Failure Detection and Dynamic Extensions for Behavior-Based Subsumption

Frederick W. P. Heckel, and G. Michael Youngblood
University of North Carolina at Charlotte
9201 University City Blvd
Charlotte, NC 28223
{fheckel, youngbld}@uncc.edu

Abstract

Behavior-based and reactive control methods are popular choices for building fast and lightweight intelligent controllers for resource-constrained systems. Reactive methods are extremely useful in highly resource-constrained applications, but at a cost: they tend to be even more susceptible to certain types of failures than deliberative techniques. Without a planner to adapt to changes, even a small failure can result in incorrect behavior from the entire controller. In this paper, we propose extensions to behavior-based subsumption that can detect four types of failures.

Introduction

Reactive control methods, while useful for building computationally inexpensive intelligent systems, suffer from susceptibility to failures that deliberative systems can handle more effectively. Using deliberative systems in these cases is not always possible, especially in resource-constrained applications, such as video game AI or robotics. Without sufficient metadata to detect errors and methods for reconfiguring the controller when errors occur, small problems can quickly build up to become catastrophic failures for the agent. The types of failures that can limit the applicability of reactive controllers may be due to changes in the agent's capabilities, modifications to the environment, or design oversights. In modern games, this can occur frequently when a character is injured or the environment undergoes a major dynamic change.

In this paper, using the monitoring techniques of planning systems as a model, we propose extensions to reactive and behavior-based systems that allow detection of four distinct classes of failure. While they are useable with other reactive and behavior-based methods, these extensions are presented in the context of a behavior-based subsumption architecture (Brooks 1986). We show that failure detection can provide new capabilities such as dynamic adaptation of subsumption controllers and inexpensive multi-agent coordination.

Behavior Monitoring

We have identified four types of errors that can be easily detected in the subsumption architecture: activation failures, capability failures, behavior interaction failures, and environmental failures. Each of these failures can be detected in a well-defined manner, though some are more expensive than others to detect.

We assume that we are dealing with two types of agents: mobile robots and virtual characters. Both commonly use reactive control, are resource-constrained, and use highly crafted controllers. There are major differences between these types of agents: the perceptions of virtual characters are highly restricted by the level of detail present in their environment, but require less processing than the output of sensors carried by intelligent robots.

Activation Failures Activation failures are the simplest types of failures to detect. These failures occur when a component of the controller fails to execute. This indicates that anticipated conditions never occur or that a higher-level behavior is preventing the execution of the layer. Activation failures can be easily detected in both virtual characters and robots, as they depend entirely on software state. While they are simple to detect, these are also the least reliable failures, as some behaviors may be expected to activate only occasionally.

Capability Failures Capability failures occur when an agent is not capable of executing a given behavior. These directly correspond to unanticipated changes in agent state; in a virtual character, it may be an item missing from the character's inventory, or an action that is no longer allowed. In robotics, this could correspond to a hardware failure or a partial loss of power. Capability failures can be detected using trigger conditions from the architecture.

Environmental Failures Environmental failures occur when agent capabilities are correct and the behavior executes, but the behavior action fails in the environment. An environmental failure may indicate that the expectations for the behavior are incorrect given the environment (e.g., a lift behavior may fail if objects in the environment are heavier than expected by the behavior) or that another agent has interfered with the execution of the behavior. These errors are relatively easy to detect in virtual environments, as the simulation in which the agent is running will return an error.

Detection of environmental failures due to other agents is feasible in real-world environments, but other types of failures may not be easily detected.

Behavior Interaction Failures Behavior interaction failures are potentially the most difficult failures to detect. This type of failure occurs when a behavior can be executed correctly, does not generate an environmental error, but also does not achieve the expected result. Failures of this type are due to behaviors interacting in a way that causes one behavior to undo or cancel out the effects of another. Detecting behavior interaction failures requires specifying expected pre- and post-conditions for a given behavior. In many cases, the triggering conditions for the behavior layer can be used for both pre and post conditions.

Case Study: Reactive Teaming

Augmenting subsumption layers with the information required for failure detection is relatively easy. Pre-requisites and expected results can be described in the same manner as the triggering conditions already used to decide when to activate a behavior. Tracking activation history adds a constant amount of additional state to each layer. The most invasive addition is detection and handling of environmental failures, but most systems will already watch for these error states.

For individual agents, failure detection can be used to remove layers that are not functioning correctly. Malfunctioning behaviors can mask correct behavior that has lower priorities. Consider a virtual guard agent that is designed to patrol the environment. In addition, this guard agent has a higher priority layer to pick up trash. Eventually, the guard will run out of room to carry the trash it gathers, but without failure monitoring, it will get stuck trying to pick up the $n + 1^{st}$ piece of trash. With failure monitoring, the trash gathering layer will be identified as failing; since it has no space in its inventory, it can remove this layer. The patrol behavior will then continue as expected.

Insertion and removal of behaviors can be very useful in scenarios where multi-agent coordination is required. We used behavior monitoring to enable the *reactive teaming* approach to multi-agent coordination (Heckel and Youngblood 2010). In reactive teaming, agents coordinate by transferring behavior layers. Agents must decide when to request behaviors from other agents and which behavior to transfer when they receive a request. One possible choice is to make requests to replace failing layers. In addition, when transferring a behavior, the agent can choose to transfer a failing layer in case a different agent can perform it successfully.

The effectiveness of reactive teaming with failure detection can be demonstrated with a case study. In our scenario, the first agent, Alice, searched the environment for cans. The second agent, Bob, searched the environment for boxes. The goal of the agents was to pick up all of the objects scattered throughout the environment. The environment was initially loaded with many cans placed in the world. After a period, a large number of boxes were added to the environment. With static controllers, each agent would perform only the task initially designed. This means that while Alice was well-occupied for the first part of the scenario, Bob was performing very little work.

We ran the scenario in two ways. The first method used a static team. In addition to Alice and Bob, 10 more agents were added, divided evenly between can and box gathering behaviors. In this case, half of the agents were guaranteed to be idle anytime only one type of item existed in the environment. This results in poor team utilization; half of the team is idle while the other half is performing a task.

The second variation used 10 *generic* agents in addition to Alice and Bob. Generic agents have a base layer, but instead of being designed to perform a task, they are designed to have activation failures. These activation failures lead the agents to use reactive teaming to request behaviors from other characters. As the scenario started, each of the 10 agents randomly received either the behavior to pick up cans or the behavior to pick up boxes based on whether they met Alice or Bob first. Since boxes are nonexistent for the first part of the scenario, the can-gathering agents were quite productive, but the box-gathering agents continued to generate activation failures. This caused in the box-gathering agents to request new behaviors, and the number of can-gathering agents surged.

As the scenario continued, boxes were added to the world. With most of the cans gone, the can-gathering agents generated activation failures, causing them to request new behaviors. The number of box-gathering agents then surged, as the majority of agents received the appropriate behavior. Once all but a few objects have been gathered, the distribution of behaviors changed again, as all of the behaviors became idle.

Behavior insertion and removal enables the reactive teaming technique. With this technique, which requires behavior failure detection, it is possible to create flexible teams of reactive agents that adapt to changing conditions. The scenario in this case study shows an example of how using the dynamic reactive teaming approach results in better team utilization than the static approach.

Conclusions

We have presented a framework for adding failure monitoring to behavior-based subsumption architectures. Four classes of errors can be detected without significant changes to the subsumption architecture, and the required metadata also enables dynamic extensions to behavior-based subsumption. Individual agent adaptation and team coordination can be improved through the use of failure detection. Failure detection is currently used to enable the reactive teaming technique for multi-agent coordination.

References

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* 2(1):14–23.

Heckel, F. W. P., and Youngblood, G. M. 2010. Multi-Agent Coordination Using Dynamic Behavior-Based Subsumption. In *Proceedings, 6th Artificial Intelligence for Interactive Digital Entertainment (AIIDE 2010).*