Automatic Code Generation for new Behaviors in a Tool for Set-plays Design

Raoni Sales¹, Ana Patricia Magalhães², Marco A. C. Simões³, Josemar R. de Souza⁴
raoni_si@hotmail.com¹; anapatriciamagalhaes@gmail.com²; msimoes@uneb.br³; josemar@uneb.br⁴
Universidade Salvador (UNIFACS)¹-², Universidade do Estado da Bahia (UNE)³-⁴

Abstract— Coordinated plans in Multi-Agent Systems (MASs) are essential for the success of the system. In Multi-Robot Systems (MRSs), collective sports consist of an important testbed for MASs. In special, robotic soccer is a well-known challenge for researchers in MRSs. Set-plays are collective plays that correspond to coordinated plans in the robotic soccer domain. Some tools for designing new set plays were developed. This article proposes a solution to support the implementation of new behaviors for simulation and execution of cooperative MASs. The manual development of these behaviors requires prior knowledge of the source codes of these tools, and a high development effort. Our solution uses Model-Driven Development (MDD), which differs from a low-level manual programming paradigm, enables the definition of a behavior in a high abstraction level and subsequent (semi) automatic transformation of it in the source code for specific tools. The solution uses templates from the target platforms, and automatically accesses tools of simulation and execution of a cooperative MAS, updating its files, inserting the codes of new behavior. Experiments in a real soccer team of robots demonstrated their feasibility in generating source code for the FCPortugal SPI planner and Framework Tools for creating set-plays in robotic soccer.

I. INTRODUCTION

Multi-Agent Systems (MASs) are a branch of Artificial Intelligence (AI) where agents perform distributed tasks in a cooperative system [1]. In this way, agents can make plans together as well as inferences establishing mutual communication [2]. Cooperative Systems can be simulated to anticipate how the agents will react in the real world. Simulation is a technique that aims to imitate the execution of a system to observe and evaluate its behavior [4]. Therefore, it is common in MASs development the use of simulation tools to anticipate the learning level and the performance of cognitive agent actions. In Section 3 we present existing tools related to the purpose of this work.

Simulation tools can be observed in MAS of different domains [5-8]. Turtle Simulator [13], SimRobot [14] and SPI planner [12] are examples of simulation tools used in the domain of robotic soccer. What differentiates each of them is how they represent the model of the world, which can vary according to the objective that the designer wants to achieve with the simulation. SPI planner, for example, is a tool that models and simulates robotic soccer set-plays. In SPI planner, each team member is defined as an agent whose behavior must be coordinated with the other team members [12]. This coordination of multiple agents acting in favor of a common goal is desirable in several areas of MAS application, e.g., the collaborative movement of a set of autonomous cars [7] or a set of medical nano robots [11], that could be coordinated using set-plays.

Simulators usually have a limited set of behaviors available for use. A behavior can be seen as an atomic action that can be performed by an agent. The coordination of the behaviors by agents is used to compose a set-play. For example, in robotic soccer domain, run, dribble and kick are examples of behaviors that can be used in a set-play. The improvement of a set-play with new behaviors demands its manual development in a tool. Therefore, it requires knowledge of the specific source codes and a great effort of programming, besides leaving the behavior coupled and dependent on the tool.

In order to contribute to the simulation of MASs, this work proposes Behavior-Def, a solution for modeling new behaviors in a platform independent way (see Section 4). We aim to decouple the inherent knowledge of the new behavior from the technology used for its implementation. The models representing new behaviors can be reused to (semi) automatically generate implementations in various tools. This solution uses Model-Driven Development (MDD) to create new behaviors in a high-level of abstraction, independent of technology and then converts these models into native source code for specific tools for modeling, simulation and execution of a MAS (see Section 2 for more details). At this time, the solution focuses on robot soccer domain, since it comprises features that involve the major challenges, shared by robotics and AI communities [10].

This paper presents the first release of Behavior-Def as an initial study towards a solution independent of platform. This release focuses on modeling new behaviors, in a high-level of abstraction, and converts it into the correspondent code for FCPortugal Framework Setplays (FFS) [16] and SPI planner [12] tools. In summary, it comprises: (i) an interface to define behaviors in a high-level of abstraction; (ii) a code generator to convert the behavior model into the code for a specific platform. The proposal was validated through a controlled experiment (presented in Section 5) where a new behavior was created for a robotic soccer team and incorporated into the FFS and SPI planner tools, and the assessment and data analysis are presented in Section 6. Section 7 contains our concluding remarks.

II. BACKGROUND

This section briefly introduces essential concepts of MDD. MDD is the main theoretical foundation required to understand our approach.
MDD is a development approach that makes intensive use of models to represent systems at a different level of abstraction (specification, design and code). Thus, models in high abstraction levels are (semi) automatically converted through a transformation chain in models in low abstraction level until generating the application code [17]. MDD approach comprises two main elements, the models and the artifacts that represent the application in different abstraction levels; and transformations, that is the software responsible for the (semi) automation of the process.

In MDD, models are the inputs to generate other models/code. Therefore, they must be developed according to modeling languages with well-defined syntax and semantic. A transformation receives models as input and processes them to generate other models or code.

III. RELATED WORK

This section presents current tools for robotic soccer simulation analyzing their features and particularly their ability to create new behaviors.

The tool PlayMaker [19] has emerged as an effort to develop set-plays easily. Its functionalities include team formation definition, containing a set of positions that players must assume in the field according to the position of the ball; and the definition of set-plays. Another tool, named Matchflow [20], was used in the definition of attack and defense formations and the definition of flow values for field regions. The flow values are used to define which regions of the field are most attractive for the agent to move with ball possession. Matchflow deals only with concepts of flows, roles and formations, it does not support the definition of set-plays. The tool SPlanner [12] is a module of the FFS developed to designing set-plays. FFS has the set of execution mechanisms necessary for a robot soccer team to perform multi-agent collaborative behaviors planned in set-plays [16].

The three tools presented above give tactical and strategic support for robotic soccer, although, Matchflow does not support the creation of set-plays. SPlanner and Playmaker provide a fixed set of behaviors, but new behaviors need to be manually implemented extending the capabilities of the tool. Only SPlanner can test set-plays through the Soccer Server [26] simulator. These tests are performed in an integrated way. It is necessary to configure SPlanner with the location of the Soccer Server simulator, Monitor tools, FC Portugal Visual Debugger, and team binaries before performing the tests [15]. This work differs from the ones above since it enables the definition of new behaviors in a high-level of abstraction, reducing the manual programming effort. It currently evolves SPlanner and FFS, but can be extended to other tools. For the best of our knowledge, there is no other work that would allow (semi) automatic code generation of new behaviors.

IV. BEHAVIOR-DEF

This section presents Behavior-Def, a solution to define new behaviors for set-plays integrated with SPlanner and FFS tools. Fig. 1 gives an overview of the main elements of this solution: the modeling interface, to support the definition of new behaviors in a high-level of abstraction; and the code generator, to produce the necessary source code according to its model and integrates it to FFS and SPlanner tools. The following subsections describe these elements.

The methodology used to develop this work started with the analysis of the source codes of the SPlanner and FFS tools focusing on their reusable elements, e.g., classes, methods and code snippets. Based on the elements identified in this analysis we developed a modeling interface, a code generator and built transformation rules, which compose part of the instructions of this generator. Finally, the evaluation of the solution was carried out through an experiment. To assess the usability of Behavior-Def we have applied questionnaires with developers.

A. Behavior-Def Modeling Interface

The modeling interface (see Fig. 2) is based on a set of parameters that encapsulates the relevant characteristics related to behavior definition.

The interface is divided into two blocks: the first one comprises fields to define basic characteristics of the behavior, e.g. the parameter Behavior Name to set the name of the behavior and the parameter Comment, to document the source code; and the second one, a conditional block, enables the definition of a set of conditions which the behavior must respect. The items of this interface are an abstraction of elements of the FFS and SPlanner tools. The source code of the behaviors provided by these tools was analyzed to identify patterns used on code and possibilities of extension points, and these elements were abstracted in a high-level interface.

Fig. 2 illustrates the definition of Cross behavior. It uses the ball to perform the action (parameter Action) and includes the action of passing the ball from one agent to another (parameter Perform ball pass). Relationships among these parameters are defined as modeling constraints, e.g. the selection of Perform ball pass implies that there is a destination point that must be selected (parameter Has DestPoint). Table 1 details the other basic parameters of the interface that have not been mentioned yet and describe their purpose.
Figure 2. New behavior modeling interface.

### TABLE I. INTERFACE PARAMETERS THAT WERE NOT MENTIONED IN THE TEXT.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionType</td>
<td>Defines a unique identifier for the behavior that will be used as a reference in along the source code.</td>
</tr>
<tr>
<td>Uses Transition Tab?</td>
<td>Opens transition tab automatically in the selection of behavior in SPLanner. Transition tab captures detailed elements about the behavior in the passage from one step to another.</td>
</tr>
<tr>
<td>Throws the ball into the goal?</td>
<td>Represents an action that throws the ball in the goal.</td>
</tr>
</tbody>
</table>

A controlled test environment was established to generate a complete set of behaviors that can be generated by the tool, where 16 different possible compositions could be created using the Behavior-Def tool. Some of these compositions can be seen in Table 2.

### TABLE II. EXAMPLES OF BEHAVIORAL COMPOSITIONS

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Behavior</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
</tr>
</thead>
<tbody>
<tr>
<td>With Ball</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Has Destpoint?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Performs ball pass?</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Additionally, actions in SPLanner can meet selection criteria, defining specific conditions in which they can be used. To model these conditions, the second block of the interface follows the same logical structure used in the SPLanner source code and determine situations in which a new behavior cannot be performed (see the block Specific conditions in Fig. 2). In the model of Cross behavior, for example, the developer may select options to this atomic action only can be used in certain set-piece situations (e.g., corner kick, free kick, goal kick, etc), and configure to occur in special areas of the field (e.g., the center or the corner of the field).

### B. Behavior-Def Code Generator

The code generator is responsible for translating the defined behavior at a high-level of abstraction into native source code for the FFS and SPLanner. A total of 9 files are changed in code generation for these two tools.

Behavior-Def code generator was built in the PHP language and uses a code generation template engine. Therefore, a set of templates was defined, each one corresponding to a source file in the FFS and SPLanner. A template is a code model extracted from FFS or SPLanner, containing the pieces of code that will be replaced by the parameters collected in the interface. The templates in conjunction with the parameters add the characteristics of the new behavior to the FFS and SPLanner source codes.

Fig. 3 shows parts of the instructions that the Behavior-Def tool uses to generate source code. Step 1 shows how Behavior-Def captures the path of files that will be modified by code generation, step 2 shows an input from the graphical interface in PHP, in step 3 the generator inserts the input from step 2 into a source code template. The step 4 shows one of the generator transformation rules, it checks the model parameters defined by the developer if the template will compose the new behavior and in step 5 the generator opens the SPLanner file to insert the template.

![Figure 3. Source code generation process.](image-url)

Code insertion points were created in the original FFS and SPLanner files and tagged in specific locations. Step 6 searches the SPLanner file for the tag `@@_newAction_goalCenter` defined as one of the source code insertion points, and in step 7 the template is inserted in the location where the tag is compatible with the definition logic of the behavior. In this way, the resulting behavior varies according to the logic extracted from the interface. So, it will result in the selection of the appropriated templates.
with this logic and the use of their respective tags as insertion points in the source code.

C. Conceptual View

The conceptual view of the solution is represented in Fig. 4. The Index.php page represents the Behavior-Def interface. The data collected in this interface are passed to the Generator, which will create the new behavior. A log file is also generated containing information about the code inserted and updated in each source file of the SPlanner and FFS. It uses the predefined templates, each one to modify a particular FFS or SPlanner file (see Fig. 5). Using PHP’s file manipulation functions, the Generator was configured to access the files in their original folders and modify them without any intervention by the user.

V. VALIDATION

Behavior-Def solution was validated in an experiment, which aims to evaluate both, the coverage of the produced code and the development effort when creating new behaviors. This goal is defined according to the Goal Question Metric (GQM) standard [25] in Fig. 6.

To guide the evaluation the following research questions were defined: Q1. Is it necessary to include code to the new behavior generated by the solution beyond the code generated by Behavior-Def? Q2. Does the solution decrease the necessary time to develop a new behavior? In order to evaluate these research questions, we defined the metric $M_1$ and the metric $M_2$, which must be calculated according to the results obtained in the execution of the experiment as described below. Metric $M_1$ measures the coverage of the generated code and is defined by

$$M_1 = \frac{\text{LOCgen}}{\text{LOCnec}}$$

(1)

where $\text{LOCgen}$ is the number of lines of code automatically generated/modified by the Behavior-Def solution and $\text{LOCnec}$ is the total lines of code necessary to implement a new behavior. Therefore, as $M_1$ becomes closer to 1, less manual coding is required to implement the new behavior.

Metric $M_2$ is development effort, defined by

$$M_2 = \sum \text{Min}$$

(2)

where $\text{Min}$ represents the number of minutes necessary to develop a new behavior using Behavior-Def compared to the number of minutes necessary to manually develop a new behavior.

A. Planning the Experiment

The validation carried out at the laboratory of Computer Architecture and Operating Systems (ACSO) [9], by researchers and students of the Universidade do Estado da Bahia (UNEB). To completely validate the case-study, it is necessary a robotic-team integrated to FFS. We have used the team Bahia Robotics Team (BahiaRT) [9]. This team was previously integrated with FFS [21]. It is a robotic soccer team for the RoboCup 3D Soccer Simulation League.

In order to compare the results of the validation, we propose the creation of a behavior similar to another already existing in the SPlanner and FFS tools. The experiment included 4 steps: (i) behavior modeling, using the proposed interface, (ii) code generation; (iii) manually addition of lines of code, if necessary; and (iv) the test of the new behavior in a set-play. The behavior used was Dribble, which consists of action with possession of the ball, and implies in the attempt to move the robot to a destination point without losing the ball possession.

The data collection was done directly, through a questionnaire, applied during the experiment, and indirectly, through the analysis of the files generated by the participants. The questionnaire verifies the perception of the developer regarding the issues such as ease of use of the solution, utility of the solution and development time. We selected to participate in the experiment professionals who have worked recently (up to 3 years) in the ACSO group, in the BahiaRT or students who belong to the ACSO group, who have experience in the FFS and in its graphic module SPlanner.
B. Execution of the Experiment

The experiment was performed with two replications following the same validation script: 1. Open the generator page in the WEB browser; 2. Read the information in the tool manual; 3. Fill out the form with behavior data; 4. Generate behavior; 5. Enter the codes that the solution does not generate (The author); 6. Recompile the SPanner; 7. Run SPanner. 8. Create a set-play using the new generated behavior, perform the set-play export to be used by the team through FFS; 9. Answer the questionnaire; 10. Perform a practical test of execution of the set-play created in the team BahiaRT (The author). First replication was applied for a specialist participant and second replication for two current ACSO students. After the validation script, each generated behavior passed through three verification steps: 1. Analyze of each source file to verify the integrity of the generated code; 2. Recognition of the new behavior by SPanner through the creation of the set-play using the new behavior; 3. Recognition of the new behavior by the FFS through the practical test of execution of the set-play created by the team.

VI. DATA COLLECTION AND ANALYSIS

The responses of the questionnaires by the participants vary a little according to the influence of the user profile on the responses and the time of accomplishment of the task. Participant 1 developed activities in ACSO over 4 years, worked in the BahiaRT team, and developed the integration layer [21] that allowed the team to use the resources of FFS and SPanner. His work was directly linked to the development of multi-agent behavioral coordination capabilities, and in this way, he gained knowledge regarding the edition of the FFS tool and the BahiaRT team. Participant 2 has been part of the ACSO team for 2 years and 6 months, but only recently joined the BahiaRT team. Participant 2 has already made changes to the SPanner GUI. Participant 3 joined ACSO 9 months ago and has since started working on the BahiaRT team, he has experience with FFS and SPanner.

A. Questionnaire

According to participant 1, the tool fulfilled the purpose of generating the behavior and making it accessible to SPanner, facilitating this stage of development (SPanner), contributing to the understanding of the flow of creation of a new behavior. However, the tool did not fully comply with the reduced development complexity requirement, as it did not yet fully implement the methods of the Spporting, ClangAction and SetplayAction classes, which make up or interact with the FCPortugal Framework Setplay tool.

Participants 2 and 3 are more familiar with the complexity of manual editing of the SPanner tool, as they have developed solutions of this nature in ACSO, and according to them the tool has fulfilled the requirement reduced development complexity, even though Behavior-Def still do not fully implement the Spporting, ClangAction and SetplayAction classes.

B. Metrics

The graph in Fig. 7 shows the result of behavior building using the Behavior-Def solution, which presented a total average of 133 lines of code generated, in contrast to the 60 lines of code that still needed to be added to complete the behavior creation. To answer the research question Q1, the metric M1 was applied to the result of counting generated and added lines, obtaining the tool coverage with the final value approximately 0.69. Regarding research question Q2, it was not possible to compare the development times of a new behavior with and without the proposed solution, since we did not know the number of minutes spent on manually developing the behavior. However, the time spent by participants in the first stage (automatic generation) varied from 3 to 10 minutes, depending on the level of familiarity of the interviewee with the parameters requested by the Generator, resulting in a final average of 5 minutes and 30 seconds, a reasonably low time for programming 133 lines of code.

The step of creating the behavior that consists of adding lines of code that are not generated automatically was performed by the developer of the proposed solution, the same that applied the experiment and presented an average time of 4 minutes. In this phase were added the codes of the behavior Dribble, already existing in the tools FFS and SPanner. Therefore, it was not possible to estimate the time it would take to develop this code from scratch. It was not necessary to change the generated codes, just add the ones that the tool does not generate, which corresponds to the implementation of the methods of the spfcportugalporting, clangaction and setplayaction classes.

![Figure 7. Code lines.](image)

C. Verification Steps

The three verification steps presented in section V (subsection B) were performed after the behaviors were created, submitting them to the analysis of their code-to-code files (step 1) and to the practical tests in the SPanner tool (step 2) and the execution of the BahiaRT team (step 3). For the accomplishment of the steps 2 and 3, it was necessary to recompile the codes of the SPanner and the team BahiaRT, so that both could recognize the new behavior. Step 2 consisted of creating a set-play using the new behavior in the SPanner tool. SPanner only exports the set-play file in cases where the behavior codes used in that move are syntactically correct. By successfully exporting the file, set-play was able to move on to step 3 of verification where the set-play was tested in the execution of the BahiaRT team. We adopt the same version used by the BahiaRT team in [21], since this version was configured to change the flow of standard moves of the team agents to a controlled set of test cases. The team successfully executed the set-play created using the new behavior, guaranteeing the semantic validation of the codes.
generated by the proposed solution.

We performed a set of test cases outside the experiment, as seen in Table 2, which the researcher specified to evaluate the other possibilities of behaviors that can be generated by the proposed solution. With regard to the expected output (drawing on the SPLanner screen, condition for selection of behavior in the SPLanner selection menu and result of the action in the field), all results were positive.

VII. CONCLUSION

Behavior-Def was validated in an experiment to define a new behavior in a robot soccer team and had 69% of the final code automatically generated in less than 10 minutes, confirming its applicability for this scenario. It is currently being used by a robot soccer team to assist in creating new behaviors for FFS and its SPLanner module. Behavior-Def does not require knowledge in the programming language of the target tools, which facilitate the definition of new behaviors. Moreover, as a code generator it also contributes to decreasing programming errors. We are currently working in a new release of our solution to support the definition of the semantics of the behavior. We are also intending to extend the solution to generate code for other simulation and execution tools in MAS. Behavior-Def is part of a larger effort to provide a set of tools capable to support Learning new entire sets of a larger effort to provide a set of tools capable to support the definition of the semantics of the behavior. We are also intending to extend the solution to generate code for other simulation and execution tools in MAS. Behavior-Def is part of a larger effort to provide a set of tools capable to support Learning new entire set-plays from demonstration of domain experts. This is a research project in progress in our research group with preliminary results already presented [27].

REFERENCES