Describing, Adapting, and Executing Behaviors across Social Robot Platforms

Madeline Mealey, Jason R. Wilson

Franklin & Marshall College
637 College Ave
Lancaster, PA 17603 USA
jrw@fandm.edu

Abstract

Robot platforms provide various hardware and software capabilities, each unique combination resulting in vastly different interfaces, and requiring researchers and developers to focus their efforts on a single platform. The differences make it tedious and time consuming to implement social behaviors on another platform. To ease the development of social behaviors and simplify working with multiple social robot platforms, we propose using a novel language to facilitate LLM-based behavior generation and a modular middleware bridging the language to a robot's API. Through a demonstration and two ablation experiments, we show that our approach can describe a range of social behaviors, adapt behaviors to a platform, and successfully execute them on two different social robot platforms. By relieving some of the challenges in implementing social behaviors across multiple platforms, our language and technology provide an approach to advancing end user development, teleoperation authoring, and autonomous social robots.

Introduction

We rarely consider how to program or define behaviors across robot platforms, despite each of which has its own API reflecting its unique combination of hardware and software components. The task of translating a robot's behaviors to another platform is tedious and time consuming because it requires programming precise details about limbs and joints. However, we often describe robots' behaviors in abstract terms, focusing on observable changes and intent (e.g., "knock on door" (Stegner, Senft, and Mutlu 2023), "acknowledge a person"(Mahadevan et al. 2024), "search pantry" (Salfity, Anderson, and Pryor 2026)). This presents a fundamental problem that roboticists encounter in which translating from abstract goals into physical hardware is not a standardized process.

Being able to easily create social behaviors across multiple robot platforms would provide multiple benefits. For example, designing a robot's interactions could proceed independently of the development of the robot. The easy development of cross-platform behaviors would facilitate replication of research findings and strengthen claims by examining people's interactions with several robot platforms. Participatory design workshops could rapidly explore different behaviors without being limited to a particular robot platform. Many more benefits could be realized, but there stands at least three challenges in creating cross-platform behaviors: 1) describing the behaviors in a clear manner while abstracting away the details of the robot, 2) adapting descriptions of behaviors to the capabilities of a particular robot, and 3) executing behaviors across multiple platforms.

In this work we present an approach that addresses all three challenges. We define a language for describing social actions, use this language to generate and adapt behaviors to a robot platform, and execute the behaviors on multiple platforms. As evidence for our approach we provide a demonstration of a proof of concept of our approach that results in behaviors executing one two social robot platforms. Additionally, in a pair of ablation experiments, we show the effectiveness of example behaviors described using our novel language and a robot descriptor to constrain behaviors to the capabilities and constraints of a particular platform.

Related Work

The literature consistently shows that low-level robot-specific control remains a barrier, while newer systems move behavior definition upward toward reusable, modular, and more human-centered representations. Our own work generating action scripts using LLM planning and middle ware software architecture aims to fix this problem, making robotics more accessible by making the interface as usable as clicking a record audio button and speaking to the robot.

Defining Robot Behaviors

Historically, behavior definition has either come through manual encoding or algorithmic generation. Manual encoding behaviors includes directly programming physical joint movements using a robot's API, requiring detailed knowledge of the robot and how it works. Behavior trees are increasingly being used for social behaviors, such as the 30+ node tree crafted by Cooper and Lemaignan to greet a person and return to a sleeping mode 2022. Other systems use symbolic representations of a sequence of actions (Scheutz et al. 2018) or coordinated controls (Huang and Mutlu 2012) to describe a robot's social behaviors. While these methods provide a lot of control and allow the designer to program

Copyright © 2026, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

the robot to follow social norms, manually constructed behaviors are often brittle, fail to adapt to diverse social situations, and require significant development time.

To resolve the development time issue, we see an increasing number of approaches to generating code, behavior trees, and symbolic representations to define a robot's social behavior. Some approaches combine ontologies and LLM planning algorithms to generate behavior trees (Ruiz-Celada et al. 2025) or symbolic scripts (Yang et al. 2023). More recently, some have leveraged the generative capabilities of LLMs to generate behavior trees (Chen et al. 2024; Salfity, Anderson, and Pryor 2026), however it is notable that other uses did not use the LLM to generate the model and instead used it to understand user intent (Chen et al. 2024) and context (Ruiz-Celada et al. 2025; Kang, Ben Moussa, and Thalmann 2024).

Our approach is more similar to (Salfity, Anderson, and Pryor 2026) and (Mahadevan et al. 2024), who use LLMs to generate behavior trees or Python code, respectively, to control the robot. Instead, we generate an action script, a symbolic representation of the robot's behavior. Like behavior trees, our script requires an execution layer to process the script, whereas GenEM produces code that is directly executable (Mahadevan et al. 2024). Our work is also similar to (Chen et al. 2024) and (Mahadevan et al. 2024) in that we leverage the LLM to interpret natural language descriptions of the intended robot behavior.

Generating Cross Platform Behaviors

Historically in order to adapt these behaviors functionally across robot platforms, researchers have used many different techniques. Some researchers have used LLMs to generate platform specific code using robot APIs (Mahadevan et al. 2024). Others have used generated behavior trees to cross-reference the robot API in order to generate platform specific details (Salfity, Anderson, and Pryor 2026). While Previously, we developed a behavior generation program which was able to be used cross-platform by utilizing separate middlewares which processed the script with the specific hardware in mind (Wilson and Yang 2024). Additionally, every robot platform has a different notion of action, which creates a problem for cross-platform use because each robot fundamentally requires its own defining actions. So to integrate these conflicting control paradigms, Elbeleidy and Wilson introduced translation interfaces that identify the "action" as the core unit of representation (Elbeleidy and Wilson 2024).

In our work, we build directly upon these abstraction principles by developing a centralized behavior generator that plans robot actions independently of the physical target platform. The generated behavioral plans are transmitted directly to a modular middleware layer. This middleware acts as a universal adapter, translating the high-level behavioral sequence into native, runtime-compatible commands for any target robot-specific program. By abstracting the generation logic away from physical runtimes, our architecture guarantees high behavior reusability and protects the core planner from platform-specific API variances.

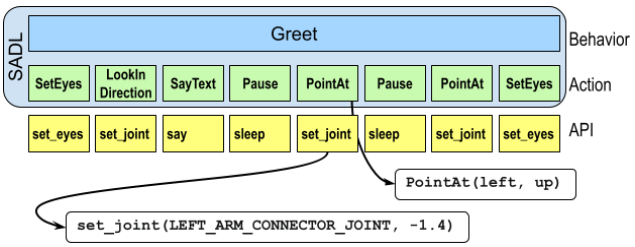


Figure 1: The "Greet" behavior is described using a sequence of 8 SADL actions, each of which get mapped to an API call by the Middleware (parameters to actions and API calls not shown). Callouts show the PointAt action, which is used to make the robot do a small wave gesture.

Technical Approach

To easily define, generate, and execute social robot behaviors across robot platforms, we used the Social Action Description Language (SADL) as the *lingua franca* across platforms. A Behavior Generator and Middleware (see Figure 2) provided the technical implementation that uses the SADL to translate a high-level instruction into behaviors adapted to and executed on a robot.

Social Action Description Language

The terms "action" and "behavior" have many interpretations in robotics (Elbeleidy and Wilson 2024). We define an action as an abstraction representation of what a robot can do. Actions may be composed together to form more complex expressions, which we call a "behavior". In Figure 1, we show how the "Greet" behavior is broken down into a sequence of eight actions. We introduce the Social Action Description Language (SADL) to **describe** social behaviors in terms of a sequence of primitive actions. SADL does not support decomposing actions into more granular actions, and instead the decomposition is expected to be defined outside the language and in a platform-specific mechanism, thus allowing for mappings from the SADL actions to robot commands to happen in a way particular to the capabilities of the robot.

SADL consists of six primitive actions: LookInDirection, Pause, PointAt, SayText, SetEyes, and TiltHead. Most of the names of the actions are chosen to describe the social semantics (e.g., PointAt) of the robot's behavior, as compared to robot semantics (e.g., MoveArm). These six primitives represent the core functions available to the majority of social robots, excluding base locomotion. This set of actions was sufficient to perform all necessary socially assistive behaviors, such as introducing a task, encouraging a child, offering hints on a puzzle, helping locate missing puzzle pieces, and celebrating task completion (Yang et al. 2023).

The intent is to abstract away the details of the robot, thus leaving the exact mechanism for executing the action left to the platform-specific middleware. Similarly, the parameters of each action are described qualitatively (e.g., left, lower-right), instead of numerically (e.g. angle of a joint). Primitive actions are composed into an Action Script, a sequence

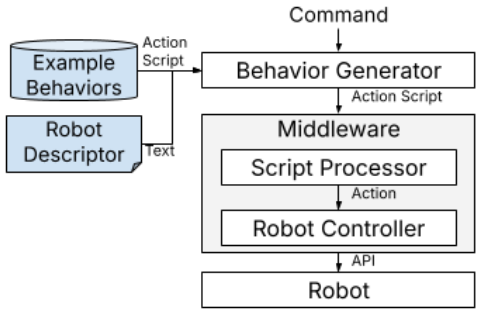


Figure 2: The Behavior Generator and Middleware work together to define, generate, and execute social behaviors across robot platforms. All Action Scripts and Actions are defined using SADL.

of actions defining a particular behavior. Represented in a standardize JSON format, defines a name for the behavior, description, and the sequence of primitive actions.

The JSON format serves as our cross-platform representation to define a robot's social behavior. The JSON format is used in two ways. First, it is the representation that is generated by the Behavior Generator component and passed to the middleware for execution. For the Behavior Generator to create this format, a description of the language and the JSON syntax are provided as part of the prompt to the LLM. We also define the example behaviors using SADL and represented in the JSON format.

Behavior Generator

The Behavior Generator translates a natural language description of a robot's social behavior into a fully executable Action Script. It uses an LLM to generate the script based on a command to the robot, a set of example behaviors, a robot descriptor to adapt to , and a description of the semantics of SADL and the corresponding JSON syntax.

Command The command is a natural language instruction in plain text describing what the robot is to do. The command is expected to be abstract, generally describing the robot's behavior in simple terms, not providing any low-level details specific to the robot. Since the LLM and the middleware to execute the behavior do not have access to the robot's perceptions, the command would need to include these details to behave correspondingly. This may include the location of a user or object.

Example Behaviors Example behaviors are a set of vetted behaviors manually defined by the authors to provide guidance to the LLM in generating good behaviors. Each example defines a single behavior and demonstrates how to compose together primitive actions that reflect social conventions while leveraging the affordances available on a social robot. The set of examples can be easily expanded, and we provide eleven predefined behaviors that cover multiple uses of each primitive action in a variety of scenarios, from greeting a person to telling a joke. The examples, which are defined in SADL and encoded in a JSON file, also helps ensure that the LLM output adheres to the semantics and syntax of SADL.

Robot Descriptor The Robot Descriptor contains a textual description of the physical affordances of a particular robot platform. This is necessary because when robots have significantly different affordances, it allows the behavior generator to avoid conflicting actions. The Robot Descriptor is a tool for allowing cross platform use because a new robot descriptor can be made for any robot.

Middleware

The middleware is a unique segment of the architecture, which we built to map social semantics to robot semantics. In other words, this is where the code differentiates the universal actions into the separate robots' own languages and executing an Action Script on a specific robot. It consists of two main components: the Script Processor, and the Robot Controller.

Script Processor Before the Script Processor, the action scripts are viewed as a whole which allows for the user to input high-level behaviors. The Script Processor, is where the action script is broken down into each individual action which is then passed to the Robot Controller to be converted into specific API calls. The Script Processor aims to be universal. It is compatible with any robot and its corresponding Robot Controller. It also provides a standardized interface, allowing action script (regardless of how it was generated) to control a robot's behavior.

Robot Controller The Robot Controller is where each SADL action is converted into a robot's API calls. Each robot platform has a unique controller, thus allowing each action to be manifested in a manner that is specific to that robot. Actions may be mapped to single API calls (as shown in Figure 1). Alternatively, actions may result in multiple controls, particularly when multiple movements are necessary to perform the action.

Since the Robot Controller directly interfaces with the robot, this is the only place in our approach where a specific robot platform must be specified. The Robot Descriptor, which is optional, often needs to work in tandem with the Robot Controller, and knowledge of how the controller maps actions to API calls is beneficial in authoring the descriptor. For example, if the Robot Controller uses head gestures for pointing, the Robot Descriptor may need to limit the simultaneous use of the LookInDirection and PointAt actions.

Proof of Concept

The proof of concept demonstrates the translation of spoken commands into executed behaviors on two social robot platforms. The Misty robot features a 3 DoF head, 1 DoF x 2 arms, and an LCD screen and for the face. The Quori v2 robot was run in simulation because the physical robot had not yet been manufactured as of the writing of this work. The body of the Quori was run in Gazebo (see Figure 3), and the it features a 2 DoF head that moves up/down and left/right but does not tilt its head side to side. It also does not have arms, and the display of the face was powered by

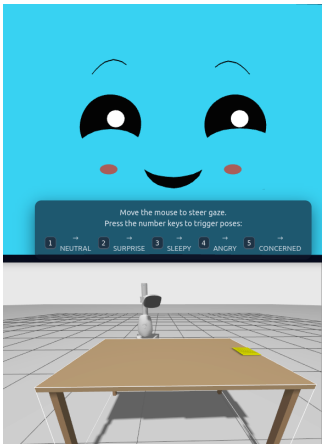


Figure 3: In simulation, Quori works in conjunction with Vizij in order to demonstrate the setEyes action.

Vizij (Elbeleidy et al. 2026). The face of the robot would be displayed on the LCD face, but the details of the face are not available in Gazebo. Instead the Vijiz face is displayed in our demonstration in a separate window. The key differences between these platforms were the arms and head tilt. The Misty has two simple flipper arms, and the Quori v2 has no arms. The Misty robot's head can tilt, moving along the roll axis, but the Quori cannot. Additionally, since SADL was originally created with Misty in mind, no robot descriptor was used when it was active. When using Quori, we used a Robot Descriptor in order to specify its affordances. This included an explanation of uts lack of arms and instructions to create behaviors so that no conflicting actions were being called at once (i.e point and look).

The Middleware were implemented in ROS2 and supports modularity and reusability by consisting of two nodes: a platform agnostic script processor, and a platform specific controller. The script processor receives the action script and prepares calls to the controller for each action in the script. Each robot controller was designed to use the same interface, defined as the actions available in SADL, allowing the script processor to use any controller without any changes. The controller defines the mapping from SADL actions to robot API calls. Separate controllers were developed for the Misty and Quori robots.

The Behavior Generator, also impelemnted as a ROS2 node, uses Gemini 3.1 Flash Lite for its LLM. It receives a description of the robot's behavior as a plain text command expressed in natural language. For the purposes of this demonstration, we included an automatic speech recognizer using the Google Web Speech API to translate the command from spoken utterance to plain text.

To explore a range of behaviors and how they are translated to each robot's capabilities, we recorded the robot responding to seven commands. These commands were selected because they collectively used all 6 of the available actions and demonstrate similarities and differences in the performed behaviors. We list here each command and what it revealed about how our architecture adapted the commands to each robot platform:

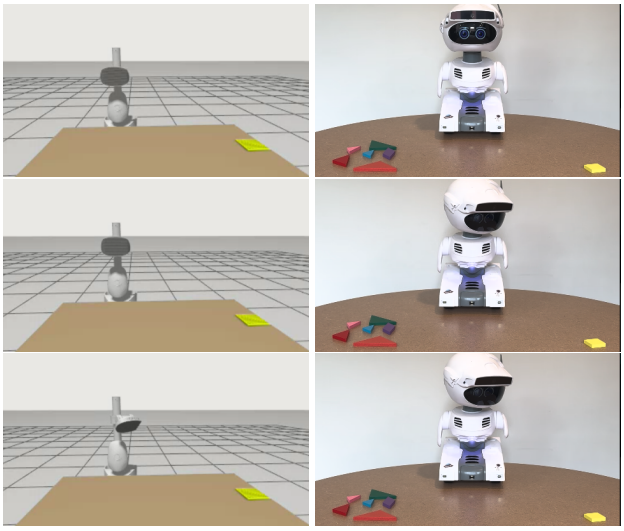


Figure 4: Each robot performs a behavior to point at the yellow piece.

- Introduce yourself: Used facial expressions in coordination with speech. The behaviors were very similar across platforms.
- Look for object on table: Used repeated head movements. The Quori behavior used fewer head movements but coordinated head movements with the eyes.
- Point at object: Used a pointing gesture. The Misty behavior coordinates head and arm movements, but Quori used its head to point. See below for a more detailed examination of this behavior.
- Give encouragement: The Misty robot coordinated a facial expression with speech and a head tilt. The Quori behaved similarly except for the elimination of the head tilt.
- Indicating correctness non-verbally: Used facial expression and a head nod. The behaviors are very similar.
- Indicating incorrectness: Misty coordinated speech with a head tilt and facial expression. Quori similarly coordinated speech and a facial expression but looks to the side instead of using a head tilt.

In Figure 4 we show the behaviors of the robots side by side in response to the command "point at the missing piece to your lower left". The Quori robot starts at its default position. To prepare a pointing gesture with its head it first looks up slightly before dipping its head in the direction of the yellow square. The Misty robot also starts in its ready position before looking at the yellow square. It then makes a small movement with its left arm to gesture in the rough direction of the yellow piece. The smaller movements of the robots are more noticeable in the video found in the Supplementary Materials.

Experiments

To assess the contributions of components of the architecture, we conducted two experiments. The first examines the role of the example behaviors, and the second experiment looks at the effectiveness of the robot descriptor.

Effectiveness of Example Behaviors

The example behaviors are intended to guide the LLM (Gemini 3.1 Flash Lite) in generating executable Action Scripts. These pre-defined behaviors give examples of how actions can be used, with the expectation that generated action scripts will use the actions similarly. Furthermore, the examples demonstrate how sequences of actions can be constructed to create a larger behavior. The expectation is that the generated behaviors will have similar sequences.

To test this expectation, we compared action scripts generated with and without a target example. To generate each action script, the architecture was prompted to create the behavior depicted in the target example. The target example then serves as the gold standard and the point of comparison for both generated action scripts. We use the example as a point of comparison because it was carefully constructed and known to produce a valid and recognizable behavior. Additionally, the intent of introducing examples is to skew the output towards these examples. If including the example behaviors is effective, the action scripts generated when the examples are provided would be more similar to those examples.

Methodology. For each of 11 hand-authored example behaviors, we generated an action script under two conditions: zero example behaviors provided, and only the one matching the target behavior. In both conditions, all other examples were not given. To account for variance in the LLM output, we repeated each condition $N = 3$ times.

To determine the similarity between the action script that was generated and the one from the target behavior, we calculated a normalized similarity score based on the Levenshtein edit distance of the two action sequences: $(1 - \text{editDistance}/\max(\text{len1}, \text{len2}))$. To calculate the edit distance, we treated the sequence of actions like a string of characters – calculating how many actions were inserted, deleted, or substituted. In this calculation, we represented each action with two granularities: the action name only (e.g., LookInDirection) and action name plus arguments (e.g., LookInDirection(down)) The former tests whether the generated behavior has the same structure, while the latter checks for an exact match. In addition to comparing each generated action script with the target behavior, we also compared the generated action scripts with each other to assess how much the script changed when the target example was included.

To analyze the types of insertions and deletions, we created separate tallies for the Pause action versus the other actions. This is due to the Pause action being manipulated more frequently than the other actions.

Results. Providing a single example increased similarity to that example's action sequence for all 11 behaviors using both analyzes. The name-only comparisons (see Table 1), which measures the structural similarity of the behaviors, showed the average similarity score improving from 0.48 to 0.74. The more granular comparison (see Table 2) that included the parameters of the actions showed the average similarity score improving from 0.38 to 0.64, indicating that not only was the overall structure of the behavior improved but also the details of the behavior more closely matched the example. The greatest improvements were found in the Greet and Introduce behaviors, with Greet being nearly an exact match of the example behavior when it was included as a reference point.

The LookForSomething and Nod behaviors showed the smallest improvements, with the LookForSomething behavior greatly differing from the example regardless of whether the example was included as a reference. Interestingly, the two generated behaviors for LookForSomething bare more similarity to each other than the target example, indicating that the system did not incorporate the example in any significant way. We contrast this with the Greet behavior, where the two generated behaviors were not similar to each other. That significant difference was the result of going from many differences with the example behavior when no example was provided, to having the same sequence of actions with only a small difference in argument.

In examining the source of the differences with the example behavior, we found that generally more actions were deleted from the example than inserted. The LookForSomething and Wave behaviors had the most deletions. The LookForSomething, which was intended to look in every direction, but the generated versions consistently left out looking into the corners (e.g., upper left). The Wave behavior generally included few repetitions of the arm's movement, and similarly the Nod behavior sometimes included one fewer repetitions of the robot's head. The consistent trend is that behaviors that have actions repeated are simplified to perform fewer repetitions of the actions.

Robot Descriptor Constraints

We integrated a robot descriptor to help adapt the generated behaviors to a particular robot platform. Specifically, we examined whether the robot descriptor was successful in limiting the generated behaviors to use only the modalities available in the Quori v2 robot, which has no arms and cannot tilt its head. The middleware for the Quori v2 was designed to use head gestures for pointing. Given the constraints of not head tilt and head gestures for pointing, our goal was to verify whether the descriptor would be successful in ensuring that the HeadTilt action was not used and that LookInDirection and PointAt did not co-occur since they would conflict. To test this expectation, we compared the action scripts generated with and without the descriptor to verify that the descriptor had the desired effect.

Methodology. We generated six behaviors for the Quori robot, with and without the descriptor. To account for variance in the LLM output, we repeated each condition 3 times. The robot descriptor, expressed in natural language, contained for types of statements:

Example	Sim(ex,w/o)	Sim(ex,w)	Sim(w/o,w)	Insertions (w)		Deletions (w)	
				Pause	Other	Pause	Other
Home	0.43	0.71	0.53	0.0	1.7	0.0	0.0
LookForSomething	0.40	0.44	0.73	0.0	0.0	5.3	5.3
LookAroundTable	0.40	0.73	0.50	0.0	0.0	1.3	1.0
Greet	0.51	1.00	0.51	0.0	0.0	0.0	0.0
Nod	0.61	0.67	0.63	0.0	0.7	0.0	1.0
Introduce	0.28	0.81	0.31	0.0	0.0	0.7	0.3
Shake	0.61	0.82	0.67	0.0	1.0	0.0	1.0
ShowLowerRight	0.73	0.93	0.79	0.0	0.0	0.7	0.0
Surprised	0.57	0.71	0.51	0.0	2.0	0.0	0.0
Wave	0.40	0.62	0.62	0.0	0.0	2.0	2.7
Joke	0.39	0.67	0.51	0.0	0.7	0.0	1.3

Table 1: Sequence-alignment analysis, action names only (parameters ignored). Sim columns are the Levenshtein-based similarity between the pairs of scripts: example, generated without example present, and generated with example present. Insertions/ Deletions are actions present in the with-example generation but not the example (Insertions) or vice versa (Deletions), split into Pause vs. all other action types.

Example	Sim(ex,w/o)	Sim(ex,w)	Sim(w/o,w)	Insertions (w)		Deletions (w)	
				Pause	Other	Pause	Other
Home	0.33	0.70	0.50	0.0	3.0	0.0	0.0
LookForSomething	0.33	0.39	0.69	0.0	0.0	10.7	11.7
LookAroundTable	0.30	0.70	0.42	0.0	0.0	3.0	2.3
Greet	0.45	0.92	0.48	0.0	0.0	0.0	0.0
Nod	0.44	0.48	0.58	0.0	1.3	0.0	2.0
Introduce	0.22	0.71	0.21	0.0	0.0	1.3	1.0
Shake	0.44	0.70	0.60	0.0	1.0	0.0	2.0
ShowLowerRight	0.57	0.79	0.75	0.0	0.0	1.3	0.0
Surprised	0.46	0.62	0.50	0.0	4.0	0.0	0.0
Wave	0.33	0.50	0.58	0.0	0.0	5.0	8.0
Joke	0.27	0.50	0.43	0.3	1.0	0.7	2.7

Table 2: Same analysis as Table 1, but includes action arguments in the comparison.

1. Physical limitations: no arms, cannot tilt head
2. Substitution: use point for deictic gestures, use look for other gestures
3. Co-occurrence: when pointing, don't look
4. Standard operation: emotional expression, looking, and speech are normal.

To evaluate the Robot Descriptor's effect on maintaining the constraints, we selected six distinct scenario types that without the descriptor featured simultaneous use of pointing and looking (2 scenarios), head tilting (2 scenarios), both, or neither (the control condition). Below is the behaviors, prompts, and purpose:

- **Indicating**: *"Show the child the red block on the table."* Promotes deictic gestures.
- **Encouragement**: *"Encourage the child to keep trying on the puzzle."* Uses head tilt to show empathy.
- **Curiosity**: ‘*React with curiosity when the child asks you a confusing question."* Head tilt to show curiosity and confusion.
- **Joke**: *"Tell the following joke: why did the robot go to the shoe store? ... to get rebooted."* Based on an example behavior, uses a TiltHead as a comedic-timing beat.
- **Regulating**: *"Give the child words of encouragement when they seem frustrated with the puzzle."* Uses head tilt to show empathy and sometimes uses point/look combo.
- **Control**: *"State that a square has 4 sides."* A purely factual statement to not promote any social behaviors.

Results. The purpose of the experiment was to evaluate how effective the robot descriptor was in constraining the generated behaviors. In Table 3, we see that there were no violations of the constraints. The combined look and point actions dropped from 7/18 to 0/18 generated behaviors. The TiltHead action was used in 9/18 behaviors generated without the descriptor and none of the behaviors with it. Perfectly adhering to the constraints was not merely the product of over filtering, and instead there is evidence that the constraints were followed with precision while maintaining the intent of the command. For example, the Indicating prompt retained its deictic gesture using PointAt but dropped the LookInDirection action when using the descriptor. This shows adherence to the condition that it uses point for deictic gestures and the constraint that looking is not used in tandem with pointing. Since the descriptor indicated that the robot cannot tilt its head side to side, all nine of the uses of the TiltHead action were replaced with a LookInDi-

Prompt	Combined Action		TiltHead Used	
	Without	With	Without	With
Indicating	3/3	0/3	0/3	0/3
Encouragement	3/3	0/3	0/3	0/3
Curiosity Encouragement	0/3	0/3	3/3	0/3
Frustrated	1/3	0/3	3/3	0/3
Joke	0/3	0/3	3/3	0/3
Control	0/3	0/3	0/3	0/3
Total	7/18	0/18	9/18	0/18

Table 3: Effect of the Quori robot descriptor on prohibiting combined head/arm actions and head titls ($N = 3$ repeats per prompt per condition). *Combined Action* = fraction of runs whose script contains both LookInDirection and PointAt. *TiltHead Used* = fraction of runs using TiltHead.

rection action. One important example of this substitution is found in the Joke prompt. Without the descriptor, the generated behaviors followed the pattern found in the example behavior, which had the robot tilting its head while pausing before giving the punchline. With the descriptor, the behavior included a similar head gesture, looking to the side instead of tilting its head. In addition to demonstrating that the constraints in the descriptor were consistently followed, it also shows the descriptor taking precedence over the actions describe in the provided examples.

Discussion

In this work, we proposed an approach that uses a standardized action language in conjunction with an architecture featuring robot middleware and an LLM-based behavior generator to define, adapt, and execute behaviors across two social robot platforms. The Social Action Description Language (SADL) supports defining example behaviors and communication between the Behavior Generator and Middleware. The robot descriptor enables the LLM to adapt behaviors to the capabilities and constraints of a particular robot platform. The Middleware implementation uses a modular architecture to connect general purpose (i.e., cross platform) components to platform-specific API calls.

The proof of concept provides evidence that our approach is able to control robots across different platforms using the same command. Additionally, it shows that a single LLM is able to successfully create fully executable action scripts. This is in contrast to GenEM, which relied on a series of LLMs because a single LLM often generated non-executable code (Mahadevan et al. 2024). An ablation experiment on the example behaviors showed that including example behaviors resulted in generated behaviors that more closely aligned with the examples. This shows that the examples may be used to promote particular patterns that may align with desired behaviors reflecting social or cultural norms, social status, personality, or communication style. Another ablation experiment provided evidence that the robot descriptor was able to adapt behaviors to the constraints of the Quori robot - namely the lack of arms and head tilt. The descriptor did not use a formal language to define a robot, thus allowing one to easily describe how behaviors should be adapted to a robot platform.

Describe. The Social Action Description Language is the language for defining robots' social behaviors and is used in the example behaviors and the interface to the Middleware. Our action names and parameter values use behavior semantics, as opposed to robot mechanic semantics. Additionally, the parameter values are qualitative rather than quantitative. These factors together create a language that allows the description of the robot's actions be in terms of what the robot is doing, not how. This creates a language which makes it easy for users to define the robot's movements. This is especially true for users who are not robot experts. This also lends itself well to incorporating LLMs to generate behaviors, because the LLM should be able to more easily interpret the behavior semantics compared to specific lower-level implementations of each movement, which would likely introduce errors.

The examples were generally beneficial, with some notable exceptions: particularly the LLM removed what it deemed to be excessive directions or repetitions. Three notable examples of this are LookForSomething, Wave, and Nod. In the LookForSomething example, the LLM only allowed the robot to look up, down, left, or right rather than including upperRight, upperLeft, lowerRight, and lowerLeft as our original example held. Wave removed half of the arm movements, and the Nod removed one of the head movements. Although the deletions could be concerning, we have not tested this with users. Empirical tests, similar to the comparison studies done in (Mahadevan et al. 2024) will be necessary to evaluate whether these deletions or other alterations made by the LLM produce recognizable behaviors.

In order to correct these errors, we could prompt the LLM to match the examples more. However,it might cause the LLM to overcompensate and match the examples too closely, and limiting how much the LLM adapts the examples to a particular context or robot. Alternatively, we could treat the examples as reusable, blackbox modules and prompt the system to directly use an example when applicable. GenEm, which incorporates a code library consisting of executable functions to perform a behavior, supports reuse by calling a previously defined function (Mahadevan et al. 2024). However, the code is specific to a platform and inhibits reusing behaviors across platforms. Also, there is a tradeoff between how much you want to reuse an existing behavior versus adapt it to a specific need or context.

Expanding the set of examples may support more reuse of behaviors, but the challenge lies in ensuring the quality of the behaviors. Automatically incorporating user feedback is one mechanism for integrating generated behaviors into the set of examples while maintaining a quality standard (Mahadevan et al. 2024). User feedback would be possible in the offline generation of behaviors, but the online generation, where the behaviors are part of an autonomous system, would prevent a reasonable opportunity to collect feedback directly from the user. Without user feedback or some other

form of behavior vetting, the system will degrade when poor quality behaviors compound as they become the basis of future behaviors.

Adapt. The Robot Descriptor is very effective in allowing SADL to be adapted across two robots. One robot (Misty) had full functionality, and no descriptor was used to generate behaviors for this robot. We used the descriptor to prevent behaviors generated for the Quori robot to contain head tilts or simultaneous use of looking and pointing gestures. There robots are similar in that they are designed to be social robots and include anthropomorphic features like a head. Other social robots that have similar features (e.g., Nao) are expected to work equally well. Adapting the behaviors to robots that are not anthropomorphic, such as a quadruped, as done by GenEm (Mahadevan et al. 2024) would require coordination between the Descriptor and Middleware.

We leveraged the language interpretation capabilities of the LLM and expressed the capabilities and conditions of the robot in natural language. In contrast, BE-AWARE used formal modifiers to identify the conditions for adapting the behavior, requiring the designer to be familiar with the formalization and available conditions (Ruiz-Celada et al. 2025). GenEm, on the other hand, adapted to a robot platform by including a library of code snippets (Mahadevan et al. 2024), essentially combining the purposes of our Example Behaviors and Robot Descriptor. While their approach provides reusable code that can be composed together to define new behaviors, it requires each robot platform to have its own library of examples. By separating the examples from the descriptor, the examples are applicable to multiple robots. Our experiments also showed that the Robot Descriptor was able to use and adapt an example, demonstrating that these features can work together.

We showed that the Robot Descriptor can adapt a behavior to a robot *platform*, and we expect the approach could also be used to adapt to a particular robot *instance*. For example, a robot may currently have a malfunctioning left arm, and the descriptor could provide a constraint to not use the left arm in any behavior. Since the Robot Descriptor used in our testing did not refer to a robot platform by name, it was unlikely that it integrated knowledge about that platform that may have already existed in the LLM. Without any prior knowledge of the robot, a descriptor that specifies a robot having no arms (by design) or one arm (by malfunction) is likely to be handled similarly. Future work will verify this, along with exploring how the descriptor could also be used to customize this behavior in other ways. For example, the behaviors could be adapted to environmental context (e.g., the robot is sitting on a table) or to express a particular communication style (e.g., high use of gestures).

Execute. The Middleware is a modular architecture implemented in ROS 2 and provides a common interface for executing social behaviors. By dividing the middleware into a script processor and robot controller, we are able to reuse a portion of the middleware (i.e., the script processor) with any robot platform. The robot controller, which is responsible for mapping SADL actions into robot API calls, ensures that platform-specific code is kept to a minimum.

Since the Middleware accepts an Action Script expressed in the JSON syntax for SADL, the Middleware is compatible with other Action Script sources, including those manually authored or generated using HTN planning (e.g., (Yang et al. 2023)). It also makes the Middleware compatible with teleoperation interfaces that may predefine short behaviors for the robot to perform (Elbeleidy and Wilson 2024). Buttons in these interfaces could result in an Action Script being sent to the Middleware for execution.

Limitations and Future Work

Our approach defines six actions that result in asynchronous service calls to control the robot. This design choice results in limitations related to actions to support robot locomotion, temporal dependencies between actions, and integrating the robot's behavior with its perceptions.

- Locomotion: The current version of SADL defines actions for only the upper body of a robot. SADL will be expanded to support robot locomotion (e.g., walking or driving) and body postures such as sitting.
- Temporal Dependencies: SADL uses a sequence of actions to define a behavior and relies on the Pause action to coordinate timing between actions. Correspondingly, the Middleware implements all actions as asynchronous service calls. To better handle coordinations, such as initiating an action after the completion of another action, SADL will be updated to include the specification of temporal constraints and the Middleware will leverage the ROS 2 action framework, allowing for API calls to be made synchronously.
- Integrating Perceptions: SADL uses qualitative descriptions of locations, assuming that the general location of an object to be looked or pointed at is known and can be provided to the Behavior Generator. However, this assumes that the object does not move and thus would inhibit any following or tracking behavior.

Each action uses qualitative parameters, as opposed to more granular, quantitative parameters. The tradeoff is that the qualitative parameters, which carry the semantics of natural language, are designed to be easier for language-based reasoners (LLMs and humans), but we sacrifice the precision available when using numeric values. Additionally, the actions are purely performative in that they do not incorporate any feedback from the robot's environment. In contrast, behavior trees rely on sensory feedback to advance its execution. Future work will explore how sensory information can be integrated into SADL and the behaviors it can describe.

Conclusion

We presented here a significant step towards enabling robot designers, content authors, end user programmers, researchers, and others to easily create and explore social behaviors. To avoid needing detailed knowledge of a robot platform, we introduced a high-level language for describing behaviors, an LLM-based behavior generator using an informal robot descriptor to adapt behaviors to a particular platform, and a middleware that executes behaviors expressed in the high-level language by mapping it to API calls.

Acknowledgments

This research was supported in part by a Hackman Summer Scholar Research award.

References

Chen, X.; Cai, Y.; Mao, Y.; Li, M.; Yang, W.; Xu, W.; and Wang, J. 2024. Integrating intent understanding and optimal behavior planning for behavior tree generation from human instructions. *arXiv preprint arXiv:2405.07474*.

Cooper, S.; and Lemaignan, S. 2022. Towards using behaviour trees for long-term social robot behaviour. In *2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 737–741. IEEE.

Elbeleidy, S.; Schoen, A.; Birmingham, C.; Ribeiro, T.; Paleologue, V.; Dooley, D.; and Mead, R. 2026. Vizij: Supporting the design, animation, and deployment of rendered robot faces. In *Proceedings of the 3rd Symposium on Unifying Representations for Robot Application Development.*

Elbeleidy, S.; and Wilson, J. R. 2024. Agreeing to Disagree: Translating Representations to Uncover a Unified Representation for Social Robot Actions. In *Proceedings of the AAAI Symposium Series*, volume 4, 346–352.

Huang, C.-M.; and Mutlu, B. 2012. Robot behavior toolkit: generating effective social behaviors for robots. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, 25–32.

Kang, H.; Ben Moussa, M.; and Thalmann, N. M. 2024. Nadine: A large language model-driven intelligent social robot with affective capabilities and human-like memory. *Computer animation and virtual worlds*, 35(4): e2290.

Mahadevan, K.; Chien, J.; Brown, N.; Xu, Z.; Parada, C.; Xia, F.; Zeng, A.; Takayama, L.; and Sadigh, D. 2024. Generative expressive robot behaviors using large language models. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, 482–491.

Ruiz-Celada, O.; Molina, V.; Suárez Feijóo, R.; Ud Din, M.; Zaplana Agut, I.; and Rosell Gratacòs, J. 2025. Using ontologies for adaptive planning and robust execution in robotic manipulation. In *Proceedings of the Joint Ontology Workshops (JOWO)*, 1–6.

Salfity, J.; Anderson, R. B.; and Pryor, M. 2026. Contract-Grounded Behavior Tree Synthesis via Coding Agents. *arXiv preprint arXiv:2607.12220*.

Scheutz, M.; Williams, T.; Krause, E.; Oosterveld, B.; Sarathy, V.; and Frasca, T. 2018. An overview of the distributed integrated cognition affect and reflection diarc architecture. *Cognitive architectures*, 165–193.

Stegner, L.; Senft, E.; and Mutlu, B. 2023. Situated participatory design: A method for in situ design of robotic interaction with older adults. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 1–15.

Wilson, J.; and Yang, Y. 2024. Software architecture to generate assistive behaviors for social robots. In *Companion of the 2024 ACM/IEEE International Conference on Human-Robot Interaction*, 1119–1123.

Yang, Y.; Langer, A.; Howard, L.; Marshall, P. J.; and Wilson, J. R. 2023. Towards an ontology for generating behaviors for socially assistive robots helping young children. In *Proceedings of the AAAI Symposium Series*, volume 2, 213–218.