particularly tied to Ubuntu 18.04 version. In order to not be tied to this operational system, we decided to use the Docker tool. Docker is a software solution where you package software into standardized units, called containers, for development, shipment and deployment, embedding into them everything the software needs to be run, including the operational system. Differently from a virtual machine, where a whole machine is virtualized, with a corresponding operational system, a Docker container image is a lightweight, standalone, executable package of software that includes only the software needed to run an application: code, runtime, system tools, system libraries and settings. Depending on the host operational system, a Docker container might have or not some sort of virtualization, but it is transparent to the user.

The final simulation arrangement took the configuration pictured in Fig. 20.

5.1. Simulation manager

The Simulation Manager (SM) is responsible for many different tasks regarding the simulation. It was built using a Python stack encapsulated in a Docker container, together with ROS and all its required environment.

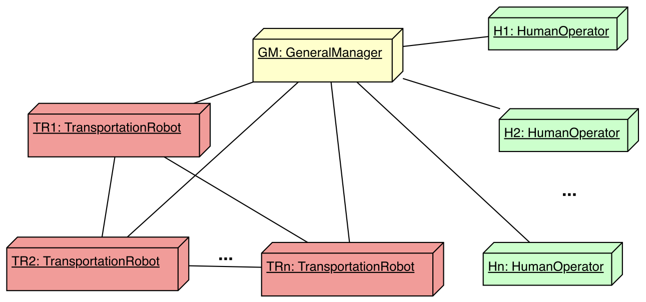

Fig. 19. The general architecture.

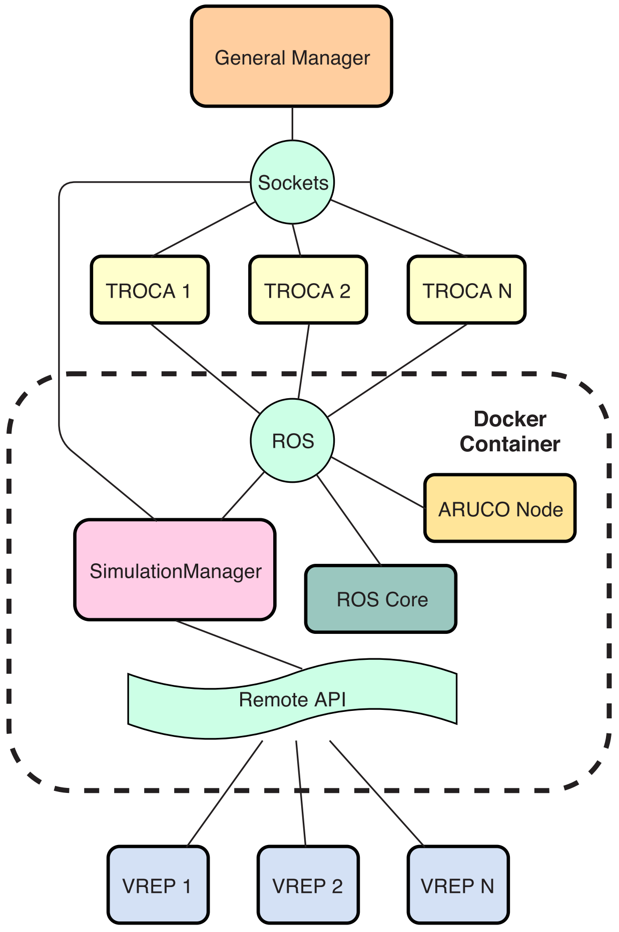

Fig. 20. The software packages architecture.

Besides coordinating the simulation in all V-REP instances, the Simulation Manager is also responsible for the creation and management of the simulation scenario, positioning the many items available within the factory:

- Shelfs (static, not created or moved during simulation);
- Processing units (static, not created or moved during simulation);
- Charging stations (static, not created or moved during simulation);
- Packages (dynamic, created and moved during simulation);
- Robots (dynamic, created and moved during simulation).

Besides that, the Simulation Manager can also be used to randomly generate different patterns of package transportation requests, allowing the configuration of different simulations, with different levels of transportation demands.

After initiating a simulation, with a factory configuration and a prescribed traffic of packages demand, the Simulation Manager starts many different processes responsible for running the experiment (see Fig. 21). The Simulation Manager starts the ROS Manager process,

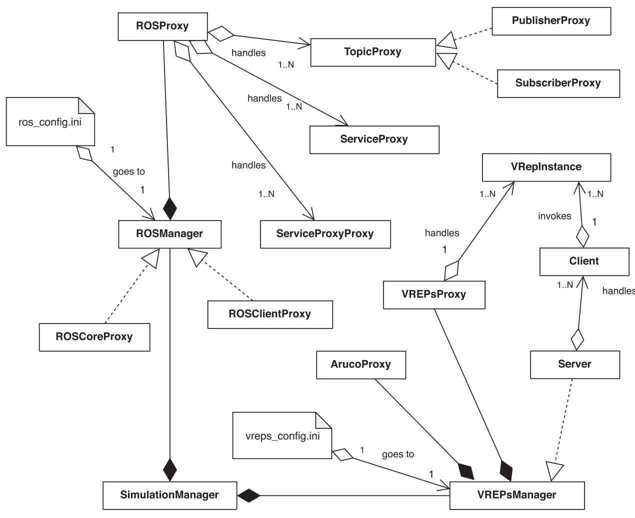

Fig. 21. The simulation manager framework.

responsible for managing ROSProxy, capable of publishing and subscribing to ROS topics, and also managing ROS services. Moreover, the Simulation Manager starts all V-REP instances, in a different set of machines, and starts controlling them.

5.2. General Manager

The General Manager (GM) is a global database, shared by all the transportation robots, and responsible for managing and storing all the objects perceived and reported by the transportation robots, during its execution. The General Manager was implemented using a Node.js + MongoDB stack, so that information is persistent and can be used to generate statistic reports as an outcome of the simulation. Fig. 22 shows the knowledge base of the General Manager. Among the many concepts held by the GM, the most important ones are the relations between AR-Tags and packages, such that any agent, human or robot, can at any time retrieve from the GM the ID and location of a given package related to some AR-Tag, and also a model of the human (Person) users, their roles and contacts.

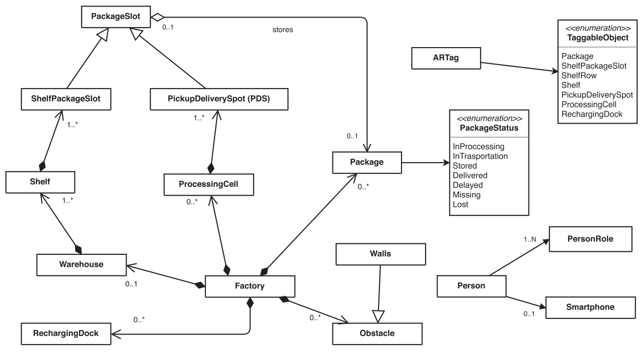

Fig. 22. The general manager knowledge base.

5.3. TROCA Agent

The TROCA Agent software embeds the Cognitive Architecture controlling each transportation robot in the scene. It was built in Java, using the CST and MECA libraries, also developed in Java. We also use the ROSJava library, which provides TROCA with the access to the ROS Topics and Services, controlled by the Simulation Manager. Each robot runs its own instance of TROCA.

5.4. Human operators and the smartphone application

In the original specifications, human operators were supposed to interact with the system, by walking in the factory floor and communicating with the system using Smartphone Applications. They were included because human-robot interaction is a very important topic to be addressed in an automatized factory with autonomous robots. Nevertheless, for our simulation purposes, these human operators are also simulated within V-REP, crossing the way of the transportation robots and eventually picking and moving packages, activities which must be detected and discovered by the warehouse inventory task of the robots. We didn't really developed a Smartphone Application for this purpose, but in a future implementation this is the way to go to introduce the interaction with real humans in a real environment.

6. Conclusion

In this paper, we presented the development of a transportation agent, using the MECA Cognitive Architecture as a background, responsible for multiple assignments in a factory floor. The experiment was constructed using the V-REP robotic simulator, where a factory scenario was built, using ROS as a background infrastructure for controlling the robots. The TROCA cognitive agent was built using the MECA Cognitive Architecture in Java language, and the CST Cognitive Systems Toolkit. In this paper, we focused in the description of our study case and in the details of the cognitive architecture developed to control the transportation robots. In a future publication, we pretend to explore different simulation scenarios, with a detailed analysis on the performance of the current architecture, under different demands, and describe the many parameters which must be tuned in order for the architecture to have different emphasis regarding the many simultaneous needs being addressed and how these needs interrelate to each other in order to build a characteristic behavior for our autonomous robot.

Acknowledgments

The authors thank Ericsson Research Brazil, Ericsson Telecomunicações S.A. Brazil (Proc. FUNCAMP 4881.7) and CEPID/BRAINN (Proc. FAPESP 2013/07559-3) for supporting this research.

References

Adinandra, S., Caarls, J., Kostić, D., Verriet, J., & Nijmeijer, H. (2012). Flexible transportation in warehouses. In *Automation in Warehouse Development* (pp. 191–207). Springer.

Augello, A., Infantino, I., Lieto, A., Pilato, G., Rizzo, R., & Vella, F. (2016). Artwork creation by a cognitive architecture integrating computational creativity and dual process approaches. *Biologically Inspired Cognitive Architectures, 15*, 74–86.

Avery, E., Kelley, T., & Davani, D. (2006). Using cognitive architectures to improve robot control: Integrating production systems, semantic networks, and sub-symbolic processing. In *15th Annual conference on behavioral representation in modeling and simulation (BRIMS)*. .

Baars, B. J. (1988). *A cognitive theory of consciousness*. Cambridge University Press.

Barsalou, L. W. (2010). Grounded cognition: Past, present, and future. *Topics in cognitive science, 2*(4), 716–724.

Benjamin, D. P., Lyons, D. M., & Lonsdale, D. W. (2004). Adapt: A cognitive architecture for robotics. In *ICCM* (pp. 337–338).

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation, 2*(1), 14–23.

Burghart, C., Mikut, R., Stiefelhagen, R., Asfour, T., Holzapfel, H., Steinhaus, P., & Dillmann, R. (2005). A cognitive architecture for a humanoid robot: A first approach. In *5th IEEE-RAS International conference on humanoid robots, 2005* (pp. 357–362). IEEE.

Christaller, T. (1999). Cognitive robotics: A new approach to artificial intelligence. *Artificial Life and Robotics, 3*(4), 221–224.

Clark, A., & Grush, R. (1999). Towards a cognitive robotics. *Adaptive Behavior, 7*(1), 5–16.

Evans, J. S. B. (2003). In two minds: Dual-process accounts of reasoning. *Trends in Cognitive Sciences, 7*(10), 454–459.

Faghihi, U., Estey, C., McCall, R., & Franklin, S. (2015). A cognitive model fleshes out kahneman's fast and slow systems. *Biologically Inspired Cognitive Architectures, 11*, 38–52.

Franklin, S., Madl, T., D'mello, S., & Snaider, J. (2014). Lida: A systems-level architecture for cognition, emotion, and learning. *IEEE Transactions on Autonomous Mental Development, 6*(1), 19–41.

Gärdenfors, P. (2014). *The geometry of meaning: Semantics based on conceptual spaces*. MIT Press.

George, D., & Hawkins, J. (2009). Towards a mathematical theory of cortical micro-circuits. *PLOS Computational Biology, 5*(10), e1000532.

Gudwin, R., Paraense, A., de Paula, S. M., Fróes, E., Gibaut, W., Castro, E., Figueiredo, V., & Raizer, K. (2017). The multipurpose enhanced cognitive architecture (meca). *Biologically Inspired Cognitive Architectures, 22*, 20–34.

Gudwin, R., Paraense, A., de Paula, S. M., Fróes, E., Gibaut, W., Castro, E., Figueiredo, V., & Raizer, K. (2018). An urban traffic controller using the meca cognitive architecture. *Biologically Inspired Cognitive Architectures, 26*, 41–54.

Gudwin, R. R. (2015). Computational semiotics: The background infrastructure to new kinds of intelligent systems. *APA Newsletter – Philosophy and Computers, 15*(1), 27–38, URL<http://www.apaonline.org/resource/collection/EADE8D52-8D02-4136-9A2A-729368501E43/ComputersV15n1.pdf>.

Gudwin, R. R. (2016). *Urban traffic simulation with SUMO - A Roadmap for the beginners, Technical Report D3*. Campinas-SP, Brazil: University of Campinas.

Hamadi, Y., Jabbour, S., & Saïs, L. (2010). Learning for dynamic subsumption. *International Journal on Artificial Intelligence Tools, 19*(04), 511–529.

Heckel, F. W., & Youngblood, G. M. (2010). Multi-agent coordination using dynamic behavior-based subsumption. In *Sixth artificial intelligence and interactive digital entertainment conference - AIIDE*. .

Jilk, D. J., Lebiere, C., O'Reilly, R. C., & Anderson, J. R. (2008). Sal: An explicitly pluralistic cognitive architecture. *Journal of Experimental and Theoretical Artificial Intelligence, 20*(3), 197–218.

Joseph, L. (2015). *Mastering ROS for robotics programming*. Packt Publishing Ltd..

Kelley, T. D. (2006). Developing a psychologically inspired cognitive architecture for robotic control: The symbolic and subsymbolic robotic intelligence control system (ss-rics). *International Journal of Advanced Robotic Systems, 3*(3), 32.

Kotseruba, I., Gonzalez, O. J. A., & Tsotsos, J. K. (2016). A review of 40 years of cognitive architecture research: Focus on perception, attention, learning and applications', arXiv preprint arXiv:1610.08602.

Kurup, U., & Lebiere, C. (2012). What can cognitive architectures do for robotics? *Biologically Inspired Cognitive Architectures, 2*, 88–99.

Lacher, A., Grabowski, R., & Cook, S. (2014). Autonomy, trust, and transportation. In *2014 AAAI Spring Symposium Series*. .

Laird, J. E. (2012). *The Soar cognitive architecture*. MIT Press.

Lemaignan, S., Ros, R., Mösenlechner, L., Alami, R., & Beetz, M. (2010). Oro, a knowledge management platform for cognitive architectures in robotics. In *2010 IEEE/RSJ International conference on intelligent robots and systems* (pp. 3548–3553).

Levesque, H., & Lakemeyer, G. (2008). Cognitive robotics. *Foundations of Artificial Intelligence, 3*, 869–886.

Lieto, A., Chella, A., & Frixione, M. (2017). Conceptual spaces for cognitive architectures: A lingua franca for different levels of representation. *Biologically Inspired Cognitive Architectures, 19*, 1–9.

Lieto, A., Radicioni, D. P., & Rho, V. (2017). Dual peccs: a cognitive system for conceptual representation and categorization. *Journal of Experimental & Theoretical Artificial Intelligence, 29*(2), 433–452.

Moreno, A., Umerez, J., & Ibañez, J. (1997). Cognition and life: The autonomy of cognition. *Brain and Cognition, 34*(1), 107–129.

Nakashima, H., & Noda, I. (1998). Dynamic subsumption architecture for programming intelligent agents. In *Proceedings of the 3rd international conference on multi agent systems - ICMAS* (pp. 190–197). IEEE Computer Society.

Osman, M. (2004). An evaluation of dual-process theories of reasoning. *Psychonomic Bulletin & Review, 11*(6), 988–1010.

Paraense, A. L. O., Raizer, K., de Paula, S. M., Rohmer, E., & Gudwin, R. R. (2016). The cognitive systems toolkit and the cst reference cognitive architecture. *Biologically Inspired Cognitive Architectures, 17*, 32–48.

Raizer, K., Paraense, A. L. O., & Gudwin, R. R. (2012). A cognitive architecture with incremental levels of machine consciousness inspired by cognitive neuroscience. *International Journal of Machine Consciousness, 04*(02), 335–352, URL<http://www.worldscientific.com/doi/abs/10.1142/S1793843012400197>.

Rohmer, E., Singh, S. P., & Freese, M. (2013). V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International conference on intelligent robots and systems (IROS)* (pp. 1321–1326). IEEE.

Samsonovich, A. V. (2010). Toward a unified catalog of implemented cognitive architectures. *BICA, 221*, 195–244.

Sun, R. (2003). A tutorial on clarion 5.0', Unpublished manuscript.

Thórisson, K., & Helgasson, H. (2012). Cognitive architectures and autonomy: A comparative review. *Journal of Artificial General Intelligence, 3*(2), 1–30.

Trafton, J. G., Hiatt, L. M., Harrison, A. M., Tamborello, F. P., II, Khemlani, S. S., & Schultz, A. C. (2013). Act-r/e: An embodied cognitive architecture for human-robot interaction. *Journal of Human-Robot Interaction, 2*(1), 30–55.

Ziemke, T., & Lowe, R. (2009). On the role of emotion in embodied cognitive architectures: From organisms to robots. *Cognitive Computation, 1*(1), 104–117.