The 2016 AAAI Fall Symposium Series:
Cross-Disciplinary Challenges for Autonomous Systems
Technical Report FS-16-03

Mixing Formal Methods, Machine Learning, and Human Interaction Through an Autonomics Framework

Braulio Coronado, Eric Gustafson, John Reeder, Douglas S. Lange
Space and Naval Warfare Systems Center Pacific
53560 Hull Street, San Diego, CA, 92152
{braulio.coronado, eric.a.gustafson1, john.d.reeder, doug.lange}@navy.mil

Abstract

Autonomic approaches aim to manage large complex systems by enabling self-adaptation in response to the changing state of these systems. As the size and complexity of systems increases, autonomics helps conceal that complexity and provides a higher level, more abstract view to human managers of these systems. One such autonomic approach is the Rainbow autonomics framework, developed at Carnegie Mellon University. This paper describes our use of Rainbow in various applications including supervisory control of unmanned systems and management of operator task assignment. It also describes enhancements made to the version of Rainbow utilized.

Introduction

Autonomic computing refers to software with the ability to self-adapt (Cohn 2003; Ganek and Corbi 2003; Murch 2004) in response to changes in its state. Due to the increasing complexity, size and speed at which systems change, autonomics has become an important area of research. The Rainbow autonomics framework (Garlan et al. 2004) provides a platform to manage networks by collecting data through probes and processing that data through gauges. Using probes and gauges, a model of the system is established and monitored using predefined rules. Strategies and tactics address rule violations and effectors cause changes on the managed network. Probes, gauges, strategies, tactics and effectors are generic and can be customized to autonomously manage any system that can be mathematically represented as a network. We begin with a use case detailing the structure of Rainbow. Next, we outline our usage of Rainbow to manage disparate systems. Finally, we describe ideas for future work and enhancements made to the version of Rainbow utilized in our applications.

Rainbow Architecture

A use case involves a Rainbow evaluation study managing server utilization (Cheng 2008) with the addition of simulated Disconnected, Intermittent and Limited connectivity (DIL) and energy usage metrics (Verbancsics and Lange 2013). In this experiment, the system is a set of virtualized servers providing services with a workload that varies throughout the day. We observe the individual components of Rainbow as it manages the number of active servers with respect to server response time and energy usage while experiencing increasingly degrading communication with the system.

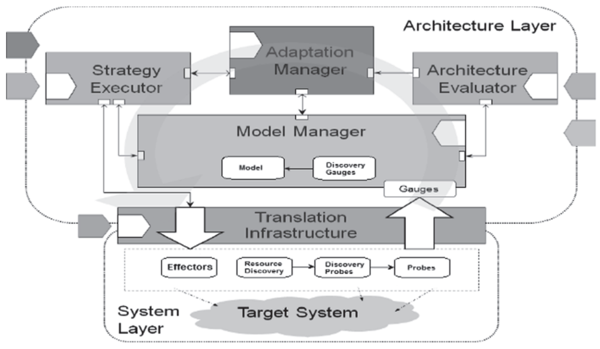


Figure 1: Rainbow consists of a Model Manager, Architecture Evaluator, Adaptation Manager and Strategy Executor which collect data and execute changes to a target system through a Translation Infrastructure in a closed loop. [used by permission of the author]

Model

A key Rainbow feature is the use of Acme, a formal architecture description language allowing the modeling of complex hierarchical systems (Garlan, Monroe, and Wile 2000). Acme enables modeling of systems through the use of components which represent the elements of a system and connectors which represent relationships between elements. Component and connector properties reflect the state of the system and rules enforce assumptions made on the model's architectural design. This formal description of the system enables validation of the model, housed in the Model Manager (Figure 1) as its state evolves. The Architecture Evaluator performs model checking as attributes within the model change. Our model in this case consists of components representing servers in the system with rules ensuring server response time and energy usage remain low.

Probes and Gauges

Probes extract data from the target system. They can exist on the system and communicate data to Rainbow's gauges or exist inside Rainbow and receive data from the target system. Gauges are internal to Rainbow and update the model through data collected from its probes. Probes in this case communicate server load and energy usage to gauges which update the respective server components. An artificial time interval of five, ten, fifteen, thirty and sixty minutes between probe reports is introduced to evaluate Rainbow's ability to manage a DIL environment. In our use of Rainbow, we have introduced machine learning algorithms into the gauges. In this use case, the system learns a model representing the pattern of operations and utilizes this model to fill in for missing reports from probes (Verbancsics and Lange 2013).

Strategies, Tactics, and Effectors

Strategies and tactics are written in Stitch, a language for the expression of adaptation decision trees (Cheng and Garlan 2012). Strategy decision trees consist of tactics with each tactic describing an action taken to address a rule violation. Actions cause changes to the system's configuration through the invocation of effectors that, like probes, may exist remotely or within Rainbow. The Adaptation Manager selects a suitable strategy and the Strategy Executor carries out the actions within the strategy. If a strategy fails in its attempt to correct a rule violation, the Adaptation Manager may select a new strategy. Strategies and tactics in our DIL example describe actions taken to address the number of active servers with respect to response time and energy usage. In periods of high activity, strategies enlist additional servers to maintain low response time. In periods of low activity, strategies reduce the number of active servers to keep energy usage low.

Autonomics in Disparate Systems

Due to the generic qualities of Rainbow, any system that can be described formally is eligible for management by Rainbow. This allows us to explore the use of autonomics in disparate use cases as follows:

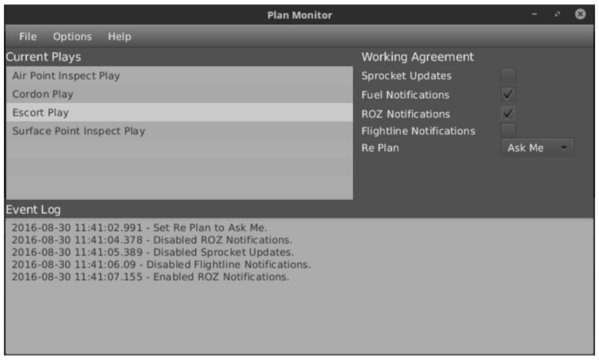
- Plan Monitor - A module within a command and control prototype providing mission related feedback and minor mission corrections.
- Task Manager - A module within a command and control system prototype assisting in the management and optimization of operator task assignment.

Command and Control Prototype

Plan Monitor and Task Manager coexist in the same system: Intelligent Multi-UxV Planner with Adaptive Collaborative Control Technologies (IMPACT), a command and control prototype controlling teams of simulated unmanned vehicles under the supervision of a single human operator (Chen et al. 2014). IMPACT provides a "playbook" as a control abstraction allowing the operator to select "plays" to manage these vehicles (Chen et al. 2014; Gutzwiller et al. 2015). An underlying goal of IMPACT is to invert autonomous vehicle staffing; from many operators for a single vehicle to one operator for many vehicles (Cummings 2015). As the number of vehicles increases, complexity of the system outpaces the operator's ability to provide effective supervisory control. Autonomics helps relieve some of this complexity by consolidating real time data into plans which are used to continuously evaluate play health. Additionally, it provides recommendations to address an ineffective play and enables global constraint monitoring. Furthermore, by modeling human operators themselves as components, autonomics can offer the ability to adjust a "sliding scale" of autonomy through operator task assignment (Gutzwiller et al. 2015).

Plan Monitor The first key to our ability to manage plans with Rainbow is the realization that our definition of a plan allows plans to be represented as networks. Therefore, we are able to model a plan within Rainbow. The second key is the generation of model elements. Clearly one does not want to manually develop a model for each plan, and establish probes and gauges to provide data on its execution. Later we describe how we generate model elements allowing us to monitor any plan developed in IMPACT. Plan Monitor's probes and gauges use the ZMQ Distributed Messaging Framework to subscribe to and process messages published to a data hub within IMPACT. These messages contain data about the simulation such as vehicle telemetry, vehicle task details, restricted zones and mission related events. Plan Monitor generates vehicle components in the model with vehicle telemetry updated in real time. Play messages are used to generate plan components enabling calculation of mission health metrics such as estimated vehicle arrival to destination and task quality level. Rules established in the model monitor global constraints such as fuel thresholds for all vehicles, vehicle presence in restricted zones and vehicle response time to flight line. Strategies and tactics publish health and constraint violations to the hub for display to the operator. Finally, a strategy publishes requests to re plan a poorly performing or at-risk play.

Task Manager Task Manager explores the use of autonomics to help manage operator workload (Gutzwiller et al. 2015). As the number of vehicles per operator increases, workload and reduced situational awareness may become a concern. We address these concerns by providing task management capability through an interactive panel in IMPACT's interface. This panel displays two queues with tasks assigned to the operator and tasks assigned to an automated assistant. Tasks are added automatically as the scenario progresses and removed as they are completed. The operator is given control over task assignment with buttons to switch tasks between queues. We are exploring the use of probes to collect operator task data such as username, login timestamps and task type, duration and priority. Using this data, gauges can compute operator task performance such as task completion and cancellation rates. Over time, we can infer more abstract metrics such as estimated time to task completion, operator attention level and ultimately, risk of handing over a task to the assistant (Lange et al. 2014). Strategies



Air Point Inspect Play
Cordon Play
Escort Play
Surface Point Inspect Play

Sprocket Updates ☐
Fuel Notifications ✓
ROZ Notifications ✓
Flightline Notifications ☐
Re Plan: Ask Me

```
2016-08-30 11:41:02.991 - Set Re Plan to Ask Me.
2016-08-30 11:41:04.378 - Disabled ROZ Notifications.
2016-08-30 11:41:05.389 - Disabled Sprocket Updates.
2016-08-30 11:41:06.09 - Disabled Flightline Notifications.
2016-08-30 11:41:07.155 - Enabled ROZ Notifications.
```

Figure 2: Plan Monitor GUI showcasing Working Agreements. The operator is granted greater control over which strategies the autonomy can use.

and tactics determine the algorithm used for task assignment based on current conditions. A strategy addresses a demanding workload by assigning more tasks to the assistant. Similarly, a strategy assigning more tasks to the operator addresses a light workload in an effort to maintain operator situational awareness.

Rainbow Enhancements and Future Work

Machine Learning

In our DIL use case mentioned earlier, we observed Rainbow's reduced effectiveness with the introduction of intermittent communication with the target system. Rainbow relies on the current state of the system to evaluate and effect changes in a timely manner. One area we have explored is the use of machine learning to predict system values when communication with the system is degraded. Instead of acting on system data as it is reported, our system acts on a combination of real and predicted data while evaluating and refining its predictions (Verbancsics and Lange 2013).

Component Generation

In a target system backed by object-oriented software, we can dynamically generate Acme components by analyzing an object's structure at runtime and describing its structure as Acme architectural designs. In Plan Monitor, the system it manages is written in Java, an object-oriented programming language. Plan Monitor component generation functions by utilizing the Java reflection API for runtime object analysis. Object metadata is mapped to equivalent structures in the Acme model resulting in a real time modeling of the system. Currently, we must express the architectural design of our models statically, using prior knowledge of our system structures to establish rules. Since Acme allows for the creation of rules at runtime and since we have a complete picture of our system elements due to component generation; future work could explore the generation of rules at runtime. This may allow us to modify our architectural design assumptions dynamically.

Working Agreements

To promote collaboration and transparency between human-autonomy teams, a working agreement concept is used to establish how and when autonomy is allowed to take action. We are exploring a configurable working agreement to allow the operator configuration of policy per strategy. By adhering to a working agreement, an operator is granted greater control over the autonomy. The motivating factor behind this effort is the fact that the operator might have knowledge about the system that the automation does not. Thus, the ability for the operator to control the autonomy's execution of certain strategies as needed is valuable. Plan Monitor features a prototype working agreement allowing the operator to restrict execution of a strategy, allow execution of a strategy without approval or allow execution but require approval (Figure 2).

Summary

In this paper, we have described the blending of a formal methods based autonomics framework with machine learning algorithms to enhance its operation. Applying object-oriented instantiation to components of the framework then allows us to utilize Rainbow to manage such disparate networks as plans or human-autonomy teams supervising the operation of autonomous vehicles.

References

Chen, J. Y.; Procci, K.; Boyce, M.; Wright, J.; Garcia, A.; and Barnes, M. 2014. Situation awareness-based agent transparency. Technical report, DTIC Document.

Cheng, S.-W., and Garlan, D. 2012. Stitch: A language for architecture-based self-adaptation. *Journal of Systems and Software* 85(12):2860–2875.

Cheng, S.-W. 2008. *Rainbow: Cost-Effective Software Architecture-Based Self-Adaptation*. Ph.D. Dissertation, Carnegie Mellon University, PittsBurgh, PA.

Cohn, D. L. 2003. Autonomic computing. In *Proceedings of the The Sixth International Symposium on Autonomous Decentralized Systems (ISADS'03)*, 5. Pisa, Italy: IEEE Computer Society.

Cummings, M. 2015. Operator interaction with centralized versus decentralized uav architectures. In *Handbook of Unmanned Aerial Vehicles*. Springer. 977–992.

Ganek, A. G., and Corbi, T. A. 2003. The dawning of the autonomic computing era. *IBM systems Journal* 42(1):5–18.

Garlan, D.; Cheng, S.-W.; Huang, A.-C.; Schmerl, B.; and Steenkiste, P. 2004. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer* 37(10):46–54.

Garlan, D.; Monroe, R. T.; and Wile, D. 2000. Acme: Architectural description of component-based systems. *Foundations of Component-based Systems* 68:47–68.

Gutzwiller, R. S.; Lange, D. S.; Reeder, J.; Morris, R. L.; and Rodas, O. 2015. Human-computer collaboration in adaptive supervisory control and function allocation of autonomous system teams. In *International Conference on Virtual, Augmented and Mixed Reality*, 447–456. Springer.

Lange, D. S.; Gutzwiller, R. S.; Verbancsics, P.; and Sin, T. 2014. Task models for human-computer collaboration in supervisory control of teams of autonomous systems. In *2014 IEEE International Inter-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)*, 97–102. IEEE.

Murch, R. 2004. *Autonomic Computing*. IBM Press.

Verbancsics, P., and Lange, D. S. 2013. Using autonomics to exercise command and control of networks in degraded environments. Technical report, Eighteenth International Command and Control Reserach and Technology Symposium, Alexandria, VA.