Coordinating a Robotics Multi-Agent System Using Behavior Trees

Lucas Haug
lucas.haug@usp.br
Universidade de São Paulo
São Paulo, São Paulo, Brazil

Anarosa Alves Franco Brandão
anarosa.brandao@usp.br
Universidade de São Paulo
São Paulo, São Paulo, Brazil

Arthur Casals
arthur.casals@usp.br
Universidade de São Paulo
São Paulo, São Paulo, Brazil

ABSTRACT

The application of multi-agent systems in robotics is a very challenging field. Several competitions involving such systems are proposed to foster research and development of strategies and mechanisms using games as the underlying domain. Among them are the ones from the *IEEE Very Small Soccer (VSSS)* category, which is the case study of this paper. In VSSS, two teams of three robots each compete in a very dynamic environment of a soccer game, to see which team scores the most goals. This work will focus on the ThunderVolt project, a team of the VSSS category from the ThundeRatz robotics team at the University of São Paulo. In this context, this paper discusses control architectures for coordinating multi-agent systems in robotics, presenting a coordination method using Behavior Trees (BTs) developed for ThunderVolt and comparing this method with a previous one used by the team, which was based on a Finite State Machine (FSM). This coordination strategy has as its most significant challenge the assignment roles and the dynamic change of roles between agents, using the reactivity of BTs to achieve a good result. The performance of this technique was compared to the same system using FSM, and its effectiveness was further evaluated in an academic robotics competition, with the results and implications discussed in detail.

CCS CONCEPTS

• **Computer systems organization → Robotic autonomy**; • **Software and its engineering → Cooperating communicating processes**; • **Computing methodologies → Distributed artificial intelligence**; **Multi-agent planning**; **Robotic planning**.

KEYWORDS

Behavior Trees, Coordination Methods, Finite State Machines, Multi-Agent System, Multi-Robot System, IEEE Very Small Size Soccer, Control Architectures.

ACM Reference Format:
Lucas Haug, Anarosa Alves Franco Brandão, and Arthur Casals. 2024. Coordinating a Robotics Multi-Agent System Using Behavior Trees. In *Proceedings of ACM SAC Conference (SAC'24).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/xx.xxx/xxx_x

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SAC'24, April 8 –April 12, 2024, Avila, Spain
© 2024 Association for Computing Machinery.
ACM ISBN 979-8-4007-0243-3/24/04…$15.00
https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

One of the biggest challenges in multi-agent systems is providing a way to coordinate multiple agents to achieve a common goal. These challenges encompass both the coordination logic, as the ones described in [2], and the control architecture (e.g., Finite State Machines, Behavior Trees, Decision Trees) used to implement the defined logic, which depend on the architecture of the system being coordinated. To successfully coordinate agents, it is crucial to have a well-defined coordination logic and an appropriate control architecture that is compatible with the system architecture.

In this regard, the ThundeRatz [21] robotics team at the Universidade de São Paulo developed a project, called ThunderVolt [22], to participate in academic competitions of the *IEEE Very Small Size Soccer (VSSS)* [17] category, which has a great challenge in terms of coordinating multiple robots. In this category, three robots, the size of a cube of 75 mm on each side, play soccer against three other robots, each game lasting ten minutes. Then, to control their robots, each team uses a camera located at the top of the field and connected to a central team computer, which performs image processing to determine the states of the robots, and uses these states to define what each robot should do based on a particular coordination strategy, sending commands to the robots accordingly.

In the ThunderVolt project, to coordinate which robot should play which role in the game, the control logic was first defined based on the system architecture specified by the category. In this way, the coordination of robots is entirely based on a central computer that determines the roles of each of the agents in the organization, taking away the possibility of the robots to participate in the coordination effort, since they do not have an overview of the state of the game.

As for the implementation, the project previously used Finite State Machines (FSM) as the control architecture in the robot coordination system. However, with several developments in the project, it was noticed that the use of FSM did not scale well. In fact, such project decision turn the system ended up becoming much more complex to understand, difficult to maintain, and make improvements.

For these reasons, it was necessary to update the control architecture used in the coordination system, so that an improvement in the project could be possible. After some analyses, Behavior Trees [7] were chosen, as they are much more modular, flexible, and easy to understand.

This paper is structured into seven sections. The second section of this paper discusses the related work, describing the relevance of this work in relation to other research. The third section presents the theoretical foundation needed to understand this paper. The fourth section introduces the ThunderVolt team organization, presenting the organization modeling and the previous coordination

strategy. The fifth section details the proposed solution to enhance the team coordination strategy using BTs. The sixth section outlines the evaluation of the proposed solution. Lastly, the seventh section presents the conclusions of this work.

2 RELATED WORK

The focus of this work is the development of a new control strategy for a multi-agent organization using classical artificial intelligence, comparing different control architectures. Given these restrictions, machine learning methods are not used, even though they can be integrated with the control architectures described below, for example, in [8, 12].

There is extensive research on the field of control architectures for robotics systems [3, 4, 6, 9, 11, 15, 16, 25], demonstrating the use of different control architectures and comparing them. Most of the systems in robotics tend to use Finite State Machines (FSMs) and Hierarchical Finite State Machines (HFSMs) due to their simplicity, comprehensibility, maturity, and widespread use [9, 11, 15]. However, Behavior Trees (BTs) have gaining more attention in recent years and are now almost as widely used as FSMs and HFSMs [11].

Even though FSMs and HFSMs are very popular, when implementing more complex applications, it is possible to observe how systems using these architectures scale poorly, complicating their maintenance, as shown in [9, 15, 16]. In contrast, BTs offer expressiveness, modularity, and flexibility, generalizing many other control architectures, such as the Subsumption Architecture, the Teleo-Reactive Paradigm, and Decision Trees [9]. In terms of expressiveness, HFSMs are the most similar control architecture to BTs, offering a powerful and robust architecture. However, as previously mentioned, HFSMs also have their disadvantages. For this reason, BTs were chosen as the control architecture used to implement the new control strategy of the team, enabling greater scalability, modularity, and ease of maintenance.

Nevertheless, in most of these researches, the focus is more on modeling the individual behaviors of isolated intelligent agents, while comparing control architectures. Concerning the use of BTs in multi-agent systems, there are other papers that are more focused on this area, such as [1], from the game development field, and [7, 26], from the robotics fields. Agis, Gottifredi and García [1] propose an extension of BTs for distributed multi-agent coordination in games, introducing a framework that enables agents to react to events and communicate with each other. Meanwhile, Yang and colleagues [26], propose the coordination of swarms of robots by using a BT that formalizes a coordination behavior and applies it to all robots, enabling information exchange among them to perform tasks. Nonetheless, the article does not detail how to handle the task assignment part using a BT. Lastly, Colledanchise and colleagues [7] highlights the benefits of employing BTs in multi-robot systems, demonstrating a system that is composed of two types of threes, one responsible for performing tasks and the other responsible for managing the task assignment. However, it does not address how to handle task switches between the robots.

Therefore, as the ThunderVolt project necessitates continuous and real-time responsiveness, dynamic task assignment, and task switching between robots, it serves as a valuable case study for exploring the use of BTs in multi-agent organizations, extending the research conducted in existing works.

3 BACKGROUND

3.1 Behavior Trees

Behavior Trees are a type of control architecture [9] considered to be very flexible, modular, easy to understand, and maintain. It is defined by a directed rooted tree, which is composed basically of two types of nodes: the `control flow nodes` and the `execution nodes`. Each control node has at least one child in the tree, while the execution nodes are leaves of the tree. So, by gathering various control flow nodes and execution nodes we can set up a tree that describes a behavior.

The execution of a BT is defined by the tick signal generated by the root node. In this way, for every execution step of the tree, the root node sends a signal to its child node indicating that it should execute and the signal propagates through the tree until it reaches a leaf node. Every node that receives a tick returns a status of *success*, *failure*, or *running*.

The control flow nodes, as their name implies, control the flow of the tree execution, defining which node should be executed and when. The four most important control flow nodes are the fallback node, the sequence node, the parallel node, and the decorator node. Meanwhile, the execution nodes define simple behaviors that the agent need to execute, therefore there are two types of execution node: the action nodes and the condition nodes. All these nodes are defined as follows.

The `fallback node` ticks each of its children, from left to right, and returns *success* or *running* for the first of its children that returns *success* or *running*, accordingly. If all of its children return *failure*, then it also returns *failure*.

The `sequence node` ticks each of its children, from left to right, and returns *success* if all of its children return *success*, otherwise, it returns *failure* or *running*.

The `parallel node` ticks all of its children at once and returns *success* if a number M of its N children return *success*, it returns *failure* if $N - M + 1$ of its children return *failure* and it returns *running* otherwise.

The `decorator node` manipulates the results of its child, depending on the policy defined by the user, for example inverting the result of its child node.

The `action node` is a node that executes a command defined by the user. It returns *running* if it is still being executed, *success*, if the action achieved its goal and *failure* otherwise.

The last type of node, the `condition node`, checks a condition defined by the user and returns *success* if the condition is true and returns *failure* otherwise.

Besides all these nodes, another important concept for the design of BTs is that of subtrees. BTs can grow and become very complex and to be able to maintain and develop complex BTs, the powerful concept of subtrees can be used. Subtrees are modular trees that can be included in a bigger tree as a component. When the main tree ticks a subtree, then the root node of the subtree ticks its children.

Finally, one last feature that is important to highlight about BTs is their reactivity. BTs can have sequence and fallback nodes that are either reactive or non-reactive. Non-reactive nodes, also

called nodes with memory, are distinguished from reactive nodes by an asterisk (*) in their representation. Reactive nodes restart the execution of their children every time they are ticked, allowing for greater responsiveness to changes in the environment. On the other hand, non-reactive nodes assume that the statuses of their children that were already executed have not changed and tick the next child that has not been executed yet, allowing for more efficient execution without unnecessary re-execution of nodes.

The representation of all these building blocks of the BTs can be seen in Figure 1.

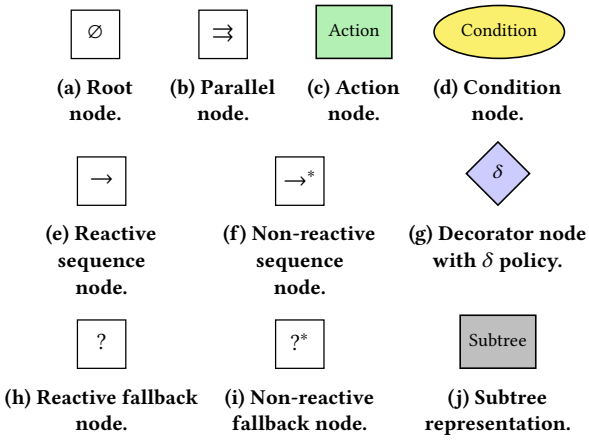

Figure 1: Building blocks of a Behavior Tree.

3.2 $\mathcal{M}$OISE$^+$

To better understand the system, so that a better update could be made, a system modeling was carried out using the $\mathcal{M}OISE^+$ [14] model, a framework developed by researchers from the Universidade de São Paulo together with researchers from the Ecole Nationale Supérieure des Mines de Saint-Etienne.

MOISE (Model of Organization for multI-agent SystEms) model [13] is a framework used in multi-agent systems to design and develop complex and dynamic organizations. It provides a set of concepts, structures, and processes that allow agents to work together in a coordinated and efficient manner. MOISE provides a way of modeling an organization from an organization-centric point of view, where the goal is to join the roles played by the agents with the plans needed to be executed, being divided into three levels:

- *Individual level*: The behavior that each robot should do for each role.
- *Social level*: The relationships between the roles.
- *Collective level*: The aggregation of roles in large structures.

However, there were some shortcomings of MOISE that needed to be handled to improve the framework. For example, to be able to explicitly define global plans for the system in the model. For that reason, an extension of MOISE was developed, which was called $\mathcal{M}OISE^+$ [14]. The $\mathcal{M}OISE^+$ then defines three types of specifications by which it is composed: The Structural Specification, the Functional Specification and the Deontic Specification. These three specifications are needed to be able to define how the MAS organization works so it can reach its goal.

The **Structural Specification** specifies what is the structure of the organization, that is which roles can be played by the agents, what is the relationship between these roles, and the groups of which these roles are part of. Thus, in the Structural Specification, it is possible to define, for example, if one role has authority over another, if they are compatible with each other, how many agents can play a role, and so on.

On the other hand, the **Functional Specification** is used to define how the organization achieves its goals, by defining global plans and using a set of global goals to define missions. The missions are used to form a social scheme for the organization and the agents in the organization can commit to missions following the rules defined in the social scheme of how many agents can commit to a mission.

Finally, the **Deontic Specification** is the one that creates a relationship between the Structural and the Functional Specification. It specifies the permissions and obligations of each role regarding a mission.

In this work, $\mathcal{M}$OISE$^+$ was used to define the coordination logic for the robotics team.

4 TEAM ORGANIZATION

As specified before, the ThunderVolt project is a project implemented by the ThundeRatz robotics team, which competes in the VSSS category. The team's primary objective is to score as many goals as possible against their opponents. To achieve this objective, the members of the ThunderVolt project have defined seven roles that the three robots of the team can play. These roles are designed to improve the team's performance in the dynamic environment of a soccer game. The seven roles are described in more detail below.

- *Goalkeeper*: Defends the goal.
- *Striker*: Carries out attacks against the opposing team.
- *Assistant*: Strategically positions itself dynamically to take advantage of rebounds in the attack state.
- *Fullback*: Helps the goalkeeper to defend the goal
- *Wingback*: Helps the defense, while trying to make counter-attacks.
- *Penalty Kicker*: Kicks penalties in the opposing team.
- *Penalty Defender*: Defends the opposing team's penalties.

As there are more roles than robots it is extremely necessary to have a form of coordinating which robot should play each role. For that, we adopted the $\mathcal{M}OISE^+$ to model how the coordination of the team occurs. This modeling is presented in the next subsection. Regarding the coordination strategy of the team, it is carried out by the central computer used in the category and is the one that was performed by an FSM but became too hard to maintain and improve.

4.1 Modeling

For the modeling, the $\mathcal{M}OISE^+$ framework was used, which was chosen due to its suitability to the application and the authors' familiarity with the model. The structural specification of the team can be seen in Figure 2.

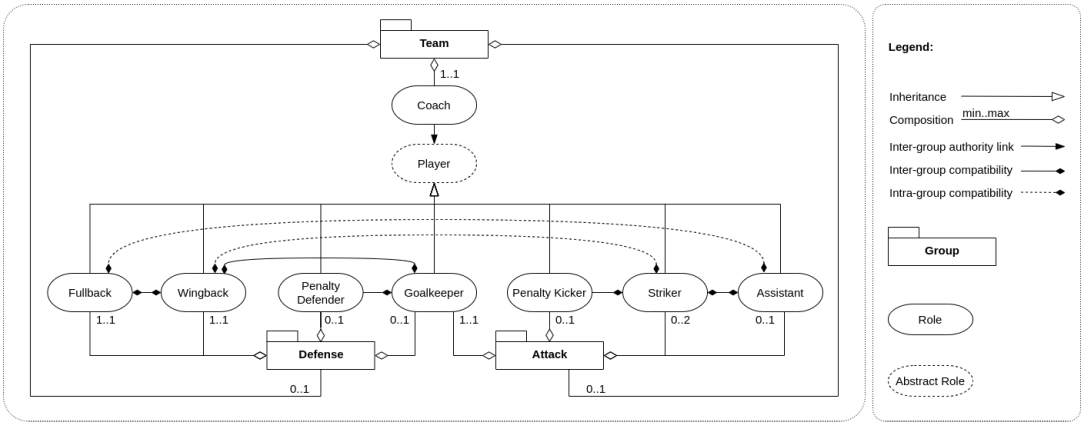

Figure 2: Mapping of the system structural specification using $\mathcal{M}$OISE$^+$.

In the structural specification, it is possible to observe the relationship between the different roles that each robot can play, this is represented by the inter- and intra-group compatibilities. The inter-group compatibilities show the relationships between roles from the same group, which means the relationships between the roles in the defense and the relationships between the roles in the attack. For example, a robot playing the Fullback role can also play the Wingback role. The intra-group compatibilities, on the other hand, show the relationships between the roles from different groups, for example, a robot playing the Wingback role, when changing from the defense state to the attack state will play the Striker role due to their compatibility.

It is also possible to observe how a leader role called **Coach** is defined, having authority over all the other roles. In addition, it is through this role that the coordination strategy was modeled using a FSM. The artifice of authority over other roles was used to have a global way of defining which robot should play which role, delegating that authority to an agent with a more general view of the game and therefore simplifying the system.

In the specification, it is also defined how many robots should play a role in a defined group and how many of each group of roles should be used, which is an important information for developing the coordination strategy of the team. These definitions are specified as the compositions. For example, in the Attack group there will always be a robot playing the Goalkeeper role, however, it is possible to have different combinations of the other roles, e.g. two Strikers or one Striker and one Assistant.

4.2 Previous coordination solution

The previous software solution for robotics team use an FSM, that can be seen in Figure 3. The FSM comprises eight possible states, consisting of three states involving role swaps and five input states. The input state defines the initial configuration of the roles that are being used and the swap states perform the swaps between the roles.

The state in which the state machine initiates is determined beforehand upon receiving an event from an automated referee developed for the category [24], so all the initialization of the strategy is done before starting the FSM. An example would be receiving a penalty event for the opposing team, whereby the entry state would be the penalty defense state and, for this case, the roles that would be used are the Penalty Defender, the Fullback, and the Wingback. Each state transition is governed by a guard function, and when the function's condition is fulfilled, a transition function is invoked. These functions are listed for each transition in Figure 3.

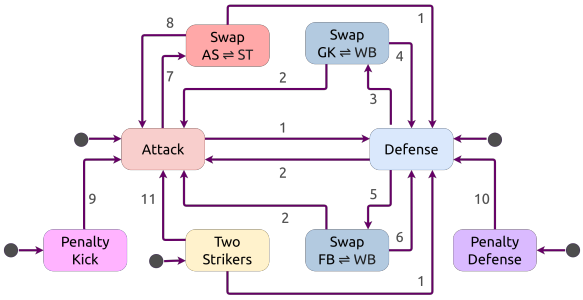

Figure 3: Previous FSM of the coordination strategy.

It is worth noting that, in all cases, to enter a swap state, a swap condition is assessed. Meanwhile, to exit these states and transit back to the attack or defense states, the negation of the swap condition is analyzed. This feature is crucial to ensure that the swap state is exited only when the swap is fully completed and cannot take place immediately afterward, guaranteeing a hysteresis effect

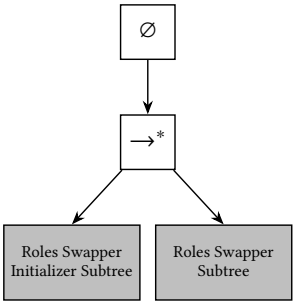

Figure 4: Base structure of the Coach's Behavior Tree.

on the system and avoiding multiple successive swaps that would destabilize the system.

5 PROPOSED COORDINATION SOLUTION USING BEHAVIOR TREES

In the proposed solution, the same $\mathcal{M}OISE^+$ model, as previously presented, is used and its implementation is made using Behavior Trees to improve the system coordination method. As the system already had a leader agent responsible for defining the robots' roles, it was possible to just refactor the agent that used the FSM to use a BT instead.

In this way, modeling the team's coordination behavior is reduced to modeling the Coach's behavior. However, it is important to note that, for this improvement, the goal of the changes was not necessarily to impact at first the system's performance, but just to improve its structure and maintainability.

For the case of the application, a tree with two main branches was developed, as can be seen in Figure 4. The left branch takes care of initializing the roles of each robot after each pause in the game, this branch is responsible for adapting the coordination system to the rules of the category [5], which the system has to obey. Also, it handles messages received from an automatic referee system [24] reporting what happened in the game and depending on what happened, sets different starting roles for each robot. This branch is omitted here for simplification, as it just handles the message events. This initialization subtree replaces the functionality of several FSM initial states, making the strategy more integrated and simpler to understand.

The right branch is responsible for performing the dynamic analysis of role changes and its structure is depicted in Figure 5. It has two sub-branches, one for the moment when the team is attacking (left branch of the fallback in the root) and the other for when it is defending (right branch of the fallback in the root). This subtree is the one responsible for performing the role changes between the roles from the attack group and the ones from the defense group. The idea of the two sub-branches follows a principle very similar to the two states of attack and defense present in the previous FSM-based strategy.

The tracking of the state of the team, whether it is attacking or defending, is done using a blackboard [18], a structure used to share environment variables between nodes. Then, in the *Roles Swapper Initializer* subtree, it is possible to easily set in the blackboard if the team should attack or defend, and in the *Roles Swapper* subtree to swap the state of the team by changing the same variable.

The two sub-branches of the *Roles Swapper* subtree have very similar structures, with the main difference being the subtrees that each branch includes. The attack branch includes the *Attack Swapper* subtree (Figure 6), which handles switching between roles that are part of the attack role group (see Figure 2), and the *Defense*

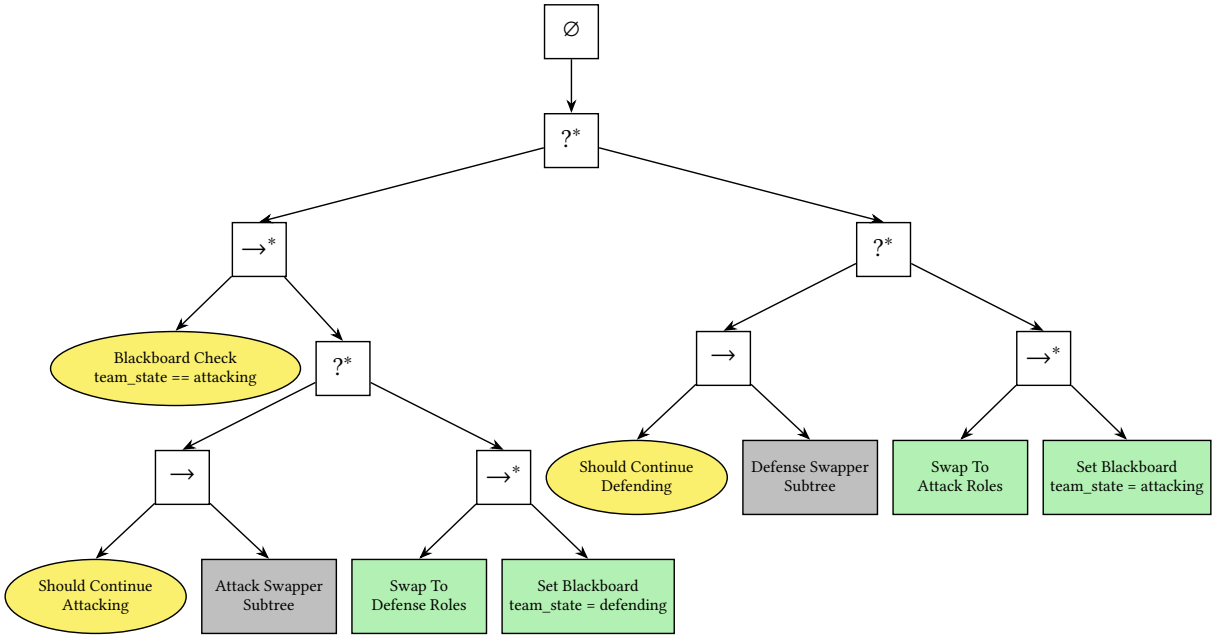

Figure 5: Roles Swapper Subtree.

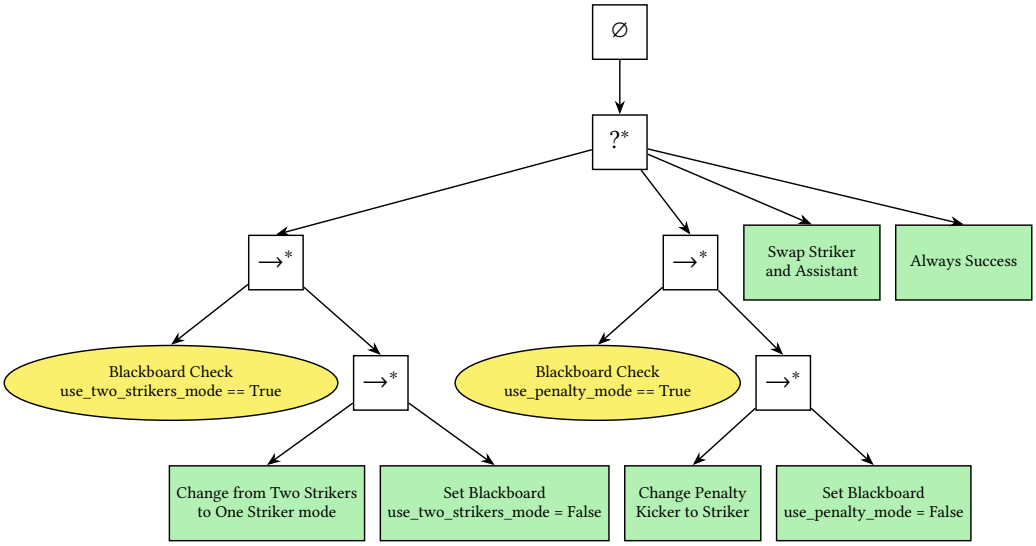

Figure 6: Attack state internal swap subtree.

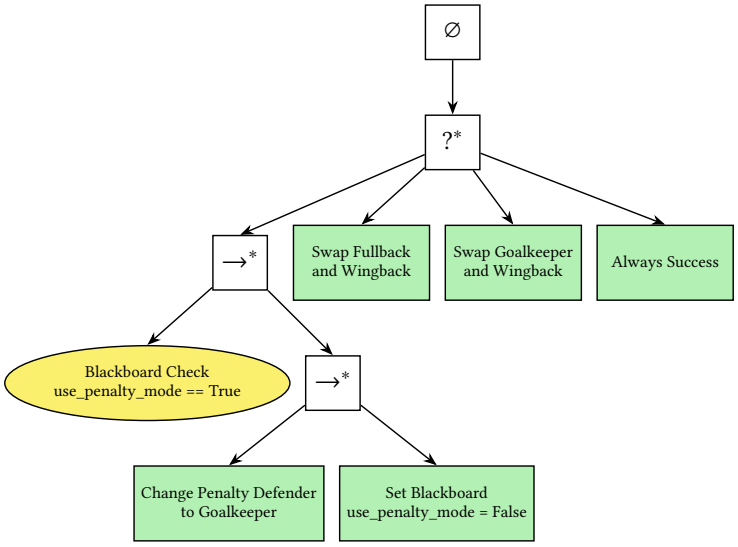

Figure 7: Defense state internal swap subtree.

Swapper subtree (Figure 7) handles switching between roles in the defense group. The *Attack Swapper* and *Defense Swapper* subtrees encompass the functionalities of the FSM's role swaps states.

The *AttackSwapper* subtree first handles when the team is using a mode where two Strikers roles are played by two robots at the same time, then it handles the case where the team is kicking a penalty. Finally it checks if the robot playing the Striker role needs to swap role with the robot playing the Assistant role, and, if nothing needs to be done, it returns success. The *DefenseSwapper* subtree has similar behavior, it first handles the case when the team is defending a penalty kick, then it handles the case of swapping the roles of the Fullback and the Wingback, then the case of swapping the Goalkeeper and the Wingback, and, if nothing needs to be done, it also returns success.

To validate the proposed model for the coordinating agent, the BT was implemented using the *BehaviorTree.CPP* [10] library, which was chosen for its open-source nature, large community, and wide use in different robotics applications.

6 EVALUATION

6.1 Formal comparison

To make a formal comparison between the FSM-based team and the BT-based team, it is necessary to take into account the objectives of the refactoring. The change was mainly aimed at modifying the system's control architecture to improve its maintenance, its

understanding, and its flexibility to change, but without necessarily impacting the team's performance at first.

It is possible to state that the system had its maintainability and flexibility improved since the system became more modular and scalable. The system increased its modularity thanks to the independent nodes that were developed in the tree and thanks to the division of the system into subtrees. Besides that, it became more scalable since to add new functionalities, such as, for example, new roles in the organization, it is not necessary to worry about numerous transitions between states. In fact, in the case of adding new roles it is only necessary to add an extra node that takes care of the swap between the new role and another compatible one.

Regarding the ease of understanding, it is possible to state that the graphical representation of the BT may seem more complex than that of the FSM, but this is because the structure developed for the BT is much more transparent of its functionalities than that of the FSM. A clear example of this fact is the priority between role changes, a characteristic that is not possible to see clearly only with the graphic representation of the FSM, but is quite explicit in the BT.

6.2 Tests between the FSM-based and BT-based teams

The tests were carried out using the FIRASim simulator [23] and the automatic referee of the category. They consisted of 250 games of ten minutes each, totaling 2500 minutes of game. The overall results of the tests are presented in Tables 1 and 2 and in Figure 8.

Table 1: Wins, losses, and ties ratio.

BT-based Team Wins	Ties	FSM-based Team Wins
34.80%	42.40%	22.80%

Table 2: Goals related metrics.

Teams	BT-based	FSM-based
Total goal	193	140
Maximum goals in a game	5	3
GPG - average	0.7720	0.5600
GPG - standard deviation	0.9550	0.7526
GDPG - average	0.2120	-0.2120
GDPG - standard deviation	1.1794	1.1794

Legend: GPG: Goals per game; GDPG: Goals difference per game.

From Table 1, it is possible to note that, although most of the games between the two teams were ties, the BT-based team had a higher percentage of wins than the FSM-based team. Furthermore, Table 2 indicates that the BT-based team had a superior performance regarding the number of goals (193 to 140 - or 35% more efficient considering absolute numbers). In addition, the average number of goals per game was also higher for the BT-based team (0.7720 to 0.5600 - almost 23% higher).

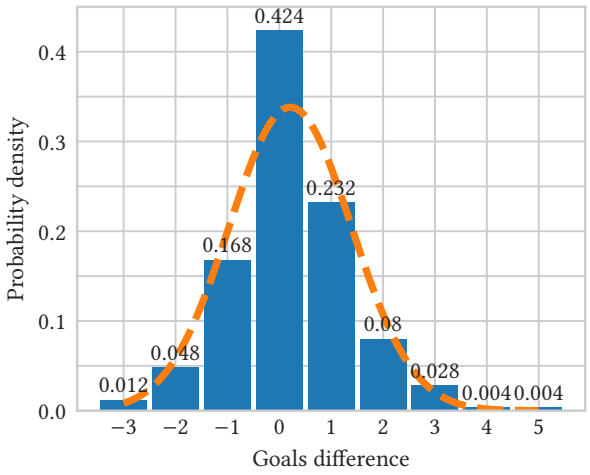

Figure 8: Goals difference histogram: $\mu = 0.2120$, $\sigma = 1.1794$.

Considering the goals difference per game metric, it is possible to perform a one-sample t-test, taking into account the null hypothesis that the two teams have similar performance. As for the null hypothesis the average goals difference per game should be zero, the p-value calculated for this case is 0.00243, which makes it possible to reject the null hypothesis with a significance of 5%, showing that indeed the BT-based team performed better than the FSM-based team.

This performance improvement is probably due to better control over the strategy reactivity and better control of certain characteristics of the strategy, such as the priority between roles changes.

6.3 Robotics Academic Competition

The implementation was also tested in a Brazilian robotics academic competition, the IRONCup 2023 [19]. The competition consisted of a round-robin tournament and was played in the virtual environment of the FIRASim simulator between the following teams: ThunderVolt, RobôCIn[1], Robotbulls[2], ITAndroids[3], Red Dragons[4], Rinobot[5] and Neon[6].

The team obtained gained four in five games (see Table 3. The team won second place in the competition, showing the effectiveness of the coordination structure (see Table 4).

7 CONCLUSION

The application case showed how it is possible to coordinate a multi-agent system with dynamic role assignment using BTs, by defining the behavior of a leader agent that controls the roles in the organization and defining swap nodes that handle each role change.

[1]https://robocin.com.br/
[2]https://inatel.br/robotica/
[3]https://www.itandroids.com.br/en/
[4]https://www.linkedin.com/company/reddragons/
[5]https://www.linkedin.com/company/rinobot-team/
[6]https://projectneon.dev/

Table 3: Results of the games played by the ThunderVoltteam [20].

Blue team				Yellow team
ThunderVolt	7	x	0	Red Dragons
Neon	0	x	14	ThunderVolt
ThunderVolt	4	x	0	ITAndroids
Rinobot	0	x	5	ThunderVolt
RobôCIn	3	x	0	ThunderVolt
Robotbulls	0	x	2	ThunderVolt

Table 4: Final results in order of the competition considering all games [20].

Team	Pts	GP	Vic	Def	GS	GC	GD
RobôCIn	18	6	6	0	68	17	51
ThunderVolt	15	6	5	1	32	3	29
Robotbulls	12	6	4	2	34	23	11
ITAndroids	9	6	3	3	36	17	19
Red Dragons	6	6	2	4	36	36	0
Rinobot	3	6	1	5	14	54	-40
Neon	0	6	0	6	7	46	-39

Legend: Pts: Points; GP: Games Played; Vic: Victories; Def: Defeats; GS: Goals Scored; GC: Goals Conceded; GD: Goal Difference.
Points Count: Each victory counts as three points and each tie counts as one point.

BTs are very easy to implement, they are flexible and modular, which facilitates maintenance, scalability, and system updates. These characteristics prove to be a great solution for coordinating multiple robots, and being a great alternative to FSMs for defining complex behaviors.

In terms of performance, the use of the new BT-based coordination strategy enabled a statistically significant improvement in relation to the FSM-based system. When analyzing this improvement, it is important to consider that the developed BT is not a literal translation of the previous FSM, but a reinterpretation, taking advantage of the characteristics of the BTs. With this in mind, it is possible to state that the team's performance may have improved thanks to better control of the strategy's reactivity, as well as, for example, the better ability to define priorities between role changes in the organization.

ACKNOWLEDGMENTS

We thank the ThunderVolt project members, for their help with the development of the solution and the ThundeRatz robotics team, the Amigos da Poli Patrimonial Fund and the Escola Politécnica from the Universidade de São Paulo for all their support. In addition, the sponsorship of STMicroelectronics, Altium, SolidWorks and Circuibras to the ThunderVolt project.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001.

REFERENCES

[1] Ramiro A. Agis, Sebastian Gottifredi, and Alejandro J. García. 2020. An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games. *Expert Systems with Applications* 155 (2020), 113457. https://doi.org/10.1016/j.eswa.2020.113457

[2] Thomas Ågotnes and Nils Bulling. 2014. Formal Methods for Coordinating Multi-Agent Systems (Dagstuhl Seminar 14332). *Dagstuhl Reports* 4, 8 (2014), 21–44. https://doi.org/10.4230/DagRep.4.8.21

[3] Oliver Biggar, Mohammad Zamani, and Iman Shames. 2021. An Expressiveness Hierarchy of Behavior Trees and Related Architectures. *IEEE Robotics and Automation Letters* 6, 3 (2021), 5397–5404. https://doi.org/10.1109/LRA.2021.3074337

[4] D Billington, V Estivill-Castro, R Hexel, and A Rock. 2010. Plausible logic facilitates engineering the behavior of autonomous robots. In *The IASTED International Conference on Software Engineering*. 41–48.

[5] CBR. 2022. *Regras IEEE Very Small Size Soccer (VSSS) - Série B.* https://www.cbrobotica.org/wp-content/uploads/2022/05/vssRules.pdf

[6] Juan Chen and DianXi Shi. 2018. Development and Composition of Robot Architecture in Dynamic Environment. In *Proceedings of the 2018 International Conference on Robotics, Control and Automation Engineering* (Beijing, China) *(RCAE 2018)*. Association for Computing Machinery, New York, NY, USA, 96–101. https://doi.org/10.1145/3303714.3303716

[7] Michele Colledanchise, Alejandro Marzinotto, Dimos V. Dimarogonas, and Petter Oegren. 2016. The Advantages of Using Behavior Trees in Mult-Robot Systems. In *Proceedings of ISR 2016: 47st International Symposium on Robotics*. 1–8.

[8] Michele Colledanchise, Ramviyas Parasuraman, and Petter Ögren. 2019. Learning of Behavior Trees for Autonomous Agents. *IEEE Transactions on Games* 11, 2 (2019), 183–189. https://doi.org/10.1109/TG.2018.2816806

[9] Michele Colledanchise and Petter Ögren. 2018. *Behavior Trees in Robotics and AI.* CRC Press. https://doi.org/10.1201/9780429489105

[10] Davide Faconti and Michele Colledanchise. 2022. *BehaviourTree.CPP.* https://github.com/BehaviorTree/BehaviorTree.CPP/

[11] Razan Ghzouli, Thorsten Berger, Einar Broch Johnsen, Andrzej Wasowski, and Swaib Dragule. 2023. Behavior Trees and State Machines in Robotics Applications. *IEEE Transactions on Software Engineering* (2023), 1–24. https://doi.org/10.1109/TSE.2023.3269081

[12] Marco Gillies. 2009. Learning Finite-State Machine Controllers From Motion Capture Data. *IEEE Transactions on Computational Intelligence and AI in Games* 1, 1 (2009), 63–72. https://doi.org/10.1109/TCIAIG.2009.2019630

[13] Mahdi Hannoun, Olivier Boissier, Jaime S. Sichman, and Claudette Sayettat. 2000. MOISE: An Organizational Model for Multi-agent Systems. In *Advances in Artificial Intelligence*, Maria Carolina Monard and Jaime Simão Sichman (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 156–165.

[14] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. 2002. A Model for the Structural, Functional, and Deontic Specification of Organizations in Multiagent Systems. In *Advances in Artificial Intelligence*, Guilherme Bittencourt and Geber L. Ramalho (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 118–128.

[15] Matteo Iovino, Julian Förster, Pietro Falco, Jen Jen Chung, Roland Siegwart, and Christian Smith. 2022. On the programming effort required to generate Behavior Trees and Finite State Machines for robotic applications. *arXiv preprint arXiv:2209.07392* (2022). arXiv:2209.07392 [cs.RO] https://arxiv.org/abs/2209.07392

[16] Matteo Iovino, Edvards Scukins, Jonathan Styrud, Petter Ögren, and Christian Smith. 2022. A survey of Behavior Trees in robotics and AI. *Robotics and Autonomous Systems* 154 (2022), 104096. https://doi.org/10.1016/j.robot.2022.104096

[17] Very Small Size League. 2023. *Very Small Size League.* Retrieved 2023-04-07 from https://vssleague.github.io/vss/index.html

[18] Microsoft. 2022. *Blackboard Design Pattern.* https://social.technet.microsoft.com/wiki/contents/articles/13215.blackboard-design-pattern.aspx

[19] RoboCore. 2023. *Inatel Robotics National Cup.* https://events.robocore.net/ironcup-2023/

[20] RoboCore. 2023. *Simulado Iron 2023- Tabela VSSS.* https://docs.google.com/spreadsheets/d/1T5c95daYr8EBxO2axLRloB10ix0VhdRm99B98Gi-76c/edit

[21] ThundeRatz. 2023. *Equipe ThundeRatz de Robótica.* https://thunderatz.org/

[22] ThundeRatz. 2023. *ThundeVolt.* https://thunderatz.org/projects/robots/thundervolt

[23] VSSSLeague. 2022. *FIRASim.* https://github.com/VSSSLeague/FIRASim

[24] VSSSLeague. 2022. *VSSReferee.* https://github.com/VSSSLeague/VSSReferee

[25] F.Y. Wang, K.J. Kyriakopoulos, A. Tsolkas, and G.N. Saridis. 1991. A Petri-net coordination model for an intelligent mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics* 21, 4 (1991), 777–789. https://doi.org/10.1109/21.108296

[26] Qin Yang, Zhiwei Luo, Wenzhan Song, and Ramviyas Parasuraman. 2019. Self-Reactive Planning of Multi-Robots with Dynamic Task Assignments. In *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. 89–91. https://doi.org/10.1109/MRS.2019.8901075