A METHODOLOGY TO IMPLEMENT SUBSUMPTION ARCHITECTURE BASED ON DISCRETE EVENT SYSTEMS FORMALISM

Phillipe Cardoso Santos*, Elyson Ádan Nunes Carvalho*, Lucas Molina*, Eduardo Oliveira Freire*

*Federal University of Sergipe
São Cristóvão, Sergipe, Brazil

Emails: phillipe@gprufs.org, ecarvalho@ufs.br, lmolina@ufs.br, efreire@ufs.br

Abstract— This paper presents a methodology to implement subsumption architecture based on discrete event systems (DES) formalism. For this, two automata models are presented so that all behaviors can be modeled easily, even for complex tasks. It also allows the inclusion of new behaviors on the system without changing the ones previously modeled. This method is tested in real experiments for three behavior layers with a *pioneer 3-DX* mobile robot and results show it as a feasible and efficient approach.

Keywords— Mobile robot, subsumption architecture, discrete event systems, automata

1 INTRODUCTION

In robotics, decision-making and execution is an important aspect used by the system to achieve its objective. In mobile robotics, the specific aspect directly linked to robust mobility is navigation competence (Siegwart and Nourbakhsh, 2004), which can be described as the process to determine an adequate and secure path from the start point to the goal point, considering the pieces of information available to the robot (Ranganathan and Koenig, 2003).

In this context, navigation architectures are researched in order to solve the problem of decision-making by the robot. A robot architecture defines how is organized the task of generating action through perception (Russell et al., 2003).

In navigation area is common to divide a complex task (global task) in some simple tasks (subtasks), which are performed through independent navigation systems called behaviors (Arkin, 1998). Then, the problem stands in how to coordinate these behaviors.

One of the most important and refereed theory in this area is the subsumption architecture, also known as behavior-based control, developed by Brooks in 1986 (Brooks, 1986). This theory started reactive navigation architecture when Brooks tried to solve deliberative approaches problems for dynamic and unknown environments.

In his work, Brooks created a set of behaviors organized in layers to accomplish tasks. These behaviors are triggered by stimuli from sensor responses. Each behavior accepts suppression and inhibition signals, where a suppression signal overrides normal input signal and the inhibiting signal inhibits completely the output. These signals allow behaviors overlap, so that the system can produce a coherent behavior.

In addition, many behaviors can be triggered simultaneously, being necessary a mechanism of choice that selects the corresponding action. For this, behavioral modules are arranged in layers in a hierarchy called subsumption, in which the lower layers are inhibited by the higher ones. Subsumption architecture idea is showed in Figure 1.

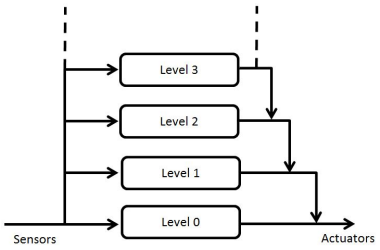

Figure 1: Layered behavior with higher levels subsuming the roles of lower level layers.

Almost 30 years has passed and Brooks' work is still cited by many authors, what shows the importance of this navigation architecture to mobile robotics. However, this is also a reflex of some problems found in this approach, such as the behaviors design for complex tasks becomes complicated; the appearance of situations where an impasse occurs and two or more process are unable to continue their execution; and the way how behaviors are encapsulated in finite state machines, which makes some intern states unaccessible to control layers. Furthermore, a formal description to implement these finite state machines is necessary in order to demonstrate its correctness.

Examples of such formalisms can be seen in (Amir and Maynard-Zhang, 2004), where a logical framework is used to implement subsumption architecture and (Baader et al., 1998) which uses an automata theoretic approach to reduce the problem into a language problem.

In this paper is proposed a methodology for designing subsumption architecture based on Discrete Event Systems (DES) formalism. This approach allowed modeling different behaviors eas-

ily, even for complex tasks, and it also allowed the inclusion of new behaviors on the system without changing the ones previously modeled. Additionally, an automaton model was created, so that all new behavior can be modeled with the same pattern.

In order to demonstrate the methodology feasibility, real experiments were performed for three behavior layers with a *pioneer 3-DX* mobile robot.

This paper is structured as follows. In section II is present a brief review about subsumption architecture. In section III the proposed methodology is presented. Results are presented in section IV and conclusions in section V.

2 Background

In 1986, Ronald Brooks designed a control architecture based on behaviors which tackles the control problem by thinking of it as a number of horizontally arranged layers. In the light of this decomposing method, a control system can be divided into a group of sub-systems, which are able to implement one behavior independently (Yongjie et al., 2006). This control method, named subsumption architecture, is very used in robotics for many different applications, such as intelligent cars, multiple robots (Nagata et al., 2013), aerial robotics and even underwater vehicles (Godin et al., 2010).

The advantages of subsumption architecture stands on the way it decomposes the system into a series of units vertically; it also eliminates the bottleneck, which occurs when sensors data must be fused with data from other sensors. Instead, control behaviors receive sensory data that is relevant only to their particular decision-making needs (Yongjie et al., 2006). Furthermore, all of the modules in subsumption can function independent of any other module in the system, introducing an increase in processing speed and improving robustness (Turner et al., 2013).

However, this hierarchically layered structure has some limitation which are described in (Yongjie et al., 2006) on the following items:

- In order to make a most proper decision, it needs to weight many factors in the selection of control commands. It is sometimes unapt for behavior of high level to inhibit that of lower level, which will miss all information of the inhibited layers;
- Another problem is encapsulating behaviors within a set of finite state machines, which make the internal state of these behaviors inaccessible to control layers. The designer of each layer cannot think out what other layers will do and he cannot forecast the relationship between his layer and other layers either. As a result, once a new behavior layer is added, other layer need to adjust accordingly;
- Although subsumption architecture raises the response speed of the system, its behavior design is comparatively simple, and short for necessary competence of reasoning and coordination. On average, it can get satisfying results for some simple tasks, but for a more complicated one, the limitation of lacking planning ability emerged clearly.

Many different methodologies were used to implement subsumption architecture in order to make it more robust. Connell (Connell, 1992) combined subsumption with a servo-control layer and a symbolic layer in a way that allows the advantages of each technique. Yongjie (Yongjie et al., 2006) proposed a hybrid architecture based on subsumption and a behavior based planning process. Antonelli (Antonelli et al., 2008) presented a technique called Null-Space Based Behavioral (NSB) to deal with relative priority of each task. Turner (Turner et al., 2013) implemented subsumption architecture by using Model Driving Development (MDD) and modeled behaviors through finite state machines.

In order to keep improving the method robustness and also facilitate its design, this paper proposes a methodology based on Discrete Event Systems (DES), modeling behaviors as automata. The idea is presented in the following section.

3 The Proposed Methodology

The proposed methodology consists in modeling each layer as an automaton which contains the activation of the controller for that subsystem and a switching rule to enable or not the activation of lower layers behaviors.

The switching rule used depends on whether the behavior goal is being achieved and how long it can be disturbed. In other words, if the behavior goal of a high priority layer is not being performed correctly, it is not interesting for the system enabling the activation of a different controller from a lower layer, even if a necessity for its activation arises. This controller would probably disturb more the goal achievement, once it may have a different objective.

Nevertheless, if the behavior goal of a high priority layer is being performed correctly and a necessity to activate a behavior from a lower layer arises, its activation can be allowed since it does not disturb significantly higher priority objectives. Hence, disturbance is allowed only during a short time τ to not compromise the system. Right after τ, the system checks if the higher behavior goal is still being performed correctly in order to continue allowing lower behaviors to be used or not.

Hence, a temporal checking is done based on a time τ in order to assure the hierarchy accomplishment. This time is defined by the designer and it must be done carefully, because this parameter represents how flexible is the system in relation to disturbance from lower priority behaviors.

This approach is an hierarchical cascade system where higher priority goals are desirable even if lower ones do not be achieved. And it can be more robust if the highest priority controller is globally asymptotically stable, because the proposed methodology makes the system always return to it after a short time τ, trying to lead the main objective to be accomplished.

Once the switching rule was defined, behaviors were modeled based on Discrete Event Systems formalism as automata. This approach avoided the requirement of modeling complex behaviors, thereby providing a foundation to make it by modeling small parts of the overall desired behavior and composing them through parallel automata composition (Cassandras and Lafortune, 2009).

Additionally, two automata models are proposed in order to design behaviors simply and easily. The first model is for the highest priority behavior and the other for all the lower behaviors, even new ones which may be still added to the system.

These automata models and a result of parallel operation are presented in next section.

3.1 Behavior automata models

The system modeling for this approach starts in the highest layer, where the most desirable behavior must be. This layer is designed based on the events defined at table 1 and the automaton can be seen at Figure 2.

Table 1: Events description of the highest priority layer automaton

Events	Description
n_1	Necessity to activate controller 1
nn_1	Non necessity to activate controller 1
A_{c1}	Activates controller 1
r_1	Switching rule being satisfied
E_{L2}	Enable layer 2
t	Time τ achieved

In this automaton the system starts checking the necessity to activating controller 1. If necessary (event n_1), controller 1 is activated(event A_{c1}) and then it is checked if the switching rule is being satisfied . If the behavior goal for this layer is being performed correctly (event r_1), the system enables the second layer (event E_{L2}) during the time τ. Right after τ (event t), the second layer must be disabled and the system return the initial state $H1$. When there is no necessity to activate controller 1 (event nn_1),the marked state $H0$ is selected, meaning the system main objective was achieved.

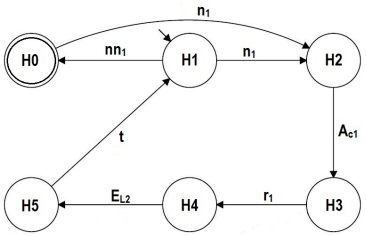

Figure 2: Highest priority layer automaton

An important point to highlight is that enabling the second layer does not mean the controller 2 will be activated. Its activation is set by the next automaton presented in Figure 3, which is designed based on the events of the table 2

Table 2: Events description of the second layer automaton

Events	Description
E_{L2}	Enabling controller 2
r_2	Switching rule 2 satisfied
n_2	Necessity to activate controller 2
nn_2	Non necessity to activate controller 2
A_{c2}	Activate controller 2
E_{L3}	Enable layer 3
t	Time τ achieved
t_2	Time τ_2 achieved

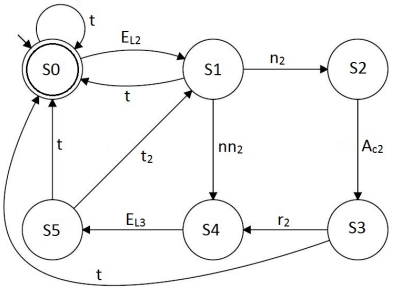

Figure 3: Second layer automaton

The second automaton is started enabling layer 2 and then checks if its controller activation is required. If there exists this requirement (event n_2), controller 2 is activated (event A_{c2}), otherwise (event nn_2), the system enables layer 3 (event E_{L3}), which is also enabled when controller 2 is activated, but the switching rule for the second behavior is being satisfied. As in the highest level automaton, the third layer is enabled only

during a certain time τ_2 (event t_2) or until the time τ of the first behavior be achieved.

This approach considers that only one controller can be active, so activation of a controller means switch for it, deactivating all the others.

The design of all other layers will follow exactly the same model used for the second one, just increasing some events to create a submission in relation to all higher priority behaviors. The automaton model to design any layer i is presented in Figure 4.

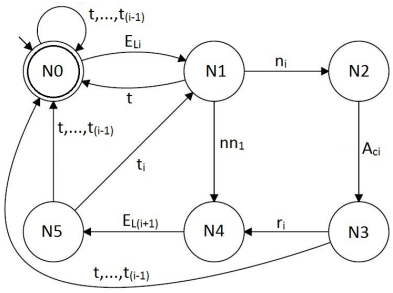

Figure 4: Automaton model for any behavior i except the highest priority one

Once all layers are designed, the overall behavior of the system is generated through parallel composition operation. An example with four layers is presented in Figure 5 to show how hard would be design the whole model without doing it by parts. In this automaton is even difficult to identify the four behavior layers, what shows the importance of using the DES formalism.

The automata analysis in terms of accessibility and co-accessibility was done on software *Supremica*(Akesson et al., 2006) with a composition up to ten behavior layers and no deadlocks or livelocks were found. A deadlock occur when the automaton reaches a state where no further event can be executed and a livelock occurs when there is a set of states reachable from one another, but with no transition going out of the set (Cassandras and Lafortune, 2009). Hence, as a temporal switching rule is used, the system always return to the main controller and possibly achieve its objective (marked state B3 on automaton B indicates the main objective achievement).

3.2 Developed behaviors to methodology validation

In order to test the proposed methodology, three behaviors were developed to the robot: move to goal point, avoid contact with obstacles and capture some artificial marks on the environment, organized in this priority order.

For both move to goal point and capture artificial marks behaviors, the goal position controller proposed by (Secchi, 1998) is used, a globally asymptotically stable controller which makes

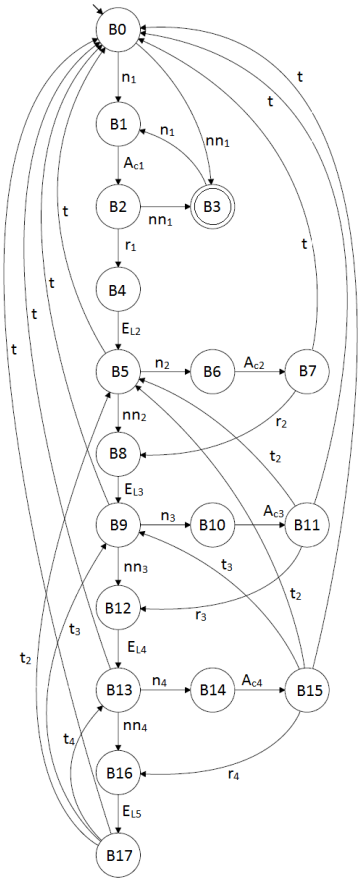

Figure 5: Automaton B: Overall behavior automaton with four behavior layers.

the robot goes from its position to a desired one based on linear (V) and angular (W) velocities.

$$V = k_v tanh(\rho) cos(\alpha) \quad (1)$$

$$W = k_w \alpha + k_v \frac{tanh(\rho)}{\rho} tanh(\rho) cos(\alpha) \quad (2)$$

where kv and kw are positive constants, ρ is the distance between the robot and the goal point and α is the robot orientation error in relation to the goal position.

For the avoid contact with obstacles behavior, the controller proposed by (Freire et al., 2004) is used, which consists in keep linear velocity constant and control angular velocity based on a virtual point created opposite to the nearest obstacle found.

$$V = k_1 \quad (3)$$

$$W = k_{w2} \alpha' + k_{v2} \frac{tanh(\rho')}{\rho'} tanh(\rho') cos(\alpha') \quad (4)$$

where k_1, kw_2 and kv_2 are a positive constants, α' is the robot orientation error in relation to the virtual point, ρ' is a fixed distance between the robot and the virtual point.

Based on these behaviors and their respective controllers, experiments were performed in order validate the methodology. Implementation details and results are presented in section 4.

4 Results

Experiments were performed with a *pioneer 3-DX* robot in different environments and the main objective was to move the robot from an initial point to a goal point. However, obstacles were placed on the environment and the robot should avoid them since this process does not make the robot deviate too much from the main objective. Furthermore, some artificial marks were spread out on the environment before the experiments starts and it was considered the robot could detect marks in a distance less than 200cm and should get them if it does not make the robot collide with obstacles or go too far from the goal point.

Position and orientation of the robot are measured based on *pioneer 3-DX*'s odometry and obstacles are detected based on its ultrasonic sensors. Moreover, no previous knowledge about the environment was used in the system besides the goal point and artificial marks coordinates, featuring a purely reactive navigation.

In relation to enable or not lower layers, the proposed methodology analysis the distance from the robot to the goal position and the angle between them to the highest priority behavior. So, the switching for other controllers is allowed during a time $\tau = 4s$, if this distance and this angle decreased in relation to the last time a switched occurred. The value of τ was arbitrarily chosen.

For the second layer, third layer is enabled during a time $\tau_2 = 1s$ if the robot is in distance higher than 70 cm in relation to the obstacle.

Considering these attributions, three experiments are presented in order to illustrate the proposed methodology.

The first experiment was performed on a simple environment with one box as obstacle, and four marks represented by red and green points. The result, based on pioneer odometry, is presented in Figure 6a and it is possible to see that both first and second objective were accomplished successfully, as long as the robot achieved the goal point with no collision. The third objective was partially accomplished because one mark was too far from the robot, so it was not detected, and the other was too close to the obstacle, so priority was given to avoid collision.

For the second experiment, a similar environment was used, but with lots of artificial marks placed in order to persuade the robot to move away from the goal point. The result presented in Figure 6b shows the robot moving away to the main objective to get the marks. However, when it goes too far, the switching rule forces the main behavior to be active and the robot moves to the goal point.

For the third experiment, obstacles were arranged to test a situation where avoiding obstacles behavior leads the robot too far to the goal point. In this case, the time $\tau = 4s$ gave enough flexibility to the system to overcome the obstacle and accomplish both first and second objectives, as it is presented in Figure 6c. This time could be increased or decreased depending on the expectation of obstacle challenge and the desired flexibility in terms of main objective deviation.

Obstacles including local minima were not used on the experiments as long as it would require a more robust controller, which is not this research focus. The controllers switching along all experiments are presented in Figure 7.

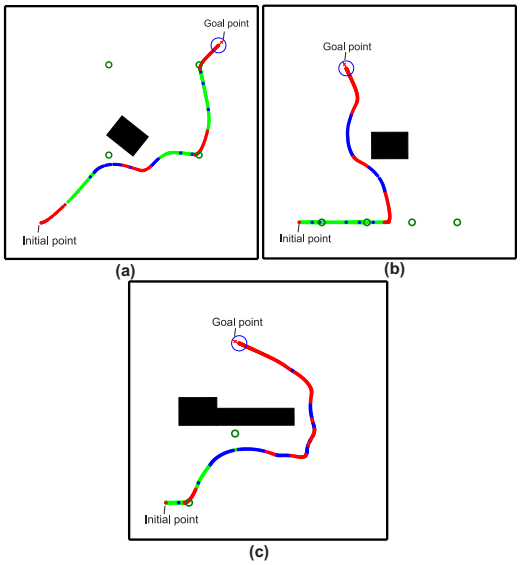

Figure 6: Robot navigation for the 3 experiments. a) Experiment 1. b) Experiment 2. c) Experiment 3. Different colors were used to indicate which behavior was active: red - go to goal position, blue - avoid obstacles and green - capture marks.

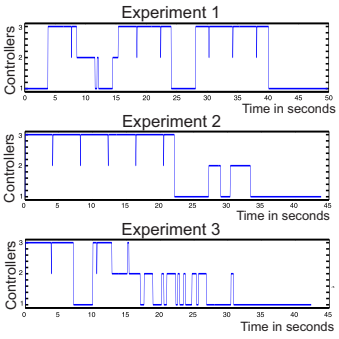

Figure 7: Controllers switching during the experiments. In this graphic controller 1 is responsible for the main objective, controller 2 to avoid obstacles and 3 to capture marks.

5 Conclusion

In this paper a methodology to implement subsumption architecture through discrete event systems formalism is proposed and experimentally

tested. This theory enabled modeling different behaviors and their interaction in a simple and formal way, since the overall behavior was modeled by parts. Furthermore, two automata models were developed, one to design the highest priority layer and the other to design all the others, even the ones which will still be added to the system.

Addition of new behaviors is an important feature of the proposed methodology, as long as the overall behavior is generated through parallel automata composition between the layers. Hence, adding a layer on the system just means to perform this operation, with no need to change the behaviors previously modeled.

Results also showed the switching rule used to enable or not activation of lower layers only during an arbitrary time τ as a feasible solution on which the designer can choose how long the high priority goals can be disturbed by lower ones.

Furthermore, the use of a formalism to implement the finite state machines allowed an analysis of system correctness with no deadlocks or livelocks found on the experiments performed on software *Supremica*. This analysis may be extended in future works to prove, theoretically, that the composition will never present deadlocks or livelocks for an arbitrary number of layers. The development of a method to choose the best value τ will also be investigated.

6 Acknowledgment

The authors would like to thanks CAPES for the financial support.

References

Akesson, K., Fabian, M., Flordal, H. and Malik, R. (2006). Supremica-an integrated environment for verification, synthesis and simulation of discrete event systems, *Discrete Event Systems, 2006 8th International Workshop on*, IEEE, pp. 384–385.

Amir, E. and Maynard-Zhang, P. (2004). Logic-based subsumption architecture, *Artificial Intelligence* **153**(1): 167–237.

Antonelli, G., Arrichiello, F. and Chiaverini, S. (2008). The null-space-based behavioral control for autonomous robotic systems, *Intelligent Service Robotics* **1**(1): 27–39.

Arkin, R. (1998). Behavior-based robotics, . *Intelligent Robotics and Autonomous Agents, v. I. MIT Press* .

Baader, F., Küsters, R. and Molitor, R. (1998). Structural subsumption considered from an automata-theoretic point of view, *Proceedings of the 1998 International Workshop on Description Logics (DL'98*, Citeseer.

Brooks, R. (1986). A robust layered control system for a mobile robot, *IEEE Jrn. Robotics and Automation, Vol.RA-2 No.1, p14-23* .

Cassandras, C. G. and Lafortune, S. (2009). *Introduction to discrete event systems*, Springer Science & Business Media.

Connell, J. H. (1992). Sss: A hybrid architecture applied to robot navigation, *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, IEEE, pp. 2719–2724.

Freire, E., Bastos-Filho, T., Sarcinelli-Filho, M. and Carelli, R. (2004). A new mobile robot control approach via fusion of control signals, *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* **34**(1): 419–429.

Godin, M., Bellingham, J., Kieft, B. and McEwen, R. (2010). Scripting language for state configured layered control of the tethys long range autonomous underwater vehicle, *OCEANS 2010*, IEEE, pp. 1–7.

Nagata, F., Otsuka, A., Watanabe, K. and Habib, M. K. (2013). Multiple mobile robots system with network-based subsumption architecture, *International Journal of Mechatronics and Manufacturing Systems* **6**(1): 57–71.

Ranganathan, A. and Koenig, S. (2003). A reactive robot architecture with planning on demand, *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, Vol. 2, IEEE, pp. 1462–1468.

Russell, S. J., Norvig, P., Canny, J. F., Malik, J. M. and Edwards, D. D. (2003). *Artificial intelligence: a modern approach*, Vol. 2, Prentice hall Upper Saddle River.

Secchi, H. (1998). Control de vehículos autoguiados con realimentación sensorial, *Control de Vehículos Autoguiados con Realimentación Sensorial* .

Siegwart, R. and Nourbakhsh, I. (2004). *Introduction to Autonomous Mobile Robots*.

Turner, J. T., Givigi, S. N. and Beaulieu, A. (2013). Implementation of a subsumption based architecture using model-driven development, *Systems Conference (SysCon), 2013 IEEE International*, IEEE, pp. 331–338.

Yongjie, Y., Qidan, Z. and Chengtao, C. (2006). Hybrid control architecture of mobile robot based on subsumption architecture, *Mechatronics and Automation, Proceedings of the 2006 IEEE International Conference on*, IEEE, pp. 2168–2172.