# 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

In Robotics Multi-Agent Systems, agents often engage in interactive task-oriented cooperation to achieve a common goal. Several contests 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 robots compete in a very dynamic environment of a soccer game. In this context, this paper presents a method to use Behavior Trees (BT) to coordinate multiple robots with distinct roles, replacing the use of Finite State Machines for such applications. BTs are often used in games and in robotics to describe individual behaviors of intelligent agents. The proposed approach employs BTs to define the behavior of a group in the context of a team of the referred category. As this category requires a central computer to control the robots, the presented method takes advantage of this architecture. Additionally, the performance of this technique was compared to the same system using FSM and was also evaluated in an academic robotics competition, and the outcomes of the experiment are discussed.

## 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, 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. 2023. Coordinating a Robotics Multi-Agent System Using Behavior Trees. In *Proceedings of 23rd ACM International Conference on Intelligent Virtual Agents (IVA'23).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

## 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, Petri Nets) 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 context, the ThundeRatz [20] robotics team at the *Universidade de São Paulo* developed a project, called ThunderVolt [21], to participate in academic competitions of the *IEEE Very Small Size Soccer (VSSS)* [16] category, which has a great challenge in terms of coordinating multiple robots. In the category, three robots the size of a 75 mm cube on each side play soccer against three other robots, each game lasts ten minutes. Then, to be able to control their robots, each team uses a camera located at the top of the field 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 must do according to a certain coordination strategy, transmitting commands to the robots.

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 robots coordination system. However, with several developments in the project, it was noticed how the use of an FSM did not scale well, 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, BTs were chosen, which, as will be described, are an alternative to the FSMs, as they are much more modular, flexible and easy to understand.

## 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

*IVA'23, September 19-22, 2023, Würzburg, Germany*

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

be integrated with the control architectures described below, for example, in [9, 11].

There is extensive research on the field of control architectures for robotics systems [3, 4, 6, 7, 14, 15, 24], demonstrating the use of different control architectures and comparing them. Most of the systems tend to use FSMs since it is simple to understand and to implement [7, 14], but as shown in [7, 14, 15], FSMs can become very complex and scale poorly, complicating the maintenance of the system. As for other control architectures, an extensive comparison is performed in [7], showing how BTs can generalize most of the control architectures used in robotics, such as the Subsumption Architecture, the Teleo-Reactive Paradigm and Decision Trees, demonstrating how expressive, readable and flexible BTs are. For this reason, BTs were chosen as the control architecture used to implement the control strategy of the team. Nevertheless, in most of these researches, the focus is more on modeling the individual behaviors of intelligent agents, while comparing the control architectures.

Concerning the use of BTs in multi-agent systems, there is some recent research, as can be seen in the article [1], in which an extension of BTs for distributed multi-agent coordination in games was developed. And there are as well advances in the area of robotics [8, 25], showing that it is possible to adapt to use BTs successfully to define the behaviors of agents in a multi-agent system. However, in both scenarios, the BTs are used to help to coordinate the system, but they are used to describe the behavior of individual agents, not the behavior of the organization.

# 3 BACKGROUND

## 3.1 Behavior Trees

Behavior Trees are a type of control architecture [7] 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 then is defined by the tick signal generated by the root node, in this way every execution step of the tree, the root node will send a signal to its child node indicating that it should execute and the signal will propagate through the tree until it reaches a leaf node. Every node that receives a tick will then return a status of *success*, *failure*, or *running*.

The control flow nodes, as their name implies, control the flow of the execution of the three, defining which node will 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 will be 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.

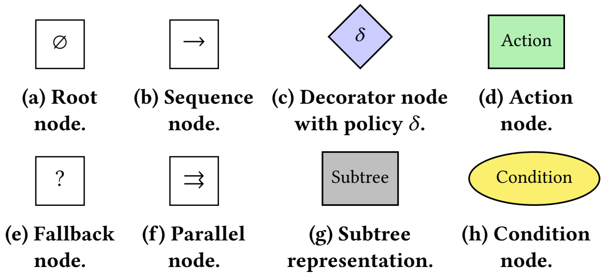
The **decorator node** manipulates the results of its child, depending on the policy defined by the user, for example inverting the result of its child node.

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, an important final 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.

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



**(a) Root node.** **(b) Sequence node.** **(c) Decorator node with policy $\delta$.** **(d) Action node.**

**(e) Fallback node.** **(f) Parallel node.** **(g) Subtree representation.** **(h) Condition node.**

**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 $\mathcal{M}OISE^+$[13] 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 [12] 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, that it is 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 $\mathcal{M}OISE^{+}$[13]. 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. 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 will achieve 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.

# 4 APPLICATION

## 4.1 Description

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 goal is to score as many goals as possible against their opponents. To achieve this goal, the members of the ThunderVolt project have defined seven roles that the three robots on 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. This coordination strategy is carried out by the central computer used in the category and is the one that was performed by a FSM but became too hard to maintain and improve.

## 4.2 Modeling

Before making the defined changes, as there was no formal way of defining the team's organization, to ensure a better understanding of the team and how it could be improved, a modeling of the system's organization was carried out. 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 4 in Appendix A.

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 central role called **Coach** was defined, which has authority over all the other roles and 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.

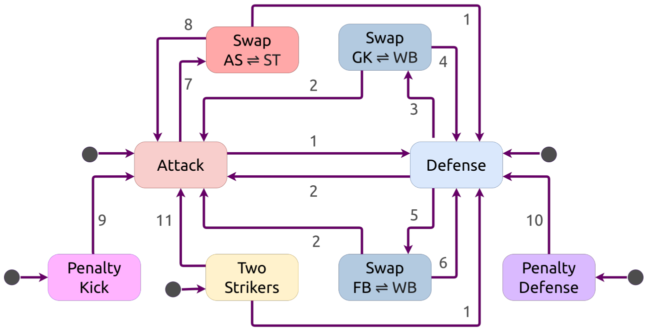
In the specification, it is also defined how many robots should play a role in a defined group and how many of each group of roles should be used, which is an important information for developing the coordination strategy of the team. These definitions are specified as the compositions, for example, 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.

## 4.3 Previous FSM

The previous solution using a FSM can be seen in Figure 2. 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 realize 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 [23], 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 2.

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



Legend
GK ⇨ Goalkeeper FB ⇨ Fullback WB ⇨ Wingback
AS ⇨ Assistant ST ⇨ Striker

Guard function/Transition function
1. attack_to_defense_condition() / attack_to_defense_transition()
2. defense_to_attack_condition() / defense_to_attack_transition()
3. goalkeeper_wingback_condition() / goalkeeper_wingback_transition()
4. not goalkeeper_wingback_condition() / void()
5. fullback_wingback_condition() / fullback_wingback_transition()
6. not fullback_wingback_condition() / void()
7. assistant_striker_condition() / assistant_striker_transition()
8. not assistant_striker_condition() / void()
9. penalty_kick_to_attack_condition() / penalty_kick_to_attack_transition()
10. penalty_defense_to_defense_condition() / penalty_defense_to_defense_transition()
11. two_strikers_to_attack_condition() / two_strikers_to_attack_transition()

**Figure 2: Previous FSM of the coordination strategy.**

## 5 BEHAVIOR TREE IMPLEMENTATION

To improve the coordination method, as the system already had a central agent responsible for defining the robots' roles, it was possible to just refactor the agent that used the FSM to use a BT. In this way, modeling the team's coordination behavior is reduced to modeling the Coach's behavior. However, it is important to notice 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 3. The first 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. This branch handles messages received from an automatic judge [23] 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.

The second branch is responsible for performing the dynamic analysis of role changes. This second branch 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), as can be seen in Figure 5 in Appendix A.

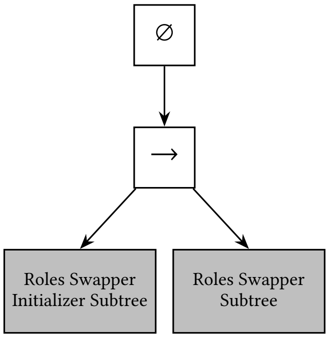


**Figure 3: Base structure of the Coach's Behavior Tree.**

The two sub-branches have similar structures, with the main difference being the subtrees that each branch includes. The attack branch includes the *AttackSwapper* subtree (Figure 6 in Appendix A), which handles switching between roles that are part of the attack role group (see Figure 4), and the *DefenseSwapper* subtree (Figure 7 in Appendix A) handles switching between roles in the defense group.

The tracking of the state of the team, whether it is attacking or defending, is done using a blackboard [17], 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 *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 EXPERIMENTS

### 6.1 Comparison between FSM and BT

To make the comparison between FSM and BT as control architectures 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 refinements, but without necessarily impacting the team's performance at first.

To test the system that was using a FSM with the system using the BT described in the previous section, the FIRASim [22] simulator was used, a simulator develop to run games in virtual competitions of the category. Twelve games of ten minutes each were played between the team using the FSM to coordinate the robots and the team using the BT, the results of the games can be seen in Table 1.

**Table 1: Results of the games played between the version using FSM and the version using BT.**

| | Team with FSM | Team with BT |
|---|---|---|
| Game 1 | 0 | 1[1] |
| Game 2 | 0 | 0 |
| Game 3 | 0 | 0 |
| Game 4 | 1 | 1 |
| Game 5 | 0 | 0 |
| Game 6 | 0 | 1 |
| Game 7 | 0 | 0 |
| Game 8 | 0 | 2[1] |
| Game 9 | 0 | 0 |
| Game 10 | 0 | 1 |
| Game 11 | 0 | 0 |
| Game 12 | 1 | 1[1] |

[1] One goal scored by penalty

Overall, the team using BT scored more goals in this test round, however, it is important to note that several of the goals were scored by penalty kicks. In another round of tests with only penalty shots, neither team managed to score a single goal against the other team, which shows the non-deterministic character of the simulation as a whole.

Therefore, it is not possible to conclude that the switch to BT had a positive or negative impact on the team's performance. The results of the games show very similar results for both teams with their different control architectures.

## 6.2 Robotics Academic Competition

The implementation was also tested in a Brazilian robotics academic competition, the IRONCup 2023 [18]. The competition was played in the virtual environment of the FIRASim simulator [22] between the following teams: ThunderVolt, RobôCIn[1], Robotbulls, ITAndroids[2], Red Dragons[3], Rinobot[4] and Neon[5].

The team obtained the results in the games presented in Table 2. The overall result of the competition can be seen in Table 3. The team won second place in the competition, showing the effectiveness of the coordination structure and the rules for changing roles implemented.

**Table 2: Results of the games played by the ThunderVolt team [19].**

| 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 3: Final results in order of the competition considering all games [19].**

| 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.

## 7 CONCLUSION

The application case showed how it is possible to define a specification of an organization for a MAS, being able to coordinate the different agents of the organization, by defining the behavior of the system with a BT in a scenario where it is needed to have a central agent.

BTs are very easy to implement, they are flexible and modular, which facilitates maintenance, scalability and system updates, proving to be a great solution for coordinating multiple robots, when defining a group behavior.

The use of BTs in replacement of FSMs did not affect the system performance and was able to improve the system architecture, by improving its maintainability, flexibility and readability.

## ACKNOWLEDGMENTS

We thank the members of the ThunderVolt project for their help with the development of the solution and the ThundeRatz Robotics Team, the Amigos da Poli Patrimonial Fund and the Polytechnic School of the University of São Paulo for all their support. In addition to giving recognition to the sponsors of the ThunderVolt Project, the companies STMicroelectronics, Altium, SolidWorks and Circuibras.

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

# 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.* Retrieved April 07, 2023 from 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. 2017. *Behavior Trees in Robotics.* Ph. D. Dissertation. School of Computer Science and Communication - KTH, SE-100 44 Stockholm, Sweden.

[8] 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.

[9] 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

[10] Davide Faconti. 2022. *BehaviourTree.CPP.* https://www.behaviortree.dev/docs/3.8/intro

[11] 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

[12] 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.

[13] 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.

[14] 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).

[15] 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

[16] Very Small Size League. 2023. *Very Small Size League.* Retrieved April 07, 2023 from https://vsssleague.github.io/vss/index.html

[17] Microsoft. 2022. *Blackboard Design Pattern.* Retrieved April 10, 2023 from https://social.technet.microsoft.com/wiki/contents/articles/13215.blackboard-design-pattern.aspx

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

[19] RoboCore. 2023. *Simulado Iron 2023- Tabela VSSS.* Retrieved April 08, 2023 from https://docs.google.com/spreadsheets/d/1T5c95daYr8EBxO2axLRloB10ix0VhdRm99B98Gi-76c/edit

[20] ThundeRatz. 2023. *Equipe ThundeRatz de Robótica.* Retrieved April 07, 2023 from https://thunderatz.org/

[21] ThundeRatz. 2023. *ThundeVolt.* Retrieved April 07, 2023 from https://thunderatz.org/projects/robots/thundervolt

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

[23] VSSSLeague. 2023. *VSSReferee.* https://github.com/VSSSLeague/VSSReferee

[24] 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

[25] 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

# A IMAGES

In this section it is possible to see the structural specification of the organization of the ThunderVolt team (Figure 4), the *RolesSwapper* subtree (Figure 5), the *AttackSwapper* subtree (Figure 6) and the *DefenseSwapper* subtree (Figure 7).

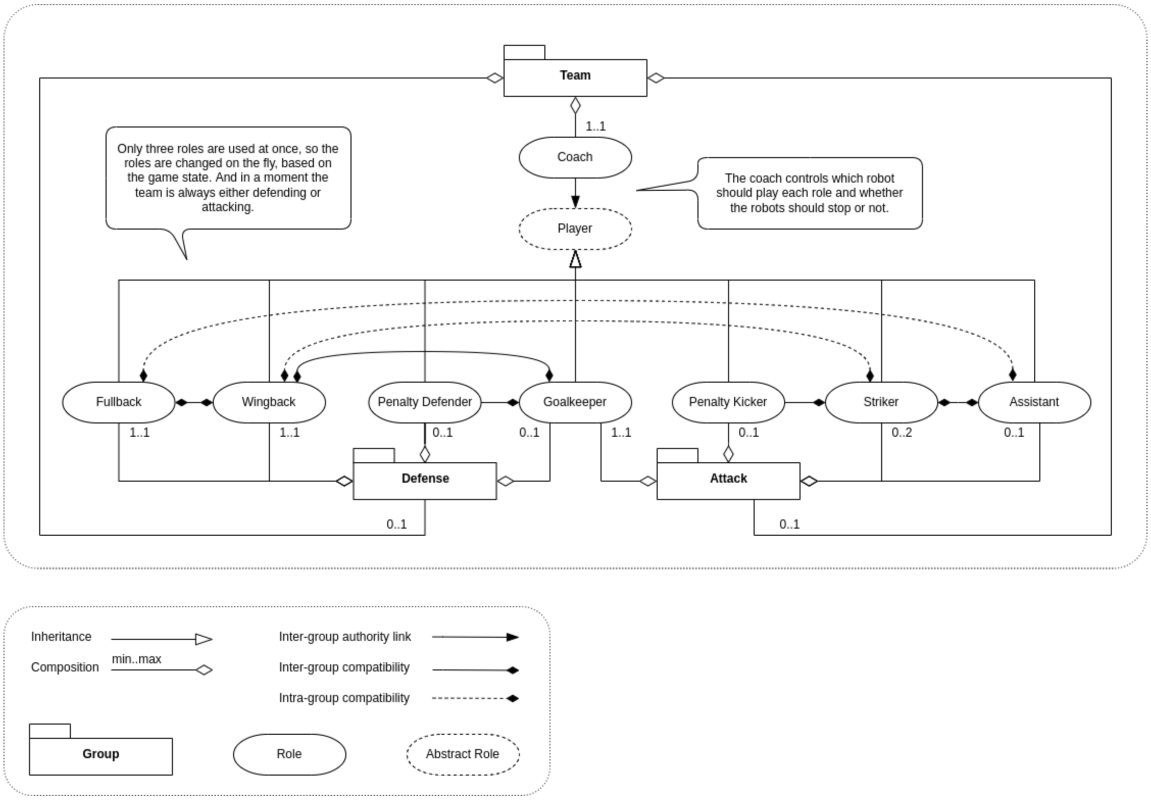

Team
1..1
Coach
Only three roles are used at once, so the roles are changed on the fly, based on the game state. And in a moment the team is always either defending or attacking.
The coach controls which robot should play each role and whether the robots should stop or not.
Player
Fullback
Wingback
Penalty Defender
Goalkeeper
Penalty Kicker
Striker
Assistant
1..1
1..1
0..1
0..1
1..1
0..1
0..2
0..1
Defense
Attack
0..1
0..1
Inheritance
Composition
min..max
Inter-group authority link
Inter-group compatibility
Intra-group compatibility
Group
Role
Abstract Role

**Figure 4: Mapping of the system structural specification using $\mathcal{M}$OISE$^+$.**

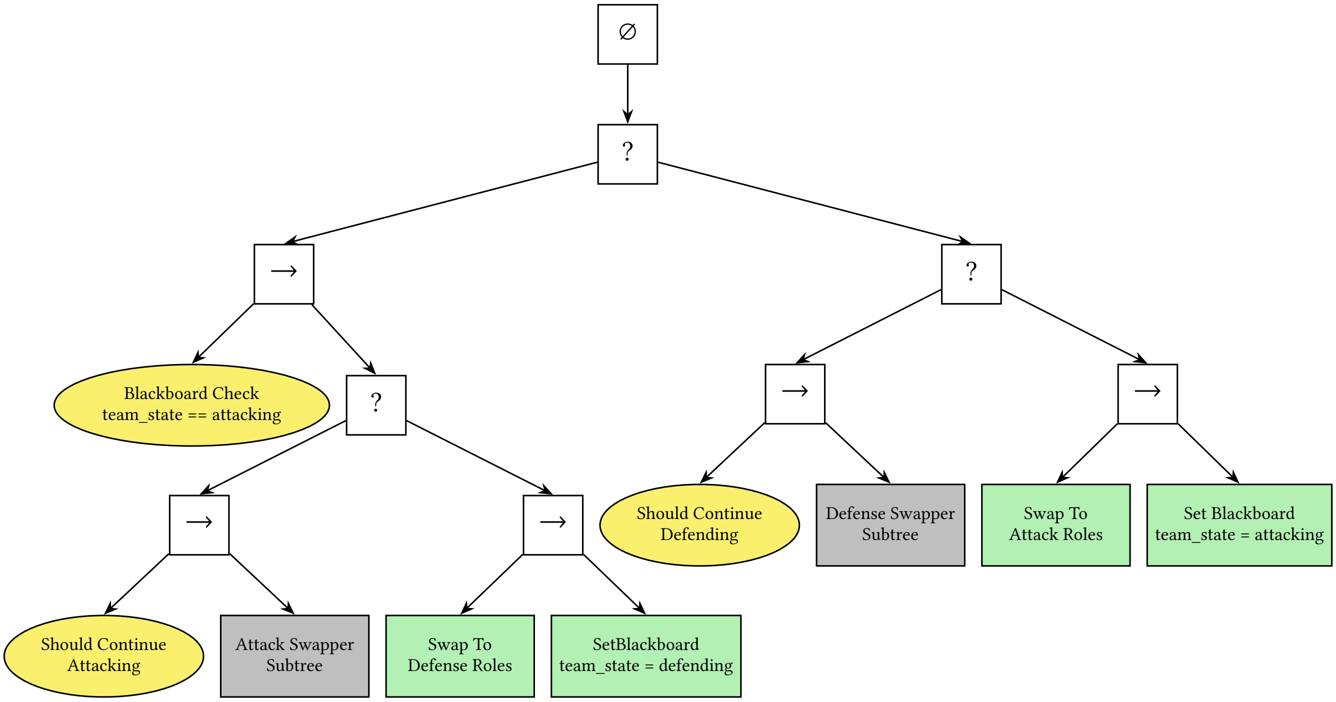


**Figure 5: Roles Swapper Subtree.**

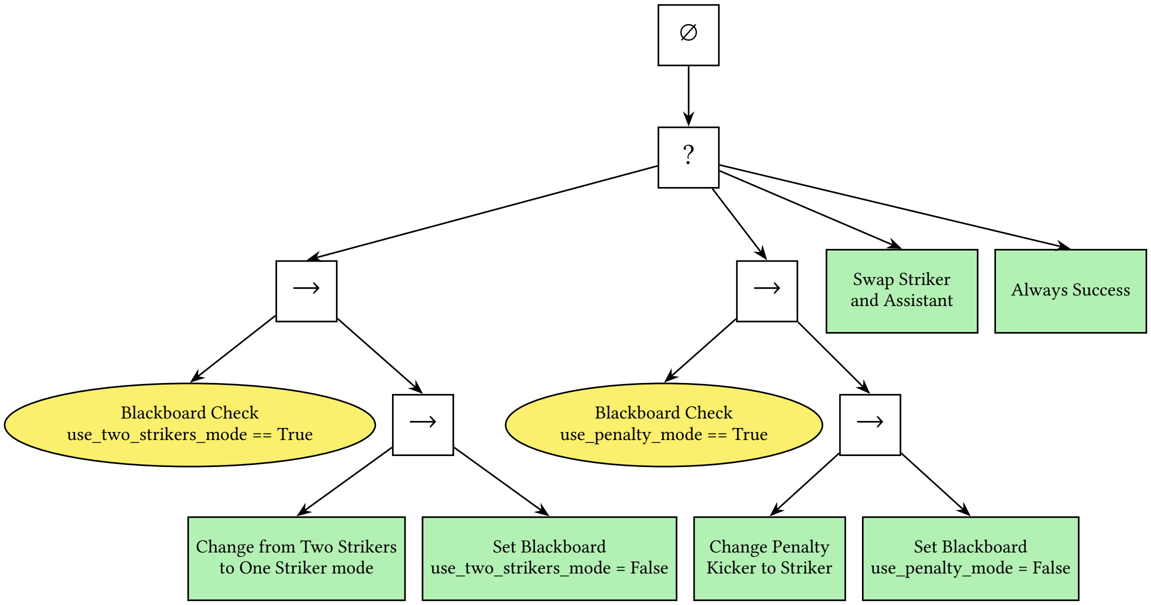

∅
?
→
→
Swap Striker and Assistant
Always Success
Blackboard Check use_two_strikers_mode == True
→
Blackboard Check use_penalty_mode == True
→
Change from Two Strikers to One Striker mode
Set Blackboard use_two_strikers_mode = False
Change Penalty Kicker to Striker
Set Blackboard use_penalty_mode = False

**Figure 6: Attack state internal swap subtree.**

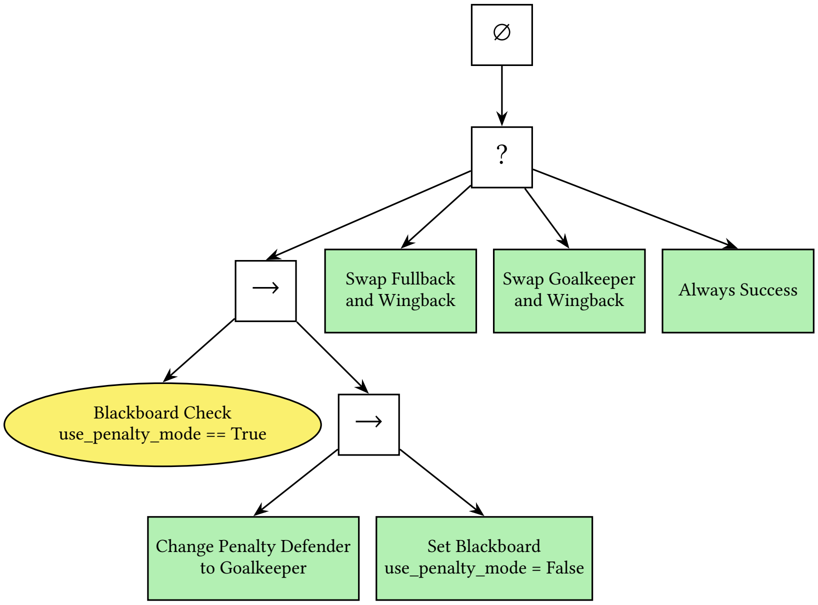

∅
?
→
Swap Fullback and Wingback
Swap Goalkeeper and Wingback
Always Success
Blackboard Check use_penalty_mode == True
→
Change Penalty Defender to Goalkeeper
Set Blackboard use_penalty_mode = False

**Figure 7: Defense state internal swap subtree.**