AFRL-RI-RS-TR-2020-109

AN UNCERTAINTY-AWARE APPROACH TO CERTIFYING SECURITY ASSURANCE FOR AUTONOMOUS SYSTEMS

THE UNIVERSITY OF TULSA

JUNE 2020

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2020-109 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
WILMAR SIFRE
Work Unit Manager

/ S /
GREGORY HADYNSKI
Assistant Technical Advisor
Computing & Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
JUNE 2020	FINAL TECHNICAL REPORT	NOV 2018 – DEC 2019

4. TITLE AND SUBTITLE

AN UNCERTAINTY-AWARE APPROACH TO CERTIFYING SECURITY ASSURANCE FOR AUTONOMOUS SYSTEMS

5a. CONTRACT NUMBER
FA8750-19-2-0002

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62788F

6. AUTHOR(S)

Rose Gamble and Betty Chung

5d. PROJECT NUMBER
T2EE

5e. TASK NUMBER
TU

5f. WORK UNIT NUMBER
LS

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

PRIME
The University of Tulsa
800 S. Tucker Drive
Tulsa OK 74104

SUB
Michigan State University
3115 Engineering Bldg
East Lansing MI 48824

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RITA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2020-109

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This report details our approach to enhance the trustworthiness of autonomous systems that need to adapt at run time to changes resulting from unexpected conditions, component failures, and enemy attacks. The primary objective of the project is to develop technology to express, codify, and automatically maintain operational system security assurance cases (SACs) for autonomous systems operating under uncertain conditions. Assurance cases have been accepted as a means for certifying the utility and satisfaction of trustworthiness, safety, and mission objectives. SACs will capture the trustworthiness and risk assessment chain of evidence from initial development through system evolution during runtime adaptation in response to environmental and system uncertainty with respect to security threats. The technologies will be demonstrated and evaluated on platforms of different scales using two case studies: 1) adaptive cruise control for autonomous rover and 2) integration of interaction-enabled testbeds to mimic search and rescue robot operation in disastrous situation.

15. SUBJECT TERMS

Security assurance cases, compliance confidence, risk assessment, self-adaptation, self-healing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**WILMAR SIFRE**
U	U	U	UU	67	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

Table of Contents

List of Figures

List of Tables

1 Executive Summary

Re-verification of critical properties, such as mission-critical and safety-critical properties, is a core aspect of deploying resilient autonomous systems that can perform self-repair at run time. Such systems must be able to adapt to most, if not all, situations that impede mission or safety objectives. Critical properties can be divided into functional and security requirements, though they may be interrelated. If an autonomous system is forced to adapt to complete its mission and that adaptation is dynamically crafted at run time, then confidence in the adaptation's compliance with functional and security requirements should be established prior to changing the system's operation. Moreover, for security threats determined as part of system situational awareness, the expected level of confidence must be coupled with defensive resilience. Traditional assurance techniques, such as testing and formal analysis of system reconfiguration (e.g., code repair or reused code), may not be available at run time due to practical performance constraints.

Maintaining a high degree of confidence that an autonomous system is compliant with its requirements while defending against security threats poses multiple, significant research challenges. To address these research challenges, we expanded our prior work on security assurance cases (SACs) (Jahan, 2018a) and developed and evaluated technology that supports expressing, codifying, and automatically maintaining SACs for autonomous systems. The objective is for the technology to be available as systems perform reconfiguration at run time to achieve mission objectives resiliently and securely in the face of adverse conditions. Assurance cases have been accepted as a means of certifying the utility and satisfaction of safety, mission, and trustworthiness objectives (Rushby, 2015). SACs were first mentioned by Goodenough (Goodenough, 2007) and consist of a structured collection of security-related claims, arguments, and evidence. Security-related claims state that the system is acceptably secure given effective security capabilities that mitigate security vulnerabilities. Argumentation of a claim describes how each claim is supported by objective evidence manifested by the security capabilities present in the system. Our SACs are transformed to capture the trustworthiness and risk assessed chain of evidence from initial development through system evolution during run-time adaptation in response to security threats (Jahan, 2018a). Given the potential need for changes to the physical and cyber system functionality due to environmental uncertainty, SACs should be embedded in the system to be dynamically updated to reflect the current security profile. In turn, security mitigations due to attacks or detected vulnerabilities may trigger subsequent system adaptations. We have explicitly addressed the dual and interacting (functional vs. security) updates of the autonomous system and the supporting SACs with respect to system resilience, risk management, and assurance traceability.

Given the complexity and independent properties of the prior efforts on monitoring, planning, and risk assessment, the results of this effort underscore and support the need to design technology for integration. With this technology, the model transformation and translation efforts are reduced, while the implementation efficiency and adaptation traceability are increased. The developed MAPE-K/MAPE-SAC framework supports the construction, analysis, and updating of SACs for

autonomous systems in the face of uncertainty. Both MAPE loops in the framework involve the monitor, analyze, plan, and execute phases, differing only in their adaptation target. A key innovation is the MAPE-SAC loop (Jahan, 2019b), as inspired by the MAPE-K loop that serves as the foundation for the feedback loop to manage autonomous systems. For the MAPE-K loop, the target is the autonomous system. For the MAPE-SAC, the target is the collection of SACs. We have retained the prior evolutionary aspects of the SACs (Jahan, 2019a) and introduced a new interaction protocol within the MAPE-K/MAPE-SAC framework to ensure that functional and security properties are satisfied with optimal utility (Jahan, 2020a). In addition, we have evaluated 3rd party tool integration into the framework (Jahan, 2020b).

2 Introduction

Modern autonomous systems require the ability to adapt their behavior to preserve their operational capability while facing uncertainty (Kephart, 2003). These adaptations must undergo a rigorous process of assuring that the system complies with functional constraints at a desired level of confidence. In environments where the system is subject to significant performance constraints, such as limited computational power or high network latency, the ability to perform dynamic assurance of adaptation operations may not be feasible. Dynamic assurance can also become infeasible if the system must comply with operational standards while simultaneously mitigating security threats, especially where conflicts emerge between functional and security constraints. When dynamic assurance is feasible, it is necessary to examine prior methods of security certification and perform risk assessment to show that the adapted system maintains compliance with its diverse set of requirements.

Security certification is a highly manual effort that documents the process and artifacts that provide a level of confidence in the system's ability to comply with its security requirements. When the system configures a new component, patch, or decision-making strategy at run time, there can be a direct and propagated impact on security compliance. Thus, run-time adaptations must be risk-assessed against both functional and security requirements. Prior work in this area must be extended to accommodate new system designs that embed adaptation and security frameworks, so that the adaptations can be holistically examined from architectural, verification and validation (V&V), and certification and accreditation (C&A) perspectives.

Assuring trustworthiness is a prime concern for self-adaptive and self-healing systems, since they must manage uncertainties in a dynamic environment (Alexander, 2015), (Yuan, 2014), (Moyano, 2013). Uncertainty may arise due to faulty hardware and software components, malicious attacks, and unpredictable environmental and/or functional requirement changes. Effectively managing the trustworthiness of the system within uncertain situations requires the capability to detect and mitigate security threats at run time and deploy autonomous, self-protecting mechanisms whose primary objective is to adapt security functionality accordingly (Yuan, 2014). The self-protecting mechanisms should maintain confidence in the security compliance of the system. The functional behavior of the system must be codified separately from

and prior to its deployment into the system to enable trustworthiness and risk assessment across both functional and security requirements.

Like self-adaptive systems, self-protecting systems should include a MAPE (Monitor, Analyze, Plan, and Execute) control loop specifically for managing security concerns. However, adapting security functionality may hinder the system's ability to maintain its other quality concerns (Le, 2015). While a self-adaptive system automatically establishes behavioral changes to preserve its functional objectives, changing functional behavior as a part of an adaptation may result in security vulnerabilities. Thus, the adaptation decisions of self-protecting systems need to consider compliance with both functional and security concerns. The system must be able to pinpoint and provide an alert regarding what and where the adaptation impact is given the system architecture as it relates to the assurance of mission completion and security constraints. Representing functional and security requirements from an assurance case perspective (Rushby, 2015) can facilitate comparable assessment of adaptations to address both types of constraints, their interactions, and their impacts.

One approach to achieve confidence in a system's security compliance is to certify the system according to a set of prescribed security controls based on the National Institute of Standards and Technology NIST SP800-53 framework (NIST, 2015), which is considered to cover the international ISO/IEC 27002 standard (ISO, 2013). Security controls assert strategies to minimize the security threat to a system. They are expressed using a hierarchical statement structure with control enhancements that provide more detailed compliance requirements. For certification, evidence must be provided that supports the existence of mechanisms that assure security control effectiveness within the system. The certification process captures all assets of an organization and its targeted system, including software, data, physical media, and computing resources. Re-certification of security compliance for traditional systems is performed on a periodic basis. Since adaptive systems are executed in a dynamic environment, assessing the level of compliance with security controls at run time due to a system change must still address how functional constraints are maintained. Because protecting a system from threats can involve both security and functional concerns, maintaining a certain confidence level in the system's requirement compliance following an adaptation is challenging (Kephart, 2003). Specifically, static solutions are insufficient since functional and security conditions should be allowed to change at run time. Dynamic solutions without coordination between functional and security expectations can lead to an increase in overall requirement violations since functional adaptations can violate security requirements and vice versa.

Any appropriate solution requires a complex configuration, management, and analysis process as part of a framework to detect and mitigate uncertainty across regions of concerns. Requiring a single, centralized MAPE (Monitor, Analyze, Plan, and Execute) control loop to handle different components or concerns can be limiting. An alternative is to employ multiple, interacting and decentralized MAPE loops (Weyns, 2013), (Vromant, 2011). The MAPE-K (Monitor, Analyze, Plan, and Execute with Knowledge) loop was designed to manage the adaptation of autonomic systems (Kephart, 2003). Coupling this control loop with how researchers have created SACs to

assess system security status (Alexander, 2015), (Lipson, 2008), (Goodenough, 2007), (Jahan, 2018a), (Jahan, 2019b), we introduce a MAPE-SAC control loop to address security compliance by extending functional adaptation methodologies to security controls, (Jahan, 2019a). The Security Assurance Case (SAC) is a structured set of arguments with evidence to assure system satisfiability to specific security claims, such as NIST security controls (NIST, 2015). Using the MAPE-SA, we have developed a framework to manage the interaction and adaptation of functional compliance with security compliance in self-adaptive systems.

Prior work introduced a template (Jahan, 2018a) to model SACs using Goal-Structured Notation (GSN) (GSN, 2018), (Kelly, 2004) that reflects security control definitions and arguments for their compliance. The template was extended to show the potential for defining and applying adaptation operators that can be deployed on an SAC (Jahan, 2019a) and the impact that changes have on compliance confidence levels. The objective of the framework is to package template instantiation with NIST security controls (NIST, 2015), adaptation operations, and SAC evaluation processes.

Our framework can manage the interaction between a traditional MAPE-K loop overseeing functional behavior changes and a MAPE-SAC loop overseeing security behavior changes that jointly ensure acceptable, adaptive behavior. We use the term “region” to refer to a conceptual separation of concerns (i.e., functional vs. security). Based on previously defined MAPE interaction patterns (Vromant, 2011), (Weyns, 2013), we separate the functional region of the system from the security region, assuming each is intended to manage adaptations that affect its concerns and constraints. We implemented a hybrid of the Regional Planning Pattern and the Coordinated Control Pattern (Vromant, 2011), (Weyns, 2013) to specify the interaction between the MAPE-K (functional region) and MAPE-SAC (security region) control loops. Control loops over separate regions coordinate to select and implement system-wide adaptations. To manage the interaction, we introduced a MAPE loop interaction protocol to support the coordination between the MAPE-K and MAPE-SAC loops. Our MAPE-K and MAPE-SAC coordination approach selects a system-level adaptation that minimizes compliance degradation across functional and security requirements. Coordination is conducted during planning and execution to ensure adaptive system behavior with optimal utility while maintaining security compliance. Multiple testbeds were used to instantiate functional assurance cases (FACs) and SACs to illustrate the MAPE loop interactions.

3 Methods, Assumptions, and Procedures

3.1 Perspective and Objectives

For security threats, determined as part of system situational awareness, the expected level of operational trustworthiness must be coupled with defensive resilience. Assessing the tradeoffs between trustworthiness and defending against security threats in an autonomous system means that (1) trustworthiness must be quantified so that its level can be gauged to be within an acceptable threshold, (2) adaptations configured by the system at run time must be risk-assessed against mission-critical and security requirements, and (3) the effect of the adaptation on the system architecture and behavior must be codified separately from and prior to its deployment into the system to allow for trustworthiness and risk assessments at run time.

To meet (1) – (3) above, the overall objective of the research effort was to develop a MAPE-K/MAPE-SAC framework to express, codify, and automatically maintain operational confidence in a system's compliance with its security requirement using adaptive SACs. SACs are structured to capture the trustworthiness and risk assessed chain of evidence from initial development through system evolution due to run-time adaptation. Such adaptations are in response to environmental and system uncertainty that may lead to security threats. The research challenges addressed by the framework were proposed as addressing the concepts of:

- Lightweight modeling, deploying, and evolving of SACs.
- Quantifying the risk of adaptation in response to security threats to mission objectives.
- Supporting the interaction between MAPE loops to dynamically manage run-time adaptations in response to changes in functional and security conditions.
- Interfacing with continuous monitoring and adaptation generation techniques.
- Dynamically evolution assurance cases as run-time adaptations are performed.

This research focused on bridging the gap between the lack of security constraint certification activities and processes to accommodate cyber resiliency and the need for run-time adaptive systems to maintain a level of trust when a security threat jeopardizes the operational mission.

3.2 Testbed Integration

To implement and evaluate the MAPE-K/MAPE-SAC framework, we established the following testbeds on which we specified and experimented with a variety of use cases.

3.2.1 The Wearable Security Experimentation Testbed

The wearable security experimentation testbed uses Raspberry Pi 3s[1] to simulate near-future (computationally powerful) wearables, allowing the wearables to intercommunicate with Bluetooth[2] and self-adapt their communication when faced with potential security vulnerabilities (Walter, 2018a). The data that a wearable collects is sent at the time it is collected, assuming the data collection rate (approx. 1kB/sec) is slower than Bluetooth communication speed (approx. 1.4 Mb/sec). The Raspberry Pis simulate three wearables based on their generic, operational requirements: a heart rate variability (stress) monitor, advanced headphones, and an insulin pump (See Figure 1), though other wearables may be simulated as needed (Walter, 2018b). The program for each wearable has been verified to meet its operational requirements under normal or default environmental factors. A meta-model of the workflow containing state variables and components that were evaluated as part of the requirements verification process has been embedded into the planner of each wearable's MAPE-K control loop. When monitoring and analysis determine that an adaptation is needed, the planner uses the embedded verification workflow (Marshall, 2018a) for each pre-defined wearable requirement, encoded as a Colored Petri Net (CPN) (Jensen, 2009), to choose the least risky potential adaptation. The process relies on a higher level of abstraction than direct model-checking or runtime verification to reduce the complexity in dynamically assessing adaptation risk to the system (Marshall, 2018b).

One goal of devising the wearable testbed is to determine how to increase awareness of environments that may be a security threat. These environments can be identified either by a user recognizing an eavesdropping device, changes to expected data potentially resulting in a Man-in-the-Middle attack, or a sudden lack of data in an area, indicating a DoS attack. If an adaptation is planned, the user is alerted to the plan, providing recognition of potential insecure environments. Another goal is to exploit social situational awareness to communicate potential security threats. Thus, each base station and wearable run an application that determines the least risky of four potential adaptations to address a security threat using its embedded verification workflow (Walter, 2018b). The application employs a communication protocol for a wearable or base station to communicate the presence of a security threat to other nearby base stations or peer wearables so that they too can adapt. The current available adaptations are (1) staying connected but sending empty packets, (2) disconnecting and awaiting secure reconnection, (3) staying connected, sending empty packets and communicating a security threat alert to a base station or peer wearable (called *fostering*), and (4) disconnecting, awaiting secure reconnection, and communicating a security threat alert to a base station or peer wearable (i.e. fostering) (Walter, 2018b).

[1] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[2] https://www.bluetooth.com/

3.2.2 The Mission Coordination Testbed

Our second testbed focuses on mission collaboration between autonomous Cozmo[3] robots. Cozmo is a small robot capable of roaming a small area, recognizing cubes and faces, and performing simple, preset tasks. It has a Software Developing Kit (SDK) for developers to program additional functionality into the robots. For example, we have added communication between multiple Cozmo robots to experiment with autonomous mission collaboration and adaptation (Jahan, 2018b). Each robot can adapt its local plan to more efficiently complete a coordinated global mission. The robots are aware of the global mission but are unaware of the local mission of the other robot. In this testbed, there is no explicit MAPE-K control loop representation to institute an adaptation. Instead, conflicts between the local plans, global plans, and intercommunicated activities institute the need for local adaptation to address plan conflicts.

The testbed relies on Raspberry Pi 3s that are physically tethered to Android[4]devices running the Cozmo SDK. Thus, there is a one-to-one relationship between a Raspberry Pi and Android device. The Android device is dedicated to running only the app that communicates through Wi-Fi to a single Cozmo robot, forming another one-to-one relationship. The testbed uses Partial-Order, Causal Link (POCL) (Cox, 2005) plans to formally describe the local mission plans and to determine plan conflicts that can impact the mission. Assurance cases with claims and evidence for their support based on the formal verification of the POCL plans are embedded into the Raspberry Pi programming to assess the impact of an adaptation on the claims. The robots act autonomously, communicating their mission plan and changes to that plan to the Android device, which causes Bluetooth communication between the dedicated Raspberry Pis (base stations). Messages are then pushed to the other Cozmo (see Figure 1) at a rate of approximately 20 times per second (380 B/sec). This communication strategy allows the robots to self-adapt their local missions based on the actions of the other robot or their own sensor data to ensure the global mission is completed (Jahan, 2018b).

3.2.3 Integrating the Testbeds

Figure 1 shows the proposed architecture of a system with multiple intercommunicating, but distinct testbeds (Walter, 2019). On the left are the self-adaptive, coordinating Cozmos, focused on completing both the global and their local mission(s). They interact with the wearable testbed on the right through a Bluetooth link between base stations, which is similar to their normal communication with each other. The direct communication between testbeds through Bluetooth requires each testbed to parse data that it is not designed to accept and that has become a hindrance to integration. To resolve these issues, we shift the testbeds to rely on the cloud to capture relevant testbed data for sharing. We have employed Artificial Intelligence (AI) and machine learning

[3] https://www.anki.com/en-us/cozmo
[4] https://www.android.com/

techniques to discover potential adaptations and rationale for their use to influence the testbeds toward self-improvement, which may even impose additional requirements that are subject to verification and validation. Currently, each testbed makes use of the cloud in different ways. The wearable testbed relies on cloud-based machine learning to discover insecure environments, prompting adaptation. The Cozmo testbed uses the cloud to primarily store information about the current mission and facilitate coordination, including potential adaptation needs. Testbed integration requires a cloud that both stores information and makes use of cloud-based AI and machine learning algorithms to understand the unique adaptation needs for both testbeds.

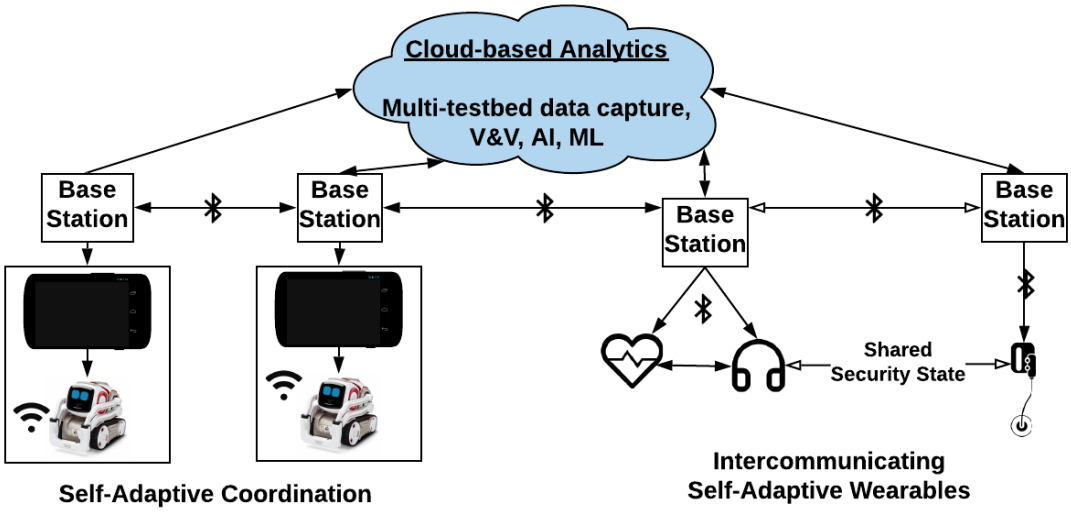

Figure 1: Architecture for integrating testbeds

If an adaptation in one testbed translates to the need for adaptation in the other, cloud services can intervene to prevent an infinite adaptation loop, testbed deadlock by forcing a change at the wrong time periods, or the other testbed from moving into a state that violates its requirements.

We describe three potential use cases originally used to experiment with integrating the two testbeds (Walter, 2019). Each use case introduces functional adaptations that we reuse to perform SAC assessments, adaptations, and trade-offs analysis to exercise the MAPE-K/MAPE-SAC framework. The first use case is a scenario where an adaptation to a Cozmo's local mission is needed to complete the global mission. However, in order to complete its local mission, additional sensor data is needed from other nearby sensors. In this case, the adaptation plan is shared between the Cozmo testbed and the wearable testbed along with a request for sensor information through the cloud. The base station of the wearables responds with information about the available and active sensors. If a needed sensor is available and active, the Cozmo testbed is able to request the specific sensor information it needs and the data is shared. The cloud service relies on AI and machine learning algorithms to determine the appropriate sensor information from the wearables, providing the format the wearables use for data communication to the Cozmo testbed and, if needed, translating this information to the Cozmo testbed in a format it can use.

An obvious example of this use case is using the Cozmo to simulate a search and rescue robot (in the "small"), such as searching for survivors in a disaster situation. The Cozmo can connect to

a nearby base station, indicating that there may be a survivor in the area. When connected, it adapts to complete the global mission of finding survivors rather than continuing its local mission of searching a prescribed area. It then requests sensor information from the base station. The base station may be connected to wearables that include heartrate (stress) or other health information. The base station can be expected to have Global Positioning System (GPS) data. This data is useful for the Cozmo's current adapted mission, so it requests heartrate and GPS data though the cloud. The heartrate information can be used to get a general sense of the health of the survivor and the GPS data can be used to pinpoint the location of the survivor for rescue. The machine learning algorithms running on the cloud can be trained to prioritize users whose health data shows that the user needs immediate medical attention or whose data shows increased health risk. While this does not result in ignoring survivors, it does allow survivors to be rescued in priority order, potentially resulting in a larger number saved.

A second use case is where the testbeds can be jointly used to assess adaptations constructed and recommended by the cloud service to improve communication between devices. The wearable testbed already includes adaptation options that result in adjusting the communication between wearables and their base stations. Thus, it is reasonable that an adaptation on the wearable testbed could influence the cloud-based decisions to change something about the internal or external communication with the Cozmo testbed. The two primary adaptations of the wearable testbed on Bluetooth adaptation are sending empty packets (to prevent eavesdropping on sensitive data) and disconnecting and awaiting a secure reconnection. In both of these cases, communication with the Cozmo testbed is interrupted in some way. In the first case, the Cozmo testbed remains connected, but is unable to receive any data from the wearables, as only empty packets are sent. In the second, the testbeds disconnect and reconnection only occurs once the wearable testbed requests it. These communication changes may force a new adaptation to the Cozmo as to how it can be interrupted when communication and data transfer are reestablished.

In addition, the adaptation plans are sent from the wearable testbed to the Cozmo testbed to inform the Cozmo testbed of the expected change to the communication. Once the Cozmo testbed is aware of the adaptation, it does not attempt to request data from the wearables, potentially limiting the available adaptations for the Cozmo but ensuring the integrity of the wearable adaptations. In the case of disconnection, the Cozmo testbed does not attempt to force a reconnection and does not accept a reconnection request from any device other than the wearable testbed that originally caused the disconnection. However, as these adaptations only apply to the Bluetooth communication, it is still possible to gain information from the cloud service, though this information will only allow old data, not up-to-the-minute information since current data is not being streamed to the cloud service.

The third use case for testbed integration occurs when an adaptation is required by one testbed based on the expectation of an adaptation in the other testbed. For example, a Cozmo may be simulating an autonomous vehicle (in the "small"), using the wearable testbed to simulate pedestrians and other vehicles in a simulated Vehicular ad-hoc Network (VANET). In this case, the Cozmo may detect the potential for a collision with another vehicle, requiring it to adapt to the

issue. This adaptation needs to be transmitted to all other autonomous vehicles in the area. It may be beneficial to transmit this information to pedestrians so they can react, if possible. In this case, the Cozmo sends its plan to the wearable testbed before adapting. The wearable testbed then adapts its operation to alert pedestrians to the issue and adjust the sensor information to react to the new Cozmo plan. In this case, a cloud service may be used to alert those not in the area, such as emergency personnel, if there is potential danger with the adaptation, providing a faster response time to a collision.

3.2.4 Difficulties with Testbed Integration

There are a number of challenges to integrating multiple testbeds. The largest issue is ensuring there is an existing and consistent protocol for the request and transfer of data between the testbeds (Walter, 2019). It is especially problematic for Bluetooth communication, as it requires the protocol to be run and interpreted locally. Currently, the best option is to ensure that the receiver requests only specific data that it knows the other testbed has. Integrating additional testbeds becomes more difficult as each new testbed requires all previous testbeds to be updated to accept the data of the new testbed.

An additional challenge is addressing the need for a method to communicate adaptation plans that can be parsed and examined by other testbeds. For example, with the current systems a message must be sent to inform all testbeds about an adaptation to ensure that the adaptation does not negatively affect the other testbed(s). However, this sharing requires all testbeds to have methods of handling all possible adaptations of all other testbeds. An intermediate system must be designed to allow a testbed to have an understanding of the more global effects of the adaptation. One assumption is that a cloud service could serve as the intermediary, providing an understanding of the possible adaptations each testbed can perform that can be propagated through the interconnection. These challenges are the subject of ongoing research at the University of Tulsa.

3.2.5 Working with Cozmo and Robot Operating System (ROS)

Cozmo robots, by default, are controlled through a direct connection to an Android or iOS[5] device running the Cozmo SDK. The Cozmo SDK, while ideal for use in teaching students about computer science, includes some oddities that make experimental research difficult. For example, when telling the Cozmo to go to a specific location, it will choose one of a handful of animations for its movement. This could be a straightforward movement, circle to the other side of the location, wiggle around, intentionally miss the location and retry, or, in very rare cases, ignore the command and move to a different location. In all these movement cases, once a command is received, Cozmo ignores any potential obstructions in its chosen path. It cannot be interrupted until its designated objective is completed. This necessitated forcing the Cozmo to either move in very small areas and constantly re-check its surroundings or limit the mission to a problem where the Cozmo is unlikely to encounter obstructions on the majority of paths it may choose. Our earlier

[5] https://www.apple.com/ios/ios-13/

experimentation of mission coordination with Cozmos did not have this issue because the local plans were crafted with small movements. However, for more complex problems, we needed the ability to dynamically adjust a path to avoid obstructions. Additionally, because the Cozmo SDK is unique to the Cozmo, porting our results to another autonomous systems, such as the Michigan State University's AutoRally car (Langford, 2019a) would not have been doable.

To address the portability issue, we shifted the control of Cozmo from custom code using the Cozmo SDK to the more generalized ROS[6]. ROS is a Publisher/Subscriber communication protocol (Birman, 1987), focused on creating communication channels between the control code and the robot being controlled. Each different type of robot has a unique ROS compatible driver that creates Publishers and Subscribers in a generic fashion, allowing control code to be used on any robot with the same sensors. For example, code can be taken from controlling an autonomous car and used on a smaller robot with a completely different design, provided both have the same or similar sensors and speed controllers. Even in the event that there are small differences, full rewriting of the control code is often not needed as many of the differences come down to issues like different naming conventions or data ranges. Thus, the use of ROS provides a solution for examining robotics problems at a small scale, with less expensive, and less dangerous robots and then testing the resulting code at a larger scale.

Unfortunately, ROS does not have default drivers for controlling the Cozmo robot. There have been attempts by others to develop a driver by creating a Cozmo SDK compliant driver that utilizes ROS Publishers and Subscribers for control and data collection, but they are incomplete and error prone. For example, we had to modify a driver that publishes speed data so that we could ensure that all of the Cozmo functions are publishing to the correct channels (as per ROS recommendations) and that all data is correctly formatted. We have successfully performed simple functions using a Cozmo controlled by ROS (such as driving forward to collect a cube and avoiding cubes) as a proof of concept and have begun work on integrating the ROS controlled Cozmo into the mission coordination testbed, removing many of the issues the Cozmo SDK has. For example, with ROS, it became possible to request the Cozmo to move in a specific path without telling it to go to a location, allowing interruption and re-planning in the event of an obstruction.

Another advantage of using ROS is its built-in simulator, Gazebo[7]. Gazebo is designed to simulate an environment and robots as accurately as possible, with realistic physics and detailed control over a robot's joints and sensors. Michigan State University researchers have used Gazebo with Evo-ROS (Langford, 2019a) and Enki (Simon, 2018) to examine setting the values of wheel-speed sensors and locations and angles of object sensors on autonomous vehicles. With Gazebo, it is possible to test code for a robot in simulation before shifting to testing on a real-world robot. Combining this simulation with Evo-ROS and Enki allows enhancements to the control of the robots or the design of the world to ensure that the robot is as resilient as possible, or to determine some set of unique and un-planned adaptations to address potential sources of uncertainty.

[6] https://www.ros.org/
[7] http://gazebosim.org/tutorials?tut=ros_overview/

Because Cozmo had no default ROS driver, it had also not been fully modeled in Gazebo. There existed only one created model of the Cozmo and its cubes. Using this work, we focused on Gazebo simulations of the Cozmo, controlled by ROS, to complete a simple mission to collect cubes. This is the same mission originally used by the mission coordination testbed (Jahan, 2018b). Figure 2 shows the simulated Cozmo moving to complete this mission within Gazebo, controlled by ROS. Because the control for the simulated Cozmo in Gazebo is identical to the control for the physical Cozmo, any code developed for the simulation is directly portable to the physical Cozmo for experimentation in the real world. Additionally, because of the generic nature of ROS, any code developed on the simulator is also directly useable on any robot with tank drive for movement and a camera sensor.

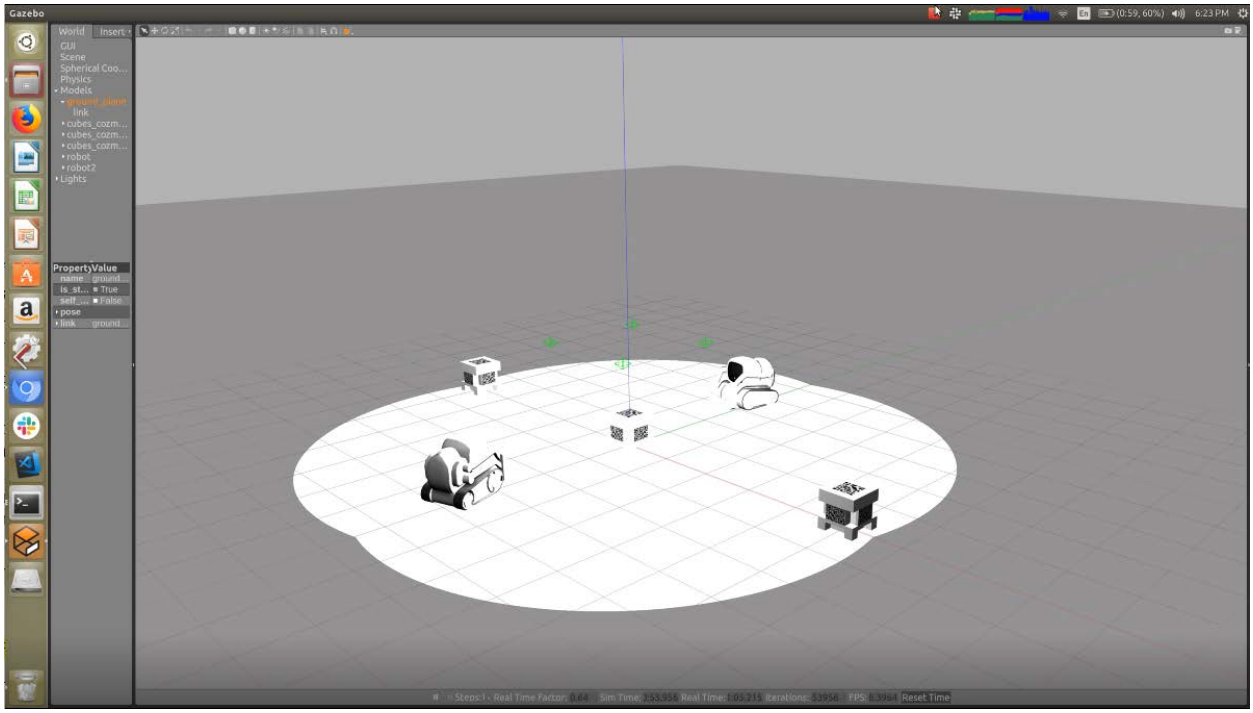

Figure 2: Simulated Cozmo completing mission coordination

3.3 MAPE-SAC: A Framework to Dynamically Manage Security Assurance Cases

The MAPE-SAC framework is motivated by the need for security-based, self-protection of autonomous systems. The objective is to monitor run-time compliance with security controls using the SACs and adapt SACs as appropriate in coordination with functional adaptation needs as dictated by the MAPE-K loop. The MAPE-SAC framework could be used to evaluate adaptation plans across a network of related SACs. This evaluation enables traceability of the system's security profile evolution (Jahan, 2018a). Analogous to the MAPE-K loop, the MAPE-SAC loop (Figure 3) comprises four main steps: monitor, analyze, plan, and execute. It relies on resources tailored to security requirements and the required confidence level of system compliance to maintain a threshold of acceptable risk.

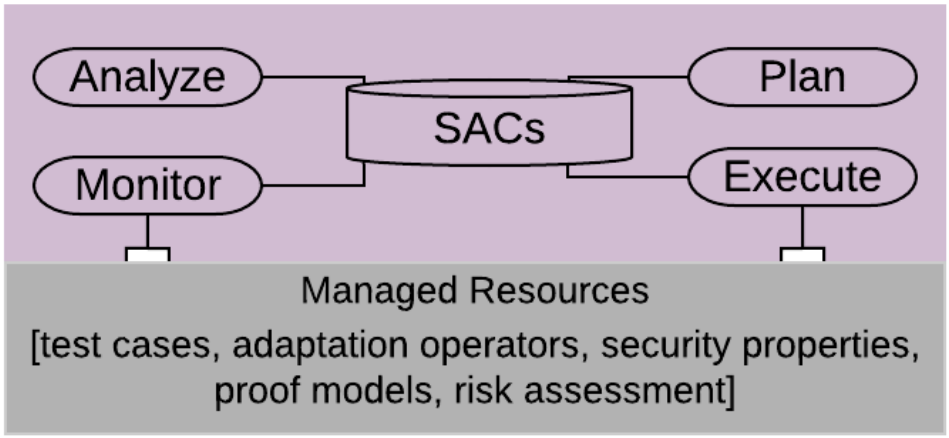

Figure 3: MAPE-SAC loop

3.3.1 Monitor

The monitor step monitors security control effectiveness in the system, which includes functional and environmental changes. SACs used in this work reflect security controls and the certification process guidelines outlined in NIST SP800-53A (NIST, 2014) for their goal satisfaction. Prior work defined a template for this form of SAC using GSN (Jahan, 2018a) by extracting structural patterns of security control statements. The template is instantiated per security control. Achievement weights associated with subgoals can be calculated to indicate the level of confidence in satisfying the main goals.

3.3.2 Analyzing

The security control's effectiveness can be assessed in the analyze phase, which may result in detecting a security threat that impacts a SAC goal or propagates through the SAC network of related controls (Jahan, 2019a). Analysis must distinguish a functional anomaly from a security threat to ensure the selection of the appropriate adaptation for that concern.

3.3.3 Planning

System reconfiguration cost and level of security confidence maintained are the main planning considerations. Thus, the planner dynamically evaluates potential adaptations by assessing the effectiveness of security controls over the network of SACs. Collaborative planning between MAPE-SAC and MAPE-K loops (Section 3.5) enables system-wide adaptation.

3.3.4 Executing

During execution, the system is reconfigured to the new adapted state that requires one or more SACs to be updated. Adaptation operations and their effects on SACs based on pre-knowledge have been examined by Jahan et al. (Jahan, 2018a).

3.3.5 Example of Certification and Modelling of SAC

The NIST SP800-53 (NIST, 2015) has become the de facto standard for security compliance best practices to protect information confidentiality, integrity, and availability. The document details 18 security control families that house security requirements with which companies working with the US government must demonstrate compliance. We define the security controls as system security constraints. A self-adaptive, self-protecting information system complying with designated security controls means (1) compliance must be guaranteed at an expected confidence level, (2) mechanisms deployed to enable security controls should not be deleted as part of an adaptation, and (3) the system should have awareness of its security controls, mechanisms enabling their effectiveness, and dependencies between controls to reduce conflict and change propagation effects.

Table 1: SI-7(5) DEFINED BY NIST SP800-53A REV. 4

SI-7 Control Enhancement (5) Software, Firmware, and Information Integrity — AUTOMATED RESPONSE TO INTEGRITY VIOLATIONS		
ASSESSMENT OBJECTIVE: Determine if:		
SI-7(5)[1]	the organization defines security safeguards to be implemented when integrity violations are discovered;	
SI-7(5)[2]	the information system automatically performs one or more of the following actions when integrity violations are discovered	
	SI-7(5)[2][a]	shuts the information system down;
	SI-7(5)[2][b]	restarts the information system; and/or
	SI-7(5)[2][c]	implements organization-defined security safeguards.

Table 1 shows the organization of the certification guidelines for security control SI-7's Control Enhancement 5 (NIST, 2014). SI-7(5) enhances the SI-7 control statement to assert that steps to safeguard the system's security should be enacted when violations to information, software, or firmware integrity are discovered. The organization tailors SI-7(5) options to their own system safeguards, in addition to restart or shutdown.

The challenge to providing security awareness to a self-adaptive system is three-fold: (1) the security control must be expressed in a way that separates, yet captures the statement information, (2) how compliance is guaranteed in the system should be made explicit but at different abstraction levels, and (3) change must be introduced into the representation in such a way that its effect is understood and can be assessed. To address the security awareness challenge, we rely on assurance case concepts (Rushby, 2015). We previously defined a SAC template (Jahan 2018a) using Goal Structuring Notation (GSN) (GSN, 2018) that expresses the pattern of a NIST security control, allowing for compliance guarantees to be represented as claim or goal arguments. In addition, we defined adaptation operators that allow the template to evolve while still satisfying the claims (Jahan, 2018a). The concepts of achievement weights and satisficing levels were introduced as initial assessment metrics along with the concept of a security control network (Jahan, 2019a). We borrow design concepts from (Hawkins, 2015) and (Denney, 2012), where researchers have introduced metamodels of GSN to create templates. A SAC can instantiate the template by providing the values of the parameters indicated by curly braces. For the current effort, we extended the prior template representation to that which is shown in Figure 4.

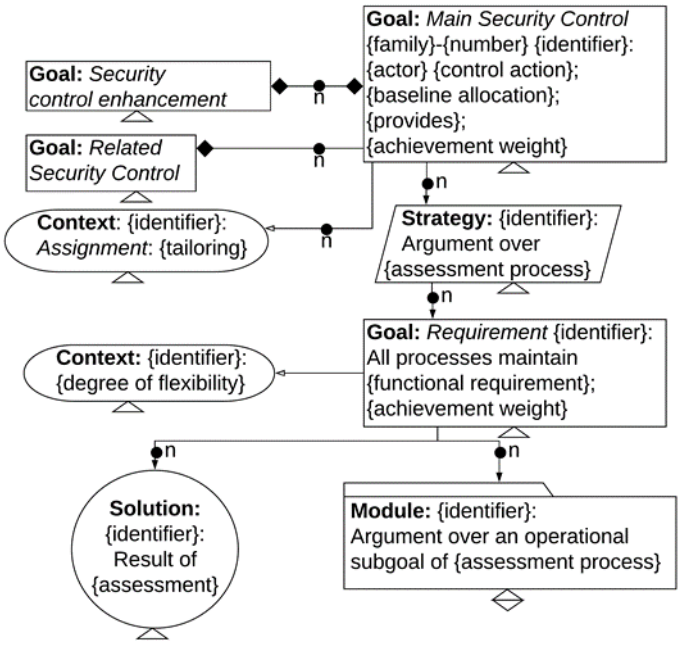

Figure 4: GSN Template for a Security Assurance Case

In Figure 4, Goals are represented as rectangles. The security control is the main assurance case Goal, which can have 0 or more enhancements that depend on it shown by the SupportedBy link (filled arrow). It is dependent on 0 or more related controls. Context nodes (ovals), attached to goals through the InContextOf link (hollow arrow), provide environment and state information. The main Goal can have 0 or more tailoring Context nodes to express selection and/or assignment. In GSN, the argument that supports the claim extends from an assessment Strategy (parallelogram) that may contain subgoals on which the argument depends. These subgoals may be defined in

lower level Modules (folder shape), such as process-specific operational goals related to an assessment method. For security controls, assessment methods can include examination, model checking, requirements verification, and testing. The assessments provide the evidence needed for the Solution (circle). We introduce an additional Context node as a numeric measure [0..1] of how flexible the satisfaction of the goal is to the full certification effort, which may be based on its singular importance or its interdependencies that could negatively impact certification through the propagation of compliance violation. A triangle associated with a GSN node means the node is abstract or uninstantiated. The joined triangles mean the node is both undeveloped and uninstantiated. The impact baseline allocation is also stated. The "provides" attribute holds the set of state variables and conditions that are part of the mechanisms needed for compliance with the security control. This set flows through a SupportedBy link that is augmented with a diamond to indicate the security control source for the provision set. In Figure 4, provision sets flow to the main control from related controls and enhancements. The achievement weight, a_w, is assigned to all goals, which holds the current value calculated at the goal for assessing the satisficing level of the main goal.

The design evidence for the system's trustworthiness is formulated through certification assessment methods of examination, interviewing, and testing (NIST, 2014). We instantiate the SAC template according to the labels, parameter values, and formal statement in (NIST, 2014) as shown for SI-7(5) in Table 1. Each instantiated SAC is used with the MAPE-SAC control loop to assess its level of trustworthiness with respect to an adaptation.

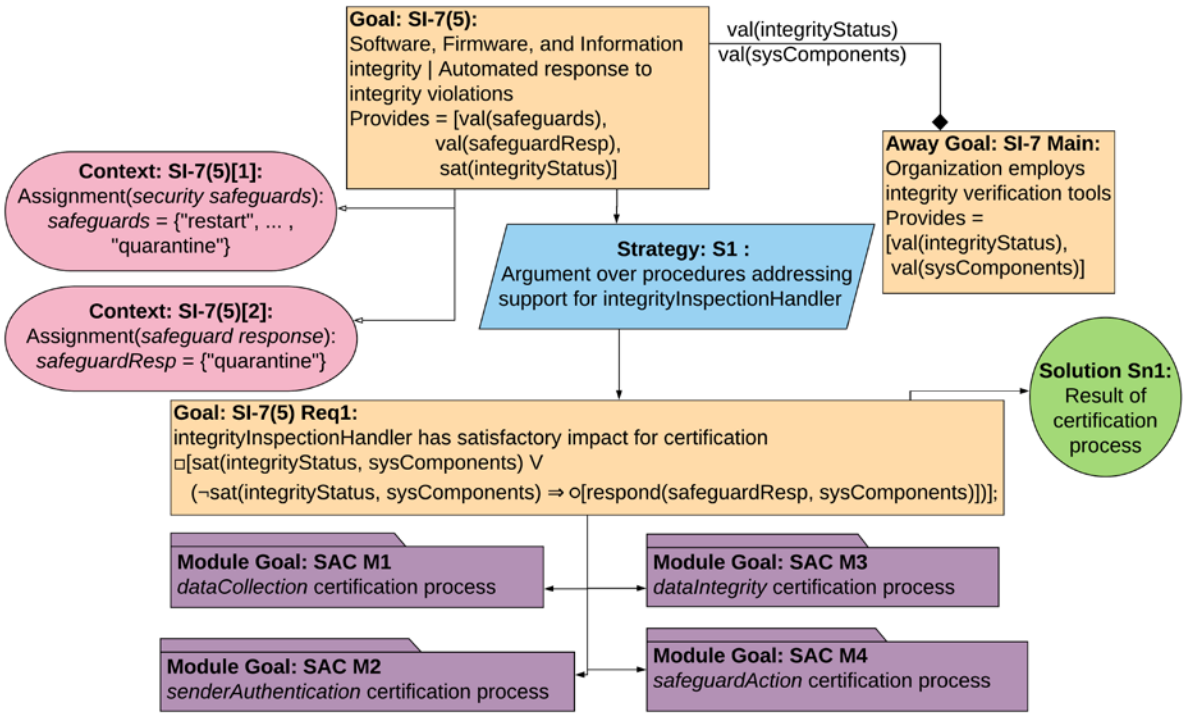

Figure 5: SI-7 (5) instantiated within the SAC GSN Template

Figure 5 shows the SAC template instantiation for SI-7(5) with the control statement as the main goal. Organizationally defined values form the context (pink ellipses) of the available safeguards (SI-7(5)[1]), with quarantine as the chosen safeguard response (SI-7(5)[2]). As a related control to SI-7(5), SI-7 is an Away Goal that provides specific parameters to SI-7(5) for its compliance strategy, as shown using the small diamond overlaid on the standard GSN supported-by arrow with two state variables ("val") in the provision set. Strategy S1 denotes the mechanism, *integrityInspectionHandler*, supporting the argumentation of SI-7(5)'s effectiveness. Req1 is a subgoal where Linear Temporal Logic (LTL) (Lichtenstein, 1985) is used to formally express the requirement as "It is always the case (□) that either the integrity of the system component is maintained or a safeguard response is activated in the next state (○)." Req1 is further decomposed into modules that are expanded into supporting operational goals directly associated with system code.

Figure 6 expands module goal *SAC M1* into OpGoal *M1 G-1* that introduces an operation to store data. Module *SAC M3* (Figure 7) involves the operation to aggregate data and compare the standard deviation (σ) to detect anomalous behavior in the event of sensor failure or compromise to activate *anomalyAlert*. Module *SAC M4* (Figure 8) defines the role of *safeguardResponse* should anomaly detection occur.

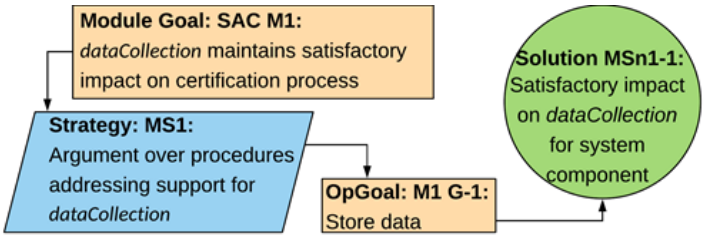

Figure 6: Module Goal: SAC M1 focusing on Data Collection

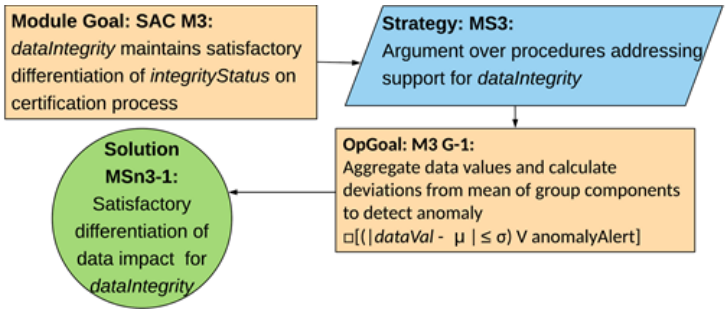

Figure 7: Module Goal: SAC M3 focusing on Data Integrity

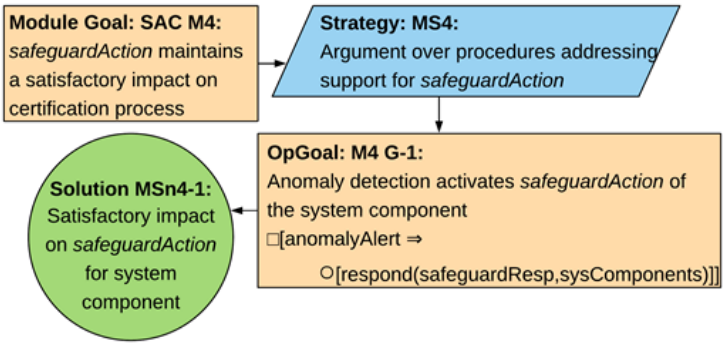

Figure 8: Module Goal: SAC M4 focusing on Safeguard Actions

3.4 Exploring and Managing Uncertainty

As part of the integration objectives to support the MAPE-based adaptation, another segment of the research was to use evolutionary computing-based techniques to identify individual execution modes that are robust, where the modes can be composed with machine learning-based adaptive logic to provide an overall resilient system. The intent was to make the modes and overall system resilient to both onboard and environmental uncertainty, including cybersecurity threats. Michigan State University's efforts with Enki have been applied to both machine learning algorithms (Langford, 2019b), as well as onboard control behavior (Langford, 2019a). The work has been applied to benchmark data and to the AutoRally autonomous vehicle, respectively. With the mode identification and the mode switching capability, we attempted to support both MAPE-K and MAPE-SAC adaptations by considering adaptations as a variation of mode switching. As such, we worked with computationally intensive analysis capabilities to identify robust modes (making use of Enki), while using machine learning techniques to identify a means to transition between modes, thus making the overall system more resilient. The MAPE infrastructure provided a systematic process for run-time monitoring of the operating context, as it affected both the functional and security-based concerns.

3.4.1 Robust Learning-enabled Systems

By experimenting with how autonomous systems can make use of machine learning, it became paramount to ensure that learning-enabled components and systems, LEC and LES, respectively (i.e., those that contain machine learning elements) are robust to a broad range of operating contexts (Langford, 2019b). A major challenge faced by the ML community is the limitations posed by the scope and coverage of the training data and test data used to develop LECs. In order to address this issue, we investigated how Enki could be used to generate synthetic training data to

i. assess the robustness of learning-enabled systems (LES) to conditions for which they had not previously been trained; and
ii. address the question whether we could use Enki-generated training data to improve the robustness of a LES

In both cases, we showed that Enki was better able to assess and improve the robustness of the LES when compared to existing approaches. As part of an early study, Enki was tasked with generating environmental conditions relevant to an onboard image classification system, comparable to what is being used for autonomous rovers and vehicles (Langford, 2019a). Enki was able to uncover a diverse range of conditions to transform existing image data sets to account for lighting, contrast, and raindrops on the camera lens. We use Enki with the EvoROS platform, with the eventual goal of being able to make ROS-based systems more resilient to both internal and environmental uncertainty.

3.4.2 Applying Enki to Onboard Controllers

We expanded the Enki and EvoROS work to explore how evolutionary-based techniques can be harnessed to uncover environmental and onboard conditions to identify a range of onboard controller behavior (Langford, 2019a). The first step was to use predefined settings for a Proportional Integral Derivative (PID) controller used to regulate the speed of an autonomous rover — the target platform with the specifications corresponding to the AutoRally rover. EvoROS is able to evolve PID controller settings that yield a better performing controller than the default PID settings. Then we used Enki to generate environmental conditions that reveal the most diverse controller behavior, ranging from near failure to unexpected behavior. Based on this information, we were able to use the Enki-discovered environmental conditions to evolve better overall resilient behavior.

3.4.3 EvoROS and AutoRally Demonstration Platform

As part of the integration efforts between the X-PLORE project (Cheng, 2020) and this project we continued work on the Evo-ROS evolutionary robotics platform[8], which enables evolutionary search to be applied to robotic systems based on the Robot Operating System (ROS). The current version developed in X-PLORE is Evo-ROS 2.0. The major feature of Evo-ROS 2.0 compared to earlier versions is to remove a dependence on Ardupilot[9] in the simulation software stack. While Ardupilot is needed for simulating certain platforms, such as the Erle-Rover, it also limits the number of platforms to which Evo-ROS can be applied. In addition, Ardupilot added an unnecessary level of complexity to the software and contained hard-coded control loops that prevented faster-than-real-time simulation. Finally, Ardupilot introduced significant overhead in the time required to prepare a simulation instance. In contrast, Evo-ROS 2.0 focuses on serving purely ROS[10]-based systems, with the inclusion of Ardupilot as an option. This design allows the software management code of Evo-ROS 2.0 to be more efficient and user-friendly.

[8] http://www.cse.msu.edu/~mckinley/autorally/frames/evo-ros.html
[9] https://ardupilot.org/
[10] https://www.ros.org/

3.4.3.1 AutoRally Simulation

Previously we completed the integration of the AutoRally autonomous vehicle into the Evo-ROS framework. AutoRally is a 1:5-scale truck developed by researchers at Georgia Tech and which the Michigan State University (MSU) team has constructed. The AutoRally software is entirely ROS-based. With the integration completed, we investigated ways to apply Evo-ROS in order to enhance the performance and safety of the AutoRally platform. We evolved parameters for the PID controller that governs the throttle, then utilized Enki in order to find "challenging" conditions. Repeating this process produced a controller that outperforms the original PID controller in terms of tracking a reference signal for desired speed of the vehicle. In addition, we developed ROS nodes to realize adaptive functionality in the case of sensor failure or anomalous readings. These nodes will be used in future work to enable dynamic adaptation of the platform as well as interaction of the MAPE-K and MAPE-SAC loops on AutoRally.

3.4.3.2 AutoRally Physical Platform

The AutoRally physical vehicle was completed at the end of the X-PLORE project and overlapped with the research effort on the MAPE loop interaction experimentation. The completed platform weighs 46 lbs with a top speed of 60 mph. Default sensors include a high-precision Inertial Measurement Unit (IMU), Global Navigation Satellite System (GNSS), Hall-effect rotation sensor on each wheel, and two front-facing machine vision cameras. The computing resources are housed in a custom compute box and include an Intel Skylake Quad-core i7, 32GB DDR4 RAM, 2TB SSD storage, an Nvidia GTX 1050 Ti GPU for real-time image processing, and WiFi and XBee network interfaces. The Michigan State University version of AutoRally is fully compatible with the ROS navigation stack, which takes in sensor and odometry data and outputs throttle and steering commands to produce navigation from the platform's current location to a goal pose (consisting of location and heading). The ROS navigation stack maintains both a global and local cost map of the environment, updating the presence of obstacles in real-time, allowing it to plan paths to goal locations while avoiding both stationary and moving obstacles. We conducted several outdoor tests of the vehicle for the purpose of calibrating the Evo-ROS simulation model. In addition, we replaced the Hemisphere GNSS receiver with a Piksi Multi GNSS receiver, requiring redesign and fabrication of the receiver housing. The new receiver can accommodate RTK correction streams, in our case from a nearby Michigan Continuously Operating Reference Station (CORS). With RTK enabled, we confirmed a location accuracy within approximately 2 centimeters. Our investigations focus on the obstacle avoidance software.

3.4.4 AutoRally Functional and Security Properties

As part of the MAPE loop interaction work, we worked on modeling functional and security constraints applicable to the AutoRally platform. Functionally, we focused on obstacle avoidance and navigation as part of autonomous driving, where we have access to several sensors, including: IMU, camera, and GPS. Key subsystem functionalities include collision avoidance, speed monitoring, remaining battery monitoring, communication among subsystems, and position monitoring. From the security perspective, we returned to the earlier SAC representation of SI-7(5) (Table 1 and Figure 5) to reexamine the initial scenario where the vehicle must provide system integrity by identifying a set of security safeguards with corresponding responses. In the context of uncertainty, several functional and security concerns were investigated in the "large". External uncertainty included environmental based, such as weather conditions as they impact the onboard sensors, external obstacles, and security attacks. Onboard (internal) uncertainty refers to incomplete requirements and unknown (unwanted) feature interactions. Security vulnerabilities include denial of service (via flooding of one or more onboard sensors), modification and/or fabrication of sensor signals.

3.5 Interactions between MAPE Loops

As adaptations introduce new state changes, the system must ensure compliance with both functional and security requirements. One issue is the adverse effect of a functional adaptation on security certification or a security adaptation that inhibits expected functional behavior. Our approach separates consideration of functional and security concerns within distinct MAPE loops, where the traditional MAPE-K loop handles functional concerns and the new MAPE-SAC loop is for security concerns (Jahan, 2019b). We define the concerns (and their associated constraints) as loosely coupled "regions," or models, of the system with prescribed, local adaptations for each region. With this representation, an adaptation in one region may affect the other region, requiring a coordinated approach to planning and executing a unified, system-wide adaptation.

Patterns for MAPE loop coordination have been proposed, focusing on how the internal components (monitor, analyze, plan, execute) are distributed and interact with their peers in other MAPE loops (Vromant, 2011). These patterns assume that each MAPE loop corresponds to a system entity and that interloop coordination is needed for system-wide adaptation (Weyns, 2013). We used a hybrid of the Regional Planning Pattern and the Coordinated Control Pattern (Weyns, 2013) instantiated in Figure 9 for the interaction between the MAPE-K and MAPE-SAC loops residing in their respective functional and security regions. Each region has dedicated monitor and analysis components that work independently and in parallel. Adaptation needs can be triggered by both or either region to engage the coordinated planner. The chosen adaptive plan causes a regional update to the assurance cases, along with coordinated execution to change the system-level code.

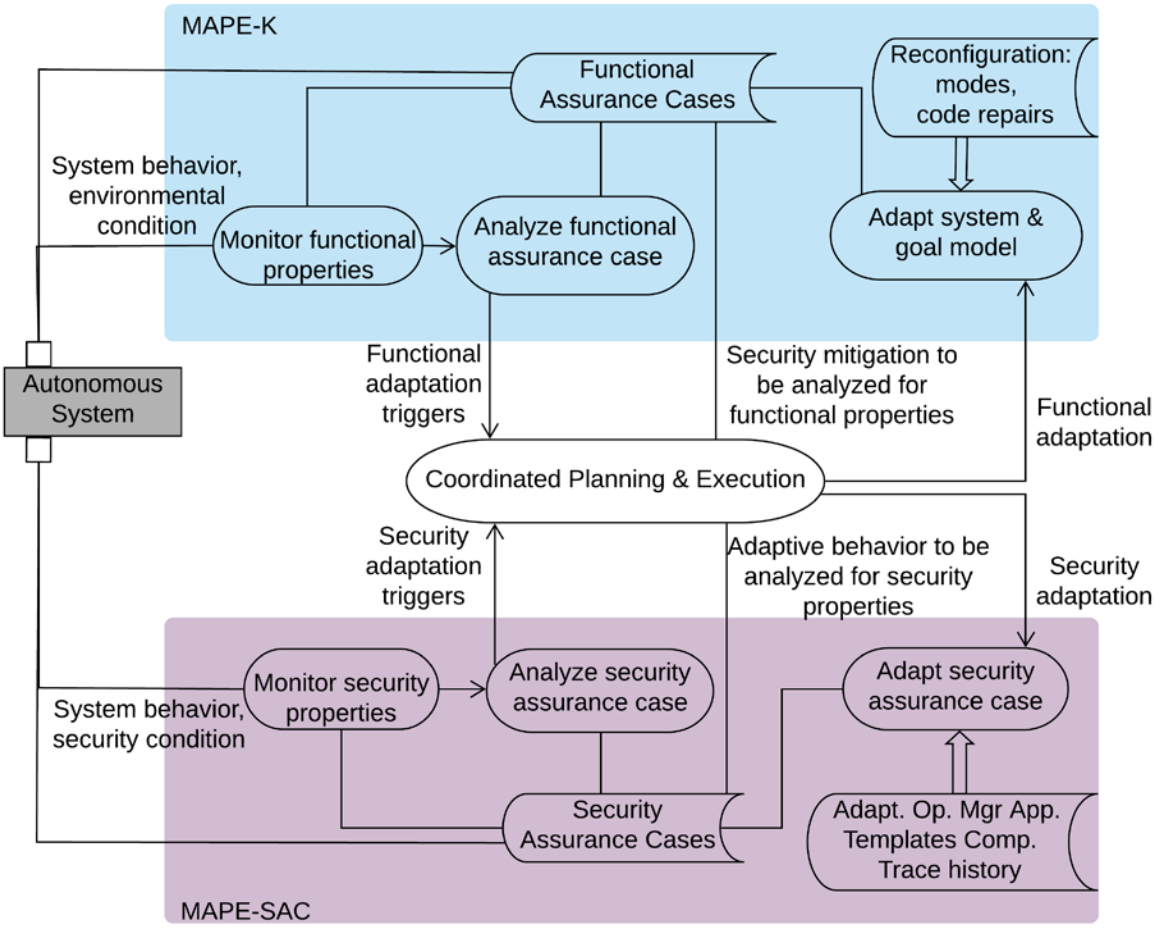

Figure 9: Interaction between MAPE-K and MAPE-SAC loops

3.5.1 Using Functional Assurance Cases with the MAPE-K loop

To demonstrate the potential for interaction between the MAPE-K and MAPE-SAC loops, consider an autonomous rover FAC use case (Jahan, 2019b), defined using GSN as shown in Figure 10. The main goal is for the rover to drive autonomously. The rover is constrained by context state variable assignments. Context *C1* sets the range of possible throttle values (*throttleRange*) to be [90, 100]%, which is the expected default range. *C2* sets the mission time threshold (*missionTimeCap*) to be 60000 seconds. *main* is supported by strategies *S1* and *S2*. *S1* expects the rover to use adaptive cruise control (ACC) for navigation. *S1* is supported by a GSN Option relationship (large black diamond), indicating subgoals *Req1* or *Req2* can be used. Contexts *C3* and *C4* provide the usage precedence to effectively create an XOR relationship between the subgoals. To support ACC, subgoal *Req1* relies on using wheel sensors for speed management and is satisfied by the rover's speed log evidence of properly functioning wheel sensors in solution *Sn1*. *Req2* can support *S1* by relying on GPS. With lower precedence, *Req2* and its supporting branch have a dashed line to indicate its inapplicability. Strategy *S2* asserts the time to complete

the mission is minimized. Its subgoal *Req3* is supported by solution *Sn3* based on throttle controller log that the current throttle is within the *throttleRange*. Subgoal *Req4* minimizes the mission time by ensuring the elapsed time (*elapsedTime*) is less than *missionTimeCap*.

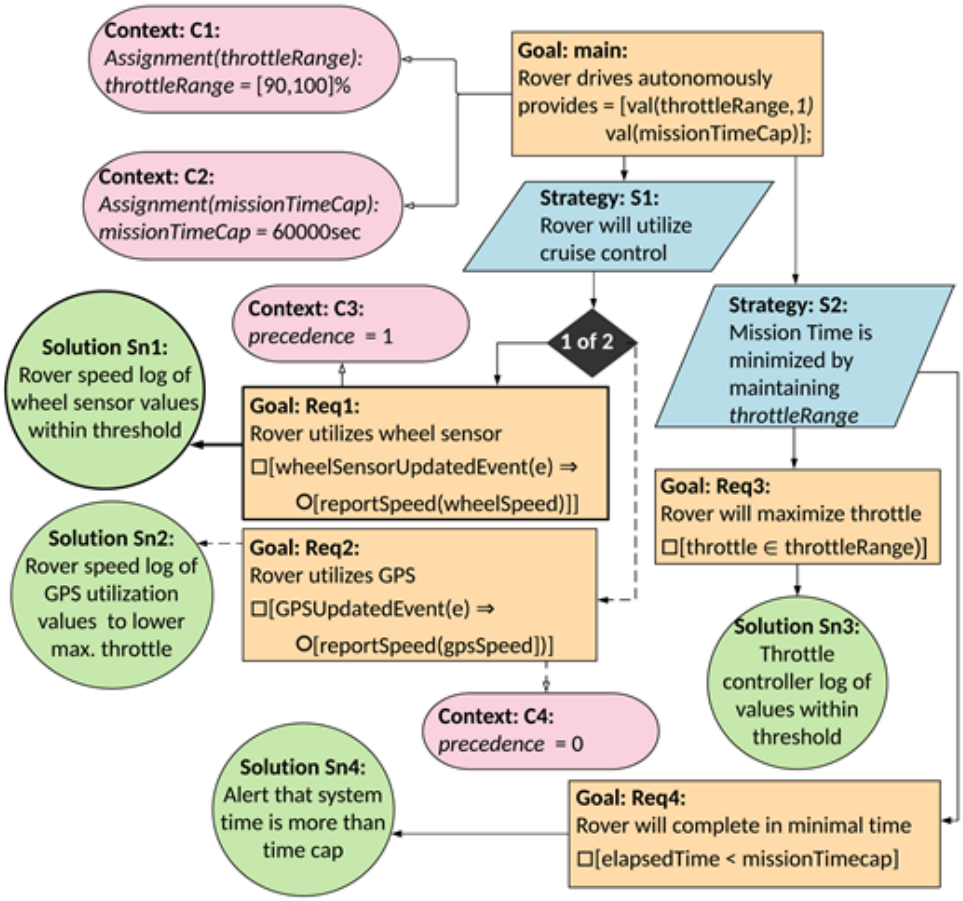

Figure 10: Functional assurance case for Rover – pre-adaptation

3.5.2 Compromised System Scenario

Assume the autonomous rover, using Adaptive Cruise Control (ACC) for navigation, is traveling at high speeds off-road. In the functional concern region (i.e., MAPE-K), a high-level claim is made that the rover should minimize travel time (strategy *S2*), which is supported by a strategy to maintain higher speeds. Due to a flaw in manufacturing, assume hackers have found a way to remotely tamper with the rover's wheel encoder used for measuring speed. Further assume that the hackers have compromised the wheel sensors and begin to send spoofed sensor values that report a significantly lower speed.

3.5.3 Trigger Adaptation

Regional monitors observe speeds reported by the wheel sensors and the vehicle's GPS unit. The MAPE-K monitor determines if the speed is within the range of possible throttle values and consistent with speeds of GPS unit. The MAPE-SAC monitors speed consistency to determine if there is an anomalous change. Although both monitors track the speed value reported by sensors, they associate the observation directly with their concerns, providing either a functional or security perspective on the behavior.

Analysis is also performed regionally with respect to functional and security concerns. The MAPE-K analysis component examines the log file of wheel sensor values given *Req1* shown in Figure 10 and discovers anomalous values. In parallel, analysis performed by the MAPE-SAC against the module goal SAC *M3* shown in Figure 7, discovers an unacceptable standard deviation (σ) collected through the run-time evidence when compared with the design evidence originally establishing *MSn3-1*, and it determines that data integrity has been compromised. Both MAPE-K and MAPE-SAC loops trigger the need for adaptation. Though we have not addressed how coordinated planning ensures it, we currently assume that trigger ordering or even delay of one region to trigger does not affect the planner's adaptation choice or outcome. Future work will examine this issue more closely.

3.5.4 Adaptation

The planner generates a system-wide plan to achieve optimal utility for functional and security objectives, such as:

- **Functional Adaptation**: Quarantine wheel sensor and use only GPS for speed control.
- **Security Adaptation**: Quarantine wheel sensor by disabling storage of wheel sensor's data.

The SI-7(5) SAC in Figure 5 asserts that the system should apply its defined safeguard action with the realization that integrity is compromised. The rover's *safeguardResponse* is to quarantine system components, blocking all traffic from affected *sysComponents*. This response needs an adaptation to disable the data collection process. The adaptation must replace the functionality of the operational goal *M1 G-1* to satisfy the module goal SAC *M1*, which is performed in Figure 11. This adaptation is performed by a Substitute operation (Jahan, 2018a) applied to replace the data collection functionality at operational goal *M1 G-1* and satisfy the module goal *SAC M1*, shown in Figure 11.

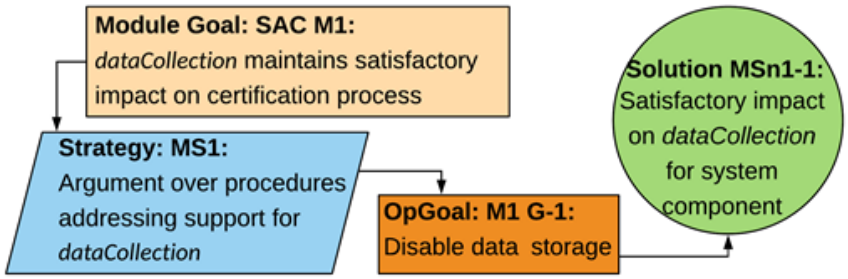

Figure 11: Module Goal: SAC M1 *dataCollection* after adaptation

Without operational wheel sensors, vehicle functionality must adapt to complete its mission objectives. As the wheel sensors are now quarantined, the coordinated planner identifies a discrepancy at the FAC subgoal *Req1* in Figure 10. Subsequently, it must craft a solution other than *S1* that supports *Req1* to satisfy *main*. Thus, an adaptation is designed to change the precedence for subgoals *Req1* and *Req2* in Figure 10 to force reliance on *Sn2* that uses GPS for speed control. However, GPS polling is slower and less precise than the wheel encoder, which decreases the maximum allowable rover speed for vehicle safety assurance. The context *C1* for assigning state variable *throttleRange* must be updated from [90, 100]% to [20, 60]%, which is based on a pre-defined set of allowable values for *throttleRange* when using only GPS for speed control. Updating the *throttleRange* satisfies solution *Sn3* that supports the evidence for subgoal *Req3* (i.e., throttle is within *throttleRange*). The adapted state changes are performed using three *ChangeVal* operations (Jahan, 2018a) on contexts *C1*, *C3*, and *C4* in the FAC shown in Figure 12.

3.6 Initial Coordination of MAPE Loops

Once the planner determines potential adaptation plans based on the triggered adaptation type, coordination across regions is performed to ensure that a functional adaptation does not violate security constraints and vice versa. To coordinate, the effect of the adapted functional behavior on the SACs is assessed to determine suitability. If the result is that the SAC must be adapted, any propagation of that adaptation must be measured within the SAC network and then reasoned about given the non-adapted FACs. The coordination goal is to achieve an optimal system-wide, global adaptation. If an adaptation in one region indicates a potential violation of a requirement in the other region, then adaptation re-planning will occur, thereby removing the previous adaptation plan from consideration. If no adaptation can be found, then the system either enters a default safe state or remains in its current state. For the described adaptation scenario, a security adaptation that disables the data storage of *sysComponents* is analyzed against the FAC (Figure 10). Disabling the data storage results means no longer needing to assess the log of the affected wheel sensor values because the FAC is adapted to satisfy *Req2* and not *Req1*. Thus, disabling the data storage of affected *sysComponents* does not violate functional behavior. However, the functional adaptation, which is to quarantine the affected wheel sensors, satisfies the security requirement of having a safeguard action.

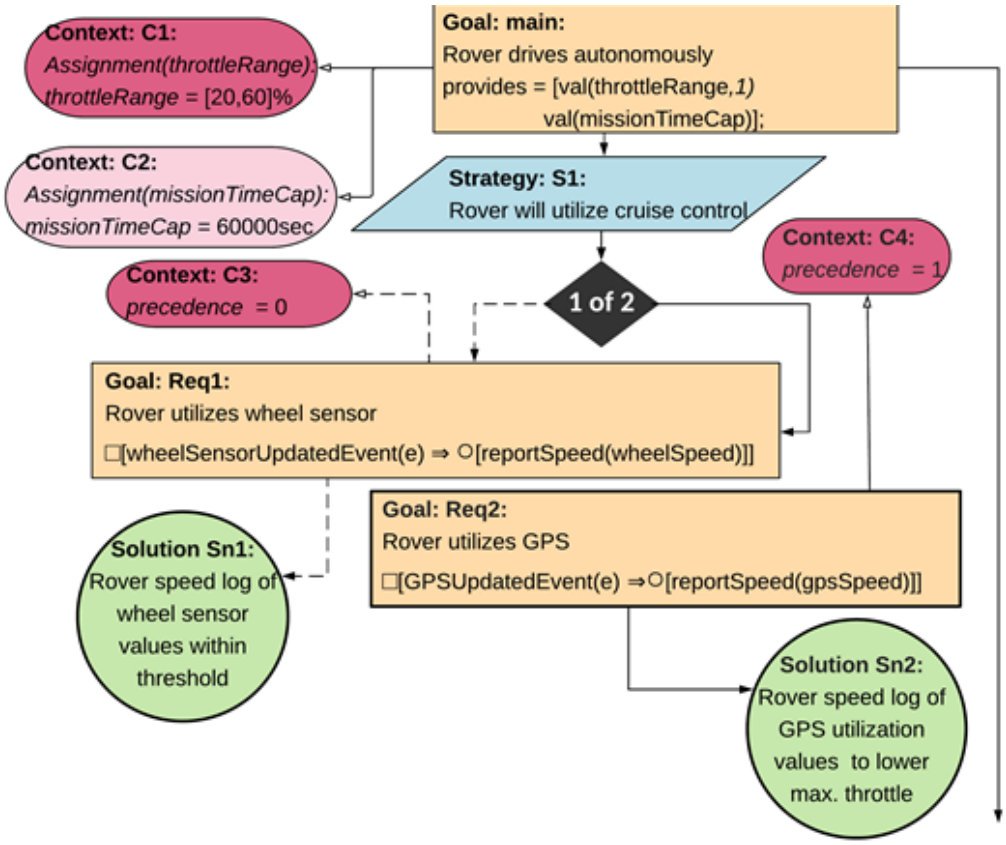

Figure 12: Functional Assurance Case of Rover – Post-adaptation

3.6.1 Executing the Adaptation

Using the hybrid MAPE loop interaction pattern based on Regional Planning and Coordinated Control Patterns (Weyns, 2013), executing the adaptation plan is also coordinated. However, the results of the changes to the code must inform any changes to the FACs and SACs, as discussed with the changes shown in Figure 11 and Figure 12. Thus, the coordination of the execute phases of the interacting MAPE loops involves system reconfigurations, while the regional execution component performs model changes.

4 Results and Discussion

4.1 Evaluating the MAPE-K/MAPE-SAC Interaction Framework on the Wearable and Cozmo Testbeds

The interaction between the MAPE-K and MAPE-SAC loops is represented in Figure 9. The Monitor and Analyze components in both MAPE loops monitor and analyze their respective concerns in parallel. Adaptation can be triggered by either or both of the MAPE loops. The loops coordinate to plan the optimal adaptation of the system that is compliant with both functional and security requirements (Jahan, 2020a). When an adaptation has been triggered for either type of

requirement, the coordinated planner identifies potential changes to functional and security behavior due to the adaptation, and each MAPE loop planner assesses the effect of the adaptation on its own region of concerns. MAPE loop planners coordinate with each other to reach the optimal adaptation with respect to both types of requirements. The optimal adaptation is the adaptation that best satisfies the compliance level of the system's security requirements without risking the evidence (e.g. proof, testing results) that supports the system's compliance with its functional requirements. To select an optimal adaptation, both planners interact with each other, which for this use case was performed manually. The chosen adaptive plan is executed in a coordinated manner to evolve regional assurance cases along with changes on the system-level code. We introduce an implementation of a novel interaction protocol for MAPE loops within the framework to create the MAPE-K/MAPE-SAC Interaction Framework discussed next.

To demonstrate the effectiveness of MAPE-K/MAPE-SAC Interaction Framework, we return to the testbeds described in Section 2. Each testbed becomes an adaptive system that operates its own MAPE-K/MAPE-SAC framework tailored to its requirements. For the wearable testbed, we allow the four possible adaptations defined in Section 2. For the Cozmo testbed, the robots can adapt to adjust their current goals to complete a global objective more quickly, such as searching a larger or different area. We focus on the natural disaster scenario from the first use case of the interacting testbed outlined in Section 2.

The assumption is that the robots are equipped with visual sensors, searching for survivors, as well as Bluetooth to connect to a survivor's phone or wearable device. Because Bluetooth can be used for localization (Daniş, 2017), the robots can discover the location of survivors accurately even when visual identification is impossible. Since it is important to prioritize survivors who may need immediate medical attention, the robots should connect to and request basic information from health-monitoring wearables, such as heart rate or stress monitors, as simulated by the wearable testbed. With the allowable adaptations available to the wearables, they must still prioritize the security of personal data, which conflicts with the needs of the robots. We focus on the security constraints documented in Table 2. The wearable testbed must share information with the robots (role-based adaptation).

To resolve this conflict, rather than remove the ability of the wearables to prioritize security completely, we rely on *fostering* as a communication protocol originally designed to pass security states between wearables quickly (Walter, 2018b). Fostering can be repurposed for passing important information to the rescue robots when needed. When adapted to perform fostering, a robot can search for Bluetooth signals and, once found, connect to the available fostering server on the wearable testbed. It can remain connected only for a short time that is long enough to request and receive information. The fostering protocol allows only a single connection request, a single request for data, and responses to these two requests before it forcibly terminates the connection.

Table 2. Interrelated Security Controls defined by NIST SP800-53A REV. 4

AC-3: ACCESS ENFORCEMENT		
ASSESSMENT OBJECTIVE: Determine if the information system enforces approved authorizations for logical access to information and system resources in accordance with applicable access control policies		

AC-18: WIRELESS ACCESS		
ASSESSMENT OBJECTIVE: Determine if the organization:		
AC-18(a)	establishes for wireless access:	
	AC-18(a)[1]	usage restrictions;
	AC-18(a)[2]	configuration/connection requirement;
	AC-18(a)[3]	implementation guidance; and
AC-18(b)	authorizes wireless access to the information system prior to allowing such connections	

AC-21: INFORMATION SHARING		
ASSESSMENT OBJECTIVE: Determine if the organization:		
AC-21(a)	AC-21(a)[1]	defines information sharing circumstances where user discretion is required;
	AC-21(a)[2]	facilitates information sharing by enabling authorized users to determine whether access authorizations assigned to the sharing partner match the access restrictions on the information for organization-defined information sharing circumstances;
AC-21(b)	AC-21(b)[1]	defines automated mechanisms or manual processes to be employed to assist users in making information sharing/collaboration decisions; and
	AC-21(b)[2]	employs organization-defined automated mechanisms or manual processes to assist users in making information sharing/collaboration decisions.

IA-3: DEVICE IDENTIFICATION AND AUTHENTICATION		
ASSESSMENT OBJECTIVE: Determine if:		
IA-3[1]	the organization defines specific and/or types of devices that the information system uniquely identifies and authenticates before establishing one or more of the following:	
	IA-3[1][a]	a local connection;
	IA-3[1][b]	a remote connection; and/or
	IA-3[1][c]	a network connection;
IA-3[2]	the information system uniquely identifies and authenticates organization-defined devices before establishing one or more of the following:	
	IA-3[2][a]	a local connection;
	IA-3[2][b]	a remote connection; and/or
	IA-3[2][c]	a network connection;

SC-40: WIRELESS LINK PROTECTION		
ASSESSMENT OBJECTIVE: Determine if:		
SC-40[1]	the organization defines:	
	SC-40[1][a]	internal wireless links to be protected from particular types of signal parameter attacks;
	SC-40[1][b]	external wireless links to be protected from particular types of signal parameter attacks;
SC-40[2]	the organization defines types of signal parameter attacks or references to sources for such attacks that are based upon exploiting the signal parameters of organization-defined internal and external wireless links; and	
SC-40[3]	the information system protects internal and external organization-defined wireless links from organization-defined types of signal parameter attacks or references to sources for such attacks.	

Though the combined testbeds can communicate via the cloud, for our purposes, we focused on their Bluetooth communication capability. Should a base station be discovered via Bluetooth, the base station of the robot will connect to the base station of the wearable and maintain that connection until the wearable location is discovered, at which point the location will be stored and the robot will continue searching its designated area for persons in need of assistance.

4.1.1 Deploying New Security Assurance Cases

The SACs needed are implemented on the base stations in each testbed. The security requirement defined by security control AC-3 (Table 2) ensures authorization and can be maintained in two ways. The first approach uses default Bluetooth pairing, which has the Raspberry Pi 3s refuse connections from unpaired devices. The second approach is within the control code, where a list of approved devices is stored and checked before connections are accepted. The latter means that, even if a device is paired, it cannot connect in order to request data without also being on the approved list. Both the request information (containing Media Access Control (MAC) addresses of the connecting devices) and the result of the request are logged into a connection log for future review. This results in a conflict preventing the robots from connecting to the wearables and requires adaptation of the SAC to allow fostering.

AC-3, which relies on security control AC-18 and AC-21 for determining access requirements, restrictions and circumstances, asserts that there must be steps to enforce authorization of logical access to information and system resources. SC-40 asserts that the information system should protect wireless links from defined signal parameter attacks. SC-40 is indirectly related to IA-3, which asserts that the information system should identify and authenticate devices before establishing a connection. Both SC-40 and IA-3 rely on AC-18 for obtaining the guidelines to implement protection and authorization mechanisms. The design evidence for the system's trustworthiness is formulated through certification assessment methods of examination, interviewing, and testing (NIST, 2014). We instantiate a security control network (SCN) according to the labels, parameter values, and formal statements as shown in Table 2.

Figure 13 shows the interrelated security controls of AC-3, AC-18, AC-21, IA-3, and SC-40 at a high level reusing the developed network constructs (Jahan, 2019a). Control statements are stated as main goal statements and the "provides" attribute holds the provisional set of state variables' values (val) and conditions (sat) as a part of the security control's dependencies for compliance. This set flows within the network through a SupportedBy link, augmented with a diamond to designate the security control source for the provisional set, which implies the interrelationship among the security controls.

Security control SC-40 (Table 2 and Figure 13), which focuses on organization-defined attacks (in this case: Man-in-the-Middle attacks), is achieved by adaptations that either result in sending empty packets or performing disconnection. This again results in a conflict requiring adaptation when fostering. We also check the current Bluetooth connection status and, if no data is being sent, disable Bluetooth entirely.

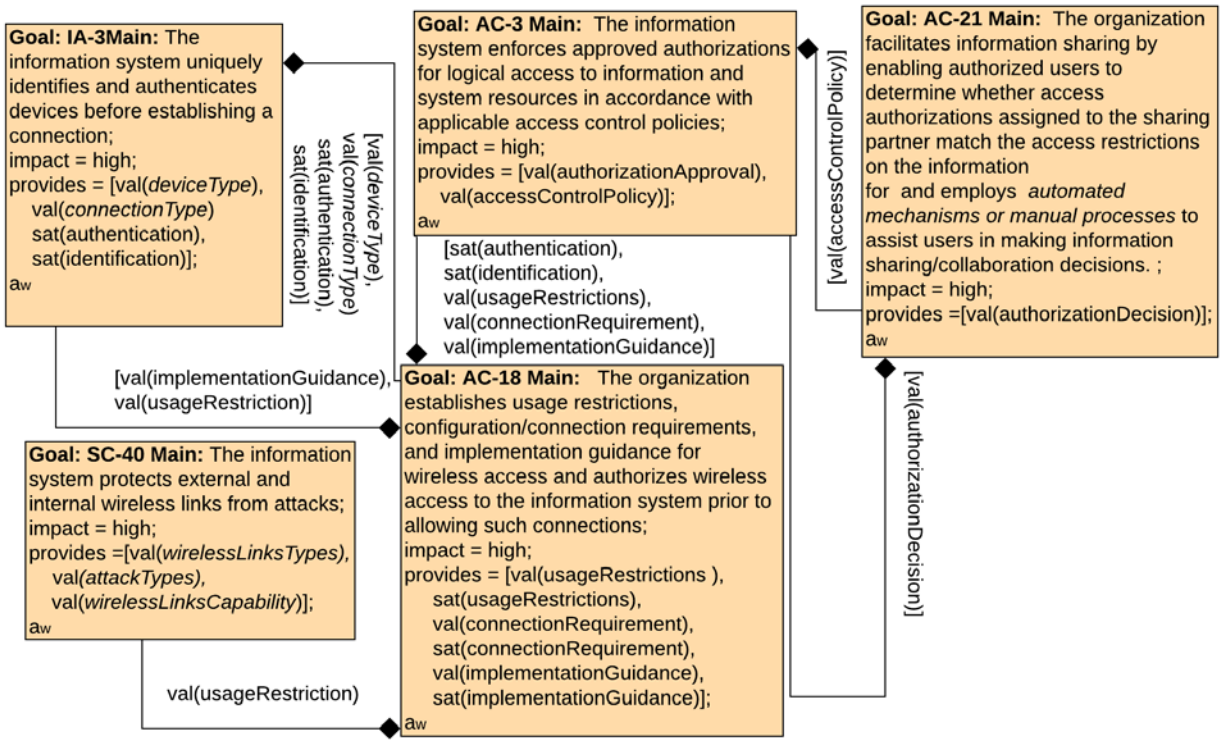

Figure 13: Instance of security control network for access control (partial)

Security control IA-3 (Table 2 and Figure 13), aimed at device identification, is achieved through the use of randomized PINs when pairing Bluetooth devices. We directly embedded the Personal Identification Number (PIN) randomization into the control code, choosing a random PIN for each connection request which must be confirmed by the user initiating the connection. The PIN is valid only for a limited time, in this case 10 seconds, before a new PIN is generated. The random PIN is stored and checked to ensure it is of sufficient length and strength. We define sufficient length as having at least 8 characters, and a strong PIN as any PIN not having the same number repeated consecutively. IA-3 also conflicts when a robot must connect to the device, requiring an additional adaptation for fostering.

We instantiate the SAC template in Figure 4 for AC-3, AC-18, AC-21, IA-3, and SC-40—covering access control (AC), information and authorization (IA), and system and communications protection (SC). Figure 14 shows the SAC for AC-3 with the control statement as the main goal. Organizationally defined values form the context (pink ellipses) of the applicable access control policies (AC-3), with an identity-based policy as the chosen access control policy. As a related control to AC-3, AC-21 is an Away Goal, a separate but related argument that provides or requires support from the main goal, which provides specific parameters to AC-3 for its compliance strategy. AC-18 is another Away Goal which requires specific parameters from AC-3 (as does AC-21) to certify its own compliance strategy. Strategy *AC-3 S1* denotes the mechanism, *enforceAuthorization*, supporting the argumentation of AC-3's effectiveness. *AC-3 Req1* is a sub-goal where LTL or other propositional logic can be used to formally express requirements unambiguously.

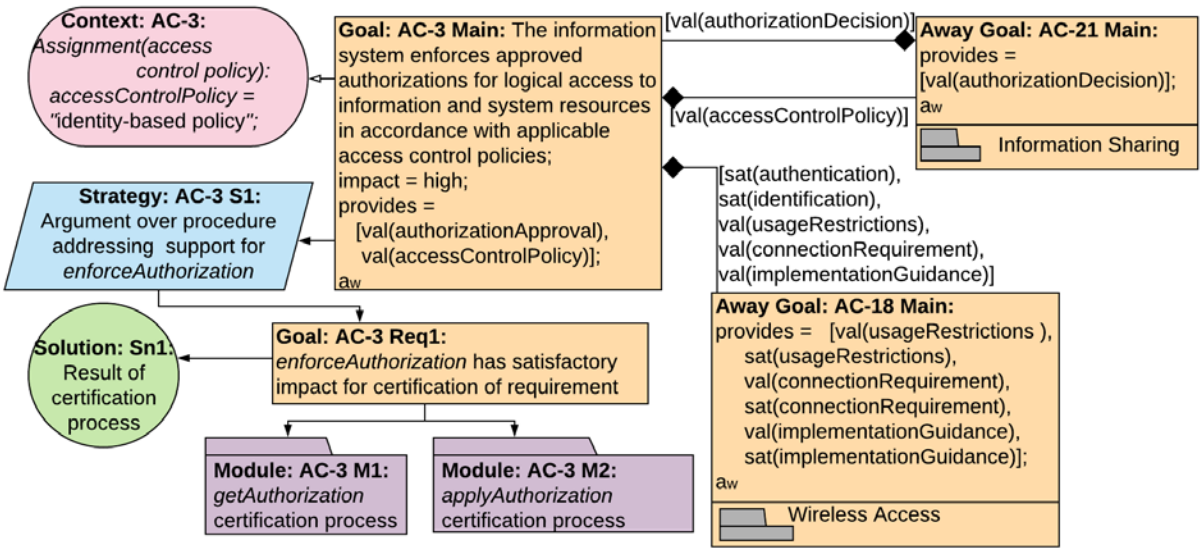

Figure 14: AC-3 instantiated within the SAC GSN template

AC-3 Req1 is further decomposed into modules that are expanded into supporting operational goals directly associated with system code (Figure 15). Solutions (green circles) support the certification of AC-3. SACs for security controls AC-18, AC-21, IA-3, and SC-40 are deployed as part of the knowledge of the MAPE-SAC control loop.

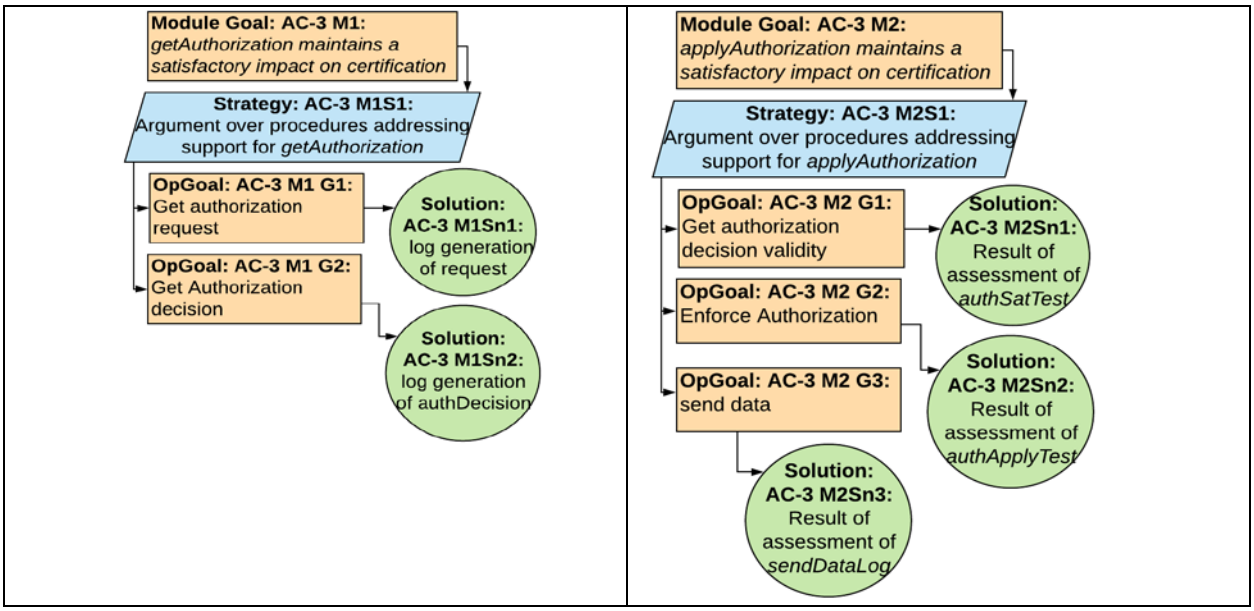

Figure 15: Expanded AC-3 Module Goals to Operational Goals

4.1.2 Revisiting Functional Assurance Cases

Both the wearables and the robots have unique requirements, which we define as FACs that cannot be violated under normal operation. These FACs are represented using GSN notation exactly like the SACs. The FACs are unique to each wearable device but are shared across robots. The heart rate variability monitor (HRVM) FAC is shown in Figure 16. The wearable main goal, to autonomously monitor both heart rate and stress level, is composed of two sub-goals. The first sub-goal ensures that the buffer that stores the heart rate data does not overflow, as a loss of heart rate data will violate the main goal. The second ensures that the buffer that stores the stress level data does not overflow. We note that the LTL square means "it is always the case" and the LTL diamond means "eventually" (Lichtenstein, 1985).

During the monitor phase of the MAPE-K loop, heart rate data, stress level data, and buffer conditions are monitored on the HRVM testbed and are analyzed to determine the user's health condition. During the analyze phase, the FAC is instantiated with observed stress level data and buffer conditions so that the system can assess its compliance status. The *determineStressLevel* and *bufferMaintenance* procedures analyze stress level data and buffer conditions over time to determine whether the user is stressed or if the buffer has overflowed, which eventually implies that the user needs immediate medical attention. An adaptation is triggered in response to significant requirement compliance degradation.

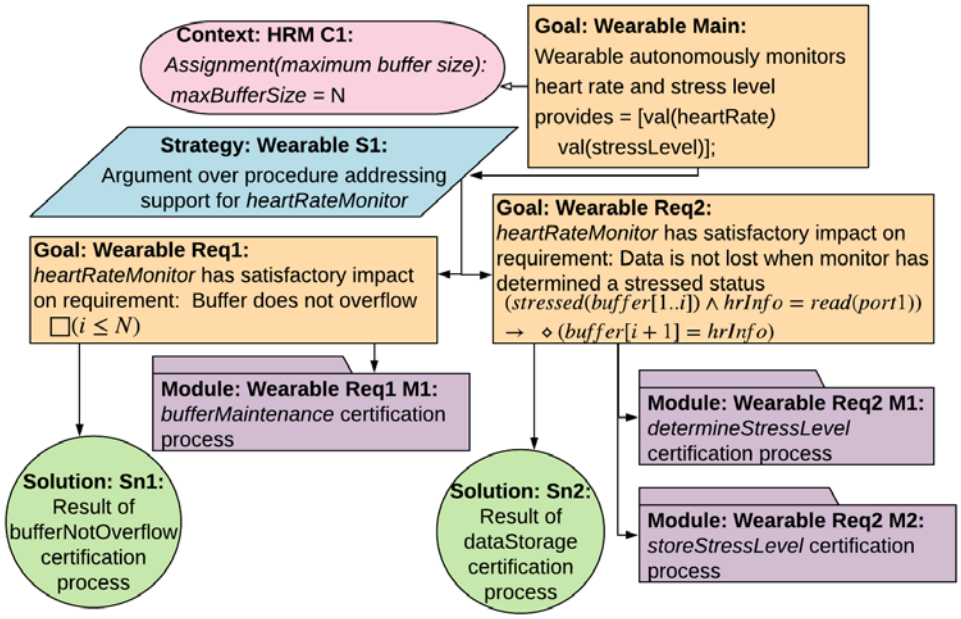

Figure 16: Heart Rate Variability Monitor Functional Assurance Case

The robot FAC is shown in Figure 17. Its primary goal is search and rescue. It contains an away goal to announce the location of people it finds and their heart rate condition, if data is available, and an away goal to integrate itself into the global goal (rescue the maximum number of people) with the other robots. It contains only one sub-goal with modules focused on detecting users and rescuing (determining the location of) users when possible. Similarly, the rescue robot testbed also maintains a MAPE-K loop for its own functional requirements. The monitor phase captures the heart rate and location of the people for rescue as environmental conditions. It captures the assigned search area and rescue mode as functional conditions. During the analyze phase, these environmental and functional conditions are analyzed to determine if the robot's integration is valid for performing the rescue mission, which is handled by *rescueProcedure*.

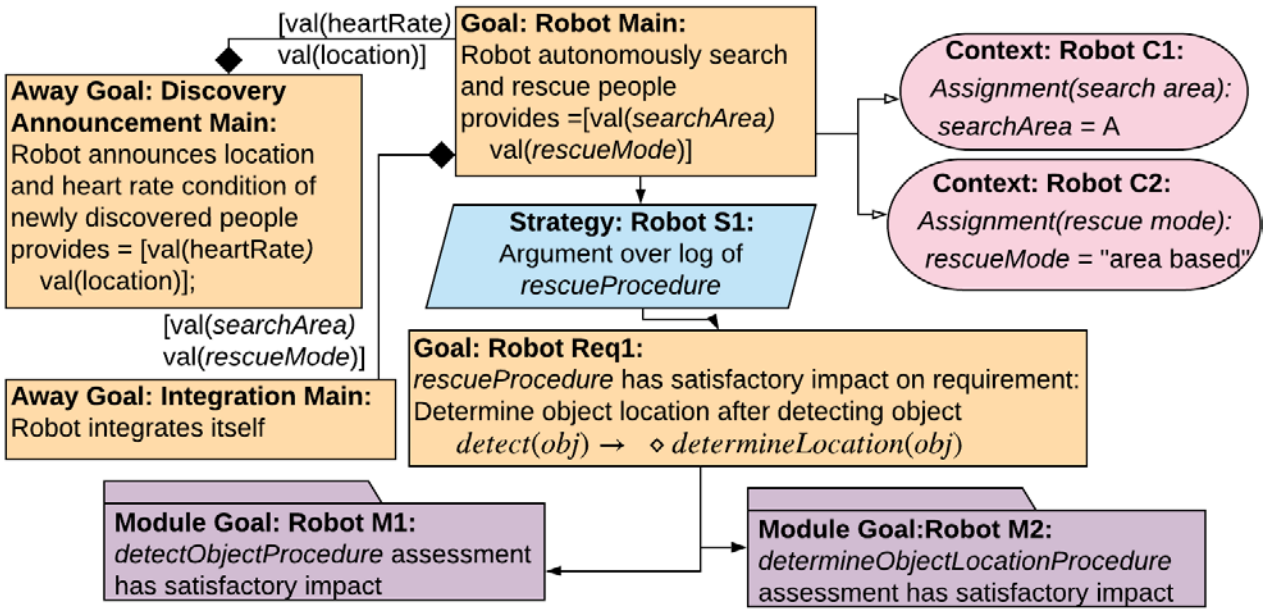

Figure 17: Autonomous Robot Functional Assurance Case

4.1.3 Deploying and Evaluating the Interaction Protocol

We designed and implemented an interaction protocol based on the Regional Planning Pattern (Weyns, 2013). The purpose of the interaction protocol is to standardize a communication pattern that processes can use to effectively coordinate their adaptation (Jahan, 2020a). The Regional Planning pattern was designed so that processes with separate regions of concerns could coordinate during the planning step to have their adaptation decided by a third process, referred to as the coordination manager, which can be deployed on a separate system or as a subprocess, as is the case with our testbed. The coordination manager is responsible for selecting an adaptation that will be executed by both coordinating processes. It is not specified by the Regional Planning pattern how the coordination manager should select an adaptation, so we have taken the liberty of defining an interaction protocol that is robust enough to be deployed by a variety of processes.

According to the Regional Planning pattern, coordination takes place during the planning step. One process must initiate, and the other process must respond. One process may not reach its respective planning step at the same time as the other process. As such, either process can wait during its planning step by blocking for a period of time, as specified by an input value, or until it receives a response. The process that initiates the coordination is termed the initiator and the other process is termed the participant. If the blocking process does not receive a response before its waiting period has expired, then the processes do not coordinate.

The interaction protocol is conducted as follows. Once they reach their planning step, the MAPE-K process and the MAPE-SAC process prompt the coordination planner to generate a set of adaptation plans and block until the coordination planner is finished. Once the coordination planner has generated a set of adaptations, it sends the set of adaptations to both the MAPE-K and MAPE-SAC processes, both processes become unblocked, and the coordination planner blocks until it receives a risk assessment from the MAPE-K process and a degree of compliance confidence assessment from the MAPE-SAC process. The MAPE-K and the MAPE-SAC processes translate each adaptation to a change set and then apply associated risk assessment methods on each change set. The results are sent to the coordination planner. It is left to each process to select and perform the assessment process according to its requirements. By separating the logic of ranking adaptations, which is performed by the MAPE-K and MAPE-SAC processes, from the logic of selecting an adaptation, which is performed by the coordination manager, the interaction protocol remains robust. Any process that can perform any kind of assessment of the adaptation changes in accordance with the requirements of the coordination manager can utilize the interaction protocol to interact with any other process that can also perform a similar assessment; the two methods of assessment do not need to be the same.

After receiving the assessment results from both the initiator and the participant, the coordination planner decides on a plan as follows. First, it selects the highest ranked plan (i.e., least risk). If the plan selected from the initiator is in the top half of the participant plan rankings, then the plan is accepted. Otherwise, the plan from the initiator is rejected and the coordination planner selects the next best plan from the participant and the plan ranking review process continues. Similarly, if the highest ranked plan is selected from the participant and is in the top half of the initiator's ranked plans, it is accepted. Otherwise, the plan selected from the participant is rejected and the coordination planner selects the next best plan from the initiator and the plan ranking process continues. The coordination manager alternates between reviewing the next highest rank plan from the initiator and reviewing the next highest ranked plan from the participant so that both the initiator and the participant each has the opportunity to have their best plan selected. This process repeats until either a plan is selected or until there are no more plans. Once the outcome is decided, the plan is communicated to the MAPE-K and MAPE-SAC processes, unblocking both processes and enabling them to proceed to their execution steps.

The interaction protocol extends our initial conceptual work in the MAPE loop interaction framework (Jahan, 2019b). Previously, to select an adaptation, the initiator would send a sequence of adaptations to the participant who would evaluate each adaptation individually and either accept

or reject the adaptation. The initiator would accept the first adaptation accepted by the participant and reject any adaptation rejected by the participant. If the participant rejected all adaptations, then the system would enter a default state and no adaptation would be applied. Our current work differs from our prior work in that (a) no adaptation is assessed individually as all adaptations are known by both the initiator and the participant, (b) the initiator can reject an adaptation that has been accepted by the participant, and (c) at least one adaptation must be accepted by both the initiator and the participant, i.e., the system cannot enter a default state.

4.1.4 Reviewing the Possible Adaptations

By default, the wearable systems prioritize the security of personal information and therefore do not allow fostering as shown in Figure 18 (left side). However, in a disaster scenario, fostering is needed by the autonomous rescue robot to help determine the condition of the victim. We restate the original adaptations defined in Section 2 as part of the simulation for wearable devices to assist a rescue mission in a disastrous area.

A1: Stay connected, send empty packets, no fostering is allowed
A2: Stay connected, send empty packets, fostering is allowed
A3: Get disconnected, no fostering is allowed
A4: Get disconnected, fostering is allowed

A wearable's interactive MAPE loops evaluate the adaptation plans by coordinating themselves to allow fostering as a functional adaptation as shown in Figure 18 (right). However, fostering violates security control IA-3 as written. Other interrelated security controls realize the operational circumstance and prioritize access to rescue-relevant data over identification and authorization as defined by IA-3. Once the context of *accessControlPolicy*, defined in security control AC-3, and *connectionRequirements*, defined in AC-18 (Figure 13), have been changed, the coordination between the MAPE loops yields an optimal system-wide adaptation. To maintain rescue mission objectives, the adaptation must prevent communication while remaining connected and allow fostering, if needed, in order to connect to a rescue robot.

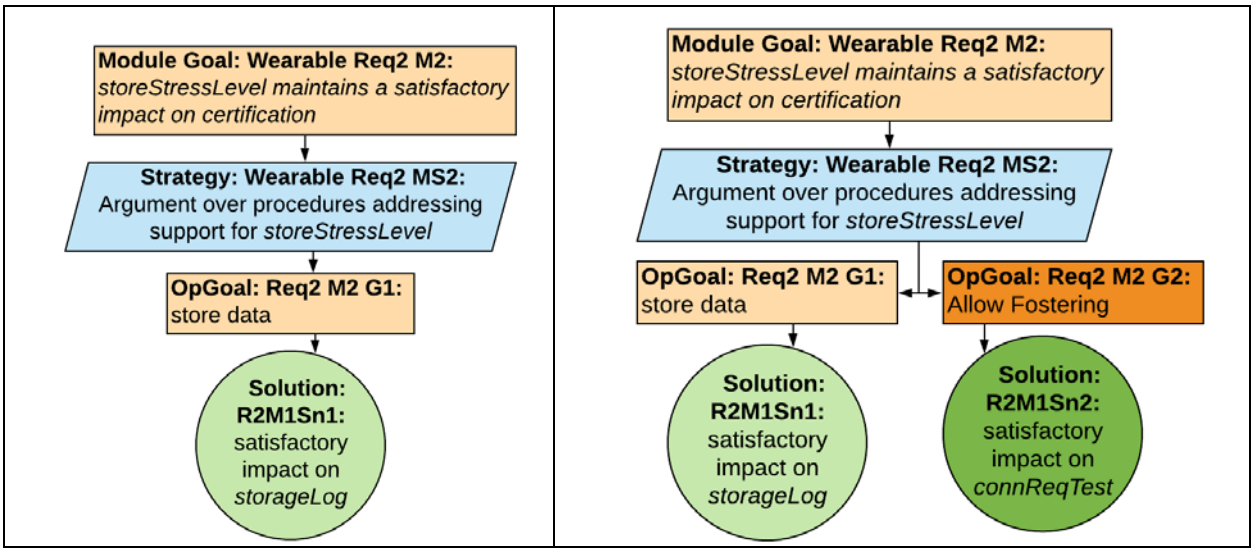

Figure 18: Expanded Wearable Req2 M2 (left: before adaptation, right: after adaptation)

4.1.5 Evaluating the Interaction Framework Use

We implemented and tested the MAPE-K loop, MAPE-SAC loop, and coordination planner on the wearable testbed. We provided an example of the heart rate variability monitor to validate the above scenario. The wearable device is connected to its base station via Bluetooth, sending data to simulate heart rate information. If the system detects the need for an adaptation, it must determine an optimal system configuration.

To evaluate the MAPE loops and their coordination, we designed a series of experiments using the four previously described adaptations: A1, A2, A3, and A4. We assume that a coordination planner implemented on a wearable device can dynamically generate adaptation plans. Further, we assume that the planners used in the MAPE-K and MAPE-SAC loops can translate those plans into a change set, respective to their region of concerns, and that they can perform assessment of changes for compliance of their requirements.

Security compliance is assessed by determining the satisficing level of the main security control goal included in SAC model. Here, a low satisficing level implies higher risk of non-compliance with a security requirement. We represented each main security control goal as a softgoal and interdependencies among the argumentations of the goals as a Soft goal Interdependency Graph (SIG) (Jahan, 2019a). The impact of the changes associated with state variables and their propagation toward the main security control goals are considered to calculate the achievement weight, a_w for each of the goals within the argumentation, which is used to assess compliance degradation.

$a_w(g) = I(g)$, for leaf nodes,
$= average\big(a_w(c)\big)$, for all $c \in children(g)$ for non-leaf nodes

The compliance degradation of a security control main goal can affect other related security controls and cause signification compliance degradation in the overall system. After determining achievement weight for each main security control goal, we calculated the satisficing level, SL for all interrelated main security controls.

$$SL(m) = average(a_w(m) + \sum_{g \in neighbors(m)} a_w(g))$$

Functional compliance is assessed by determining the risk of reusing the original proof of verification process, as changes may inhibit reusing original proof (Marshall, 2018a). The planners then supply their risk assessment to the coordination planner, which then selects an adaptation plan based on the ordering of the adaptation plans across the two provided risk assessments. For our risk assessment, we produced a *changeSet* to determine (1) what state variables are affected due to change, (2) what are the conditions of the changes, and (3) how do the changes impact the state variables. The risk assessment process uses this information to identify conflicts that arise during the verification process due to the changes and to alert when there are potential conflicts. The alerts include all of the impact multipliers needed for the risk assessment on the changes, which are (1) the impact on state variables due to changes, M_{VC}, (2) the impact on component due to changes, M_{PL}, and (3) what the planner believes the impact on state variables is, $\hat{p}$. We multiply these impact multipliers together to estimate the probability of risk associated with each alert, t.

$$\boldsymbol{p(t) = M_{VC}(t)\, M_{PL}(t)\hat{p}\,(t)}$$

Then, we calculate the expected utility of the adaptation plan using the following utility function.

$$\boldsymbol{E = \sum_{r \in R} w(r) \prod_{t \in T} p(t)}$$

Here, $w(r)$ is utility weight of requirements needed to maintain system compliance. Higher expected utility means the adaptation is less risky, and adaptation plans are ranked based on their expected utility.

For the purposes of our experiments, the coordination planner provides the four aforementioned adaptation plans: A1, A2, A3, and A4, to the MAPE-K and MAPE-SAC planners which, in return, provide their risk assessment and satisficing level of the four adaptation plans.

Three experiments are conducted to evaluate the MAPE-K and MAPE-SAC coordination on the wearable security testbed. The first two experiments are designed to evaluate the expected best and worst behavior of the MAPE loop coordination, as determined by the ordering of the

adaptation plans provided by the MAPE-K and MAPE-SAC planners. The best case is that the orderings of the risk assessments are the same, and the worst case is that they are the reverse of one another. The final experiment is designed to provide an assessment of the MAPE loop coordination across all possible wearables. MAPE-K and MAPE-SAC planners can provide a wide variety of risk assessments based on the concerns of the wearable system on which they are implemented.

Each experiment comprises three processes. The first process manages the MAPE-K loop, the second process manages the MAPE-SAC loop, and the third process manages the coordination planner. The MAPE-K process loops through its monitor and analyze steps, as does the MAPE-SAC process, constantly checking to determine if the system is still secure. The current security state of the system is represented as a global Boolean flag that is set to true if the wearable system is in a secure state or false if the wearable system is in an insecure state. If the monitor step determines that the wearable system is still in a secure state, then no adaptation is needed and the plan and execute steps produce no adaptation.

After all threads are started, the state of the wearable system is flipped to insecure. Both MAPE processes then observe that the wearable system is insecure in their respective monitor steps, determine that an adaptation is needed in their respective analyze steps, and interact with the coordination planner in their planning step.

For the first experiment, both the MAPE-K and the MAPE-SAC processes provide the same risk assessment to the coordination planner: A1, A2, A3, A4. The first set of plans selected by the coordination planner are A1 and A1. Since the first set of plans match, the coordination planner decides on A1.

For the second experiment, the MAPE-K process provides the following risk assessment: A1, A2, A3, A4, and the MAPE-SAC process provides the following risk assessment: A4, A3, A2, A1. The coordination planner decides on plan A3. The MAPE-K process communicated first and became the initiator. Since A1 (the highest ranked plan from the MAPE-K process) is in the bottom half excluding the middle of the risk assessment from the MAPE-SAC process, it is rejected and A2 is selected. Then, since A4 (the highest ranked plan from the MAPE-SAC process) is in the bottom half excluding the middle of the risk assessment from the MAPE-K process, it is rejected and A3 is selected. Since A2 is ranked in the top half inclusive by the MAPE-SAC, the coordination planner decides on A2. If the MAPE-SAC had been the initiator, then the coordination planner would have decided on A3 for the same reason.

In the third and final experiment, we iterate over every possible permutation of A1, A2, A3, and A4 for the MAPE-K loop. For each permutation, we iterate over every possible permutation of A1, A2, A3, and A4 for the MAPE-SAC loop. There are 24 possible permutations of each set for a total of 576 trials. Each plan decided by the coordination planner is recorded, the results are aggregated into Table 3. After each trial, the system is reset so that the initial conditions of each trial are the same.

Table 3 shows the results gathered from the evaluation conducted in the third experiment. From this table, we can see that plan A1 was selected 146 times, A2 was selected 135 times, A3 was

selected 147 times, A4 was selected 148 times, and in 0 cases there was no agreement between the MAPE-K and the MAPE-SAC loops. Based on these results, we can see that each adaptation plan is selected with roughly equal likelihood, as should be expected. In addition, we can see that if no coordination takes place, then in 42% of the trials, a violation will occur in either the functional concerns or the security concerns. A violation occurs when a functional adaptation or a security adaptation causes significant degradation in the system's compliance with its security requirements or functional requirements, respectively. Whereas, if coordination does take place, then a violation will occur in 0% of the trials with the adaptations selected.

Table 3: Results from implementing the interaction protocol with 576 trials

Plan	Count	Percentage
A1	**146**	**0.25**
A2	**135**	**0.23**
A3	**147**	**0.26**
A4	**148**	**0.26**
Violations with Interaction	**0**	**0**
Violations without Interaction	**240**	**0.42**

We also examined the average position of the chosen adaptation for both the MAPE-K and MAPE-SAC loops. Ideally, this average should be similar over the 576 trials and between 1 and 2, indicating that the first or second plan of the MAPE-K and MAPE-SAC are chosen more often than not. In our trials, the MAPE-K average position was 1.66, while the average position of the MAPE-SAC was 1.61. These numbers are very similar and, importantly, very close to the optimal position. It is important to observe this value as each time an adaptation is rejected, the next adaptation is less satisfactory for either the MAPE-K or MAPE-SAC process. In our experiment the MAPE-K was more often the initiator. This results in the MAPE-SAC plan being prioritized when planners disagree.

4.2 Extending the MAPE-SAC Implementation to 3rd Party Technologies

To evaluate the use of 3rd party technologies within the research effort, we expanded the MAPE-SAC loop specifically to incorporate Enki (Langford, 2019b), as implemented and packaged by Michigan State University, and Darjeeling, as implemented and package by Carnegie Melon University (Jahan, 2020b). Enki generates scenarios by considering uncertainties for the system. Darjeeling generates patches for code repair on test case failures, happens on uncertain situation[11]. We maintained the risk assessment and security compliance degradation assessment technologies for use on Darjeeling generated patches to determine the best adaptation. Figure 19 shows the connectivity and functionality needed within the MAPE-SAC loop and assessment processes in order to use Enki and Darjeeling as black box components.

[11] https://github.com/squaresLab/Darjeeling

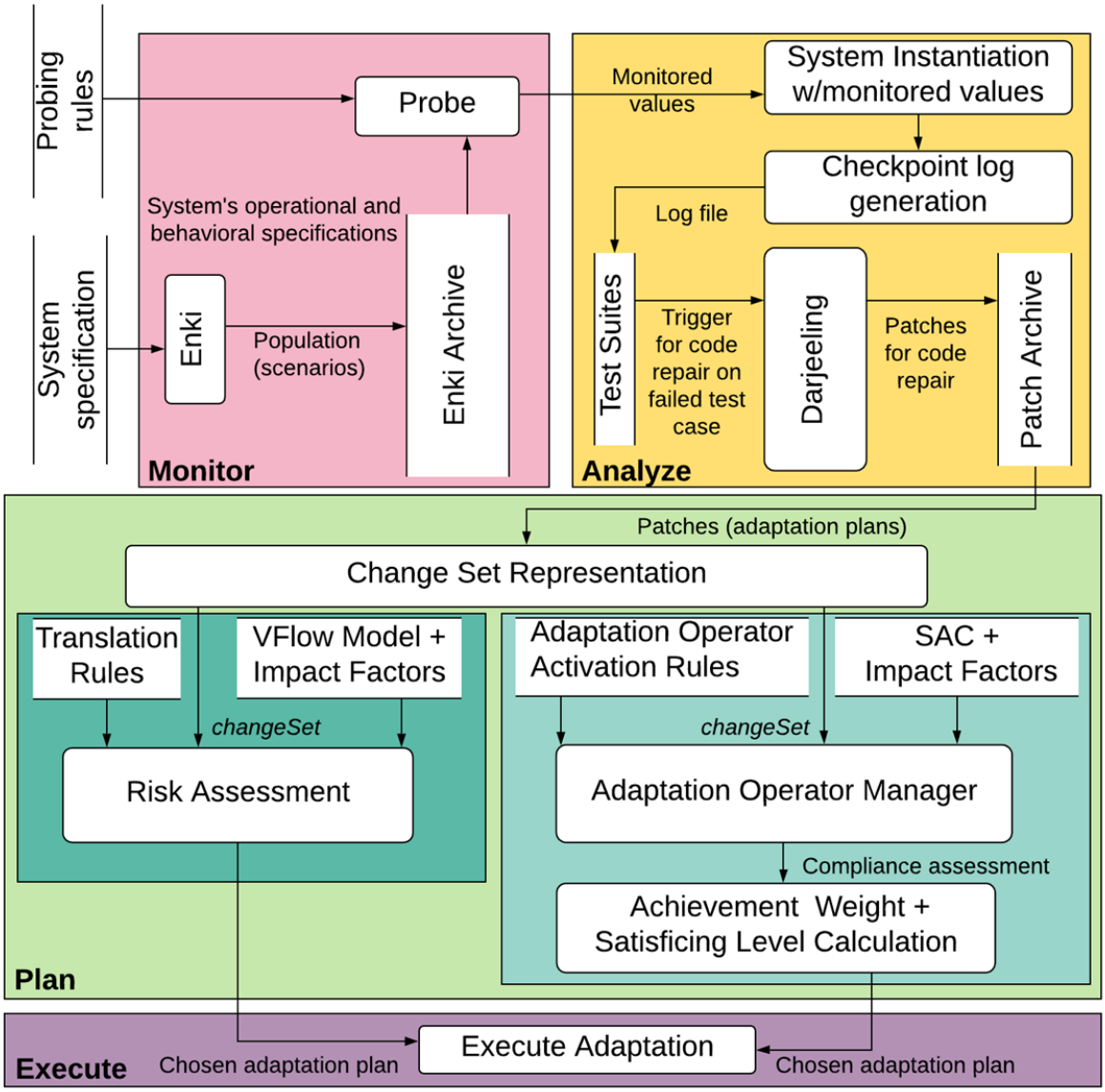

Figure 19: Enki, Darjeeling and SAC framework incorporation

4.2.1 Generate potential scenarios to monitor Using Enki

The key challenges with Learning-Enabled Systems (LES) is determining whether training data is enough to ensure the LES is resilient to environmental uncertainty and how to obtain better training data to improve the system's performance when it is not (Langford, 2019b). Enki is used to assess environmental uncertainty. Enki performs novelty search to automatically discover operational conditions that result in the most diverse system behavior of a System Under Test (SUT). The specification for the operational conditions and variables, which introduce uncertainty with corresponding ranges of values, are provided by the user for Enki to generate potential scenarios to monitor. We applied Enki to our Multi-Mode Traveler System (MMTS) application to generate scenarios with different combinations of key state variables from prior work (Marshall,

2018). MMTS is a simulation of how a "traveler" moves on a grid with enemies statically placed in random positions. The traveler moves while avoiding enemies and maintaining a fuel threshold range. The traveler's fuel level along with the minimum and maximum fuel levels are initially set. A move may increase, decrease, or maintain the current level of fuel in the traveler's reserves depending on which move is chosen. That is, the traveler's fuel is not associated with its current position, but how it travels along the grid. Given $p = (x, y)$ is the traveler's position, we worked with three MMTS requirements shown in Table 4 to illustrate the integration of the 3rd party components.

Table 4: MMTS requirements and their LTL representation

	Statement	LTL Expression	Type
MR1	The fuel level must satisfy the defined threshold	$\square(minFuel \leq fuel \leq maxFuel)$	Invariant
MR2	The traveler must not enter an enemy position	$\square(p \notin enemyPos)$	Invariant
MR3	If the traveler can move, it must move	$\square(canMove(p) \implies \diamond[\exists q : q \neq p : moveTo(q)]$	Progress

Enki requires an executable script for each application (Langford, 2019b). Every executable script contains three items:

1. An operation specification property that returns a dictionary of values for Enki to explore.
2. A behavior specification property that returns a dictionary of values for Enki to diversify.
3. A execute function that takes a dictionary of values that matches operation specification and returns a dictionary of values that matches behavior specification.

From our MMTS requirements, we identified variables that may have uncertain behaviors (Jahan, 2020b). We specified the operation specification and behavior specification of those variables and wrote the execution to specify how Enki should evaluate each selected individual in its search as shown in Figure 20, Figure 21, and Figure 22. The operation specification in Figure 20 defines the search space for Enki. It is expected to be in the form of a dictionary, where each entry is a separate operational parameter for the system being assessed by Enki. The key for each entry in the dictionary will be the name of the operational parameter, and the value for each entry will be a range of permissible values for the operational parameter. We specify the operational specifications for MMTS as shown in Figure 20 to reflect the following specifications:

- Specification of fuel range (fuel, maxFuel, and minFuel in Figure 20).
- Specification of target collections (1-4 in Figure 20), and
- Specification of enemy positions (1-8 in Figure 20).

```
def operation_specification(self):
    return {
        'fuel': [0, 100],
        'maxFuel': [0, 100],
        'minFuel': [0, 100],
        'enemyPos1': [True,False],
        'enemyPos2': [True,False],
        'enemyPos3': [True,False],
        'enemyPos4': [True,False],
        'enemyPos5': [True,False],
        'enemyPos6': [True,False],
        'enemyPos7': [True,False],
        'enemyPos8': [True,False],
        'target1': [True,False],
        'target2': [True,False],
        'target3': [True,False],
        'target4': [True,False],
    }
```

Figure 20: Operational specification for MMTS state variables within executable script

The behavior specification in Figure 21 defines which values Enki will consider when ranking based on the novelty score of each individual set of operational parameters selected during its search. These parameters may or may not include the parameters defined in the operation specification (Figure 20). The format of this property is identical to the operation specification: a dictionary with each entry defining a parameter name and permissible values. We specify the behavioural specification for MMTS based on whether it can move (canMove), its current position (currentPos), where it can move to (toMove), and what that next position can be (nextPos) as shown in Figure 21.

```
def behavior_specification(self):
    return {
        'canMove':[True, False],
        'currentPos':[True, False],
        'toMove':[True, False],
        'nextPos':[1, 8],

    }
```

Figure 21: Behavioral specification for MMTS state variables within executable script

The execute function in Figure 22 defines how Enki will evaluate each selected set of values for the given operational parameters. Each value for each key will be within the ranges defined in the operational specification. The intention is for the execute function to set up an environment that reflects the provided operational values, execute some functionality under that context, and return the results. It takes as input a dictionary of values with keys matching the operational specification (Figure 20), and is expected to return a dictionary of values with keys matching the behavior

specification (Figure 21). In Figure 22, the execute function for MMTS is shown, which includes all of the operational and behavioral specifications along with reflecting the functionality to choose next position by avoiding enemies (MR2) and allow moving to next position if the fuel threshold is maintained (MR1) and canMove is set to true (MR3).

```
def execute(self, operation_values):
    fuel = operation_values['fuel']
    maxFuel = operation_values['maxFuel']
    minFuel = operation_values['minFuel']
    enemyPos1 = operation_values['enemyPos1']
    enemyPos2 = operation_values['enemyPos2']
    enemyPos3 = operation_values['enemyPos3']
    enemyPos4 = operation_values['enemyPos4']
    enemyPos5 = operation_values['enemyPos5']
    enemyPos6 = operation_values['enemyPos6']
    enemyPos7 = operation_values['enemyPos7']
    enemyPos8 = operation_values['enemyPos8']
    target1 = operation_values['target1']
    target2 = operation_values['target2']
    target3 = operation_values['target3']
    target4 = operation_values['target4']        .
    nextPos =  1 if enemyPos1==False else 2 if enemyPos2==False else 3
               if enemyPos3==False    else 4  if enemyPos4==False else 5
               if enemyPos5==False else 6 if enemyPos6==False else 7
               if enemyPos7==False else 8 if enemyPos8==False else 0
    canMove =  True if fuel<=maxFuel and fuel>=minFuel and nextPos!=0 else False
    currentPos =  True
    toMove =  canMove
    behavior_values = {
        'canMove': canMove,
        'currentPos': currentPos,
        'toMove': toMove,
        'nextPos': nextPos,
    }
    return behavior_values
```

Figure 22: A execute function to generate the Scenario

Enki implements an evolution-based search process and generates populations of individuals by executing the provided executable script (Langford, 2019b). Each population represents a different scenario of selected environmental effects. The population is then stored into the archive and the novelty score is updated. The resulting archive contains a set of scenarios. In Figure 23, one of population generated by Enki for MMTS is shown, where the novelty score and the resulted operational and behavioral parameters values are considered as an instance of a scenario.

```
"score": 0.2261904627084732,
"operation_values": {
    "enemyPos1": true,
    "enemyPos2": true,
    "enemyPos3": false,
    "enemyPos4": true,
    "enemyPos5": false,
    "enemyPos6": true,
    "enemyPos7": true,
    "enemyPos8": true,
    "fuel": 16,
    "maxFuel": 99,
    "minFuel": 3,
    "target1": true,
    "target2": false,
    "target3": false,
    "target4": false
},
"behavior_values": {
    "canMove": [
        true
    ],
    "currentPos": [
        true
    ],
    "toMove": [
        true
    ],
    "nextPos": [
        3
    ]
}
```


Figure 23: Example of population generated by Enki from the archive

We have developed a parsing tool to parse the archive and set rules to probe the scenario being monitored. The monitored values are logged and used for instantiating the MMTS system.

4.2.2 Generate Patches for Code Repair Using Darjeeling

Darjeeling uses BugZoo[12] to keep track of the associated passing and failing test cases, as well as the lines that are run in each. The information from BugZoo is used by Darjeeling to create a weighted path of possible repair locations. MMTS has an instantiation method as shown in Figure 24, and we have instantiated the MMTS application with the set of monitored values from each scenario generated by Enki as shown in Figure 25.

[12] https://squareslab.github.io/BugZoo/

```
void setup(int x, int y, int fuel, int minFuel, int maxFuel,  int enemyPos, int [] target, int priorityPath) {
    fuel = Set_Fuel(fuel);
    min_fuel = Set_MinFuel(minFuel);
    max_fuel = Set_MaxFuel(maxFuel);
    _x = Set_PositionX(x);
    _y = Set_PositionY(y);
    enemy_Position(ENEMY_POSITION, enemyPos);
    x_Coordinate(X_POSITION);
    y_Coordinate(Y_POSITION);
    printf("x = %d, y = %d", x, y);
    printf("_x = %d, _y = %d", _x, _y);
}
int update(int caseId)
{
    int proceed = 0;
    int fuelCheckPassed = Check_FuelConsistency(fuel, min_fuel, max_fuel);
    printf(" x = %d, y =%d and Fuel = %d\n", _x, _y, fuel);
    // gCS start
    if (fuelCheckPassed)
{
```

Figure 24: Instantiation method for MMTS

```
int main() {
    int target[] = { 0, 1, 1, 0 }
    setup(0, 0, 7, 66, 19, 2, target, 4);
    for (int i = 0; i < 140; i++) {
        update(i);
    }
```

Figure 25: Instantiation of MMTS

We embedded Checkpoint Log within the system to collect the system status with the instantiated value. Checkpoint collects the log shown in Figure 26 whether

- Maintaining fuel consistency,
- Avoiding enemies, or
- Making a next move when valid path is available.

```
0;14-09-2017:12.12.05;1st Check Fuel;true
0;14-09-2017:12.12.05;1st Check Enemy;true
0;14-09-2017:12.12.05;Valid Positions NonEmpty;true
0;14-09-2017:12.12.05;New Position;true
0;14-09-2017:12.12.05;2nd Check Fuel;true
0;14-09-2017:12.12.05;Change Position;true
0;14-09-2017:12.12.05;2nd Check Enemy;true
```

Figure 26: Check Point logs for MMTS

Unavailability of any valid path to reach a target using path priority rules produces a failed test case. Failed test cases in checkpoint simulations trigger Darjeeling. By running Darjeeling, we discovered over 300+ candidate repairs (Jahan, 2020b). However, most of the repairs violate MMTS constraints. Two of which are relevant.

1. Ignore enemy check while generating valid path list, which may affect enemyPos variable due to changes.
2. Introduce new priority path to reach the target, which may affect priorityPath variable due to changes.

In Figure 27, patch 1 is shown. It deletes statements that handle exploring the grids and checking for current enemy positions. It also appends a statement to disable the 4th priority path. This patch causes a violation of requirement MR2 in Table 4. In Figure 28, patch 2 is shown, which assigns the 3rd priority path as the active one for reaching the targets.

```
        int enemyPos = 0;
        for (int x = 0; x < MAX_SIZE; x++)
        {
-               for (int y = 0; y < MAX_SIZE; y++)
-               {
-                       if (X_POSITION[x][y] == _x && Y_POSITION[x][y] == _y)
-                       {
-                               if (ENEMY_POSITION[x][y] == 1)
-                               {
-                                       //waiting = 1;
-                                       enemyPos = 1;
-                               }
-                               else
-                               {
-                                       enemyPos = 0;
-                               }
-                               break;
-                       }
-               }
-       }
+               priorityPath[3] = 0;    }
        return enemyPos;
 }
 int getChangeFuel(int _x, int _y, int _newPosX, int _newPosY)
```

Figure 27: A Darjeeling generated patch by removing enemy check conditions

```
--- MMTS/BaseComponent.c
+++ MMTS/BaseComponent.c
@@ -345,7 +345,7 @@
                case 13:
                        priorityPath[2] = 1;
                        moveTraveler(priorityPath, moveNumber);
-                       priorityPath[2] = 0;
+                       targetsCollected[0] = 1;
                        break;
                default:
                        moveTraveler(priorityPath, moveNumber);
```

Figure 28: A Darjeeling generated patch by changing priority path

4.2.3 Incorporating SAC framework for adaptation assessment

To incorporate the SAC framework into the Adaptation Assessment Framework, we have developed a security assurance case for our MMTS application (Jahan, 2020b). The SAC framework deals with security requirements and interacts with functional requirements to investigate any functional changes that cause compliance degradation with respect to security requirement. For the MMTS application, we have selected the security control SI-7, which is the parent of SI-7(5) used earlier and employs integrity verification tool for unauthorized changes. The security assurance case is shown in Figure 29. MMTS adopts this SAC to comply with the security requirements that the system's integrity is maintained by not allowing any anomalous changes. The security requirement for complying with security control SI-7 relies on the functional requirement of *integrityCheckerTool* and the underlying lower level functionality. The functional requirement for *integrityCheckerTool* is to generate a flag when anomaly is detected. To do so, information about MMTS's state is logged and analyzed by the lower level Opp Goals (M1 G1-G7 and M2 G1 in Figure 29). On the other hand, our verification workflow technology (Marshall, 2018a and 2018b) estimates the potential risk due to the changes as a part of impact verification and determines the *integrityStatus*. ProM visualizations of the execution traces are used as evidence reflecting the change impact on integrityStatus.

We perform a security compliance degradation assessment on this security assurance case due to the changes that occur in patch 1 and patch 2 from Darjeeling.

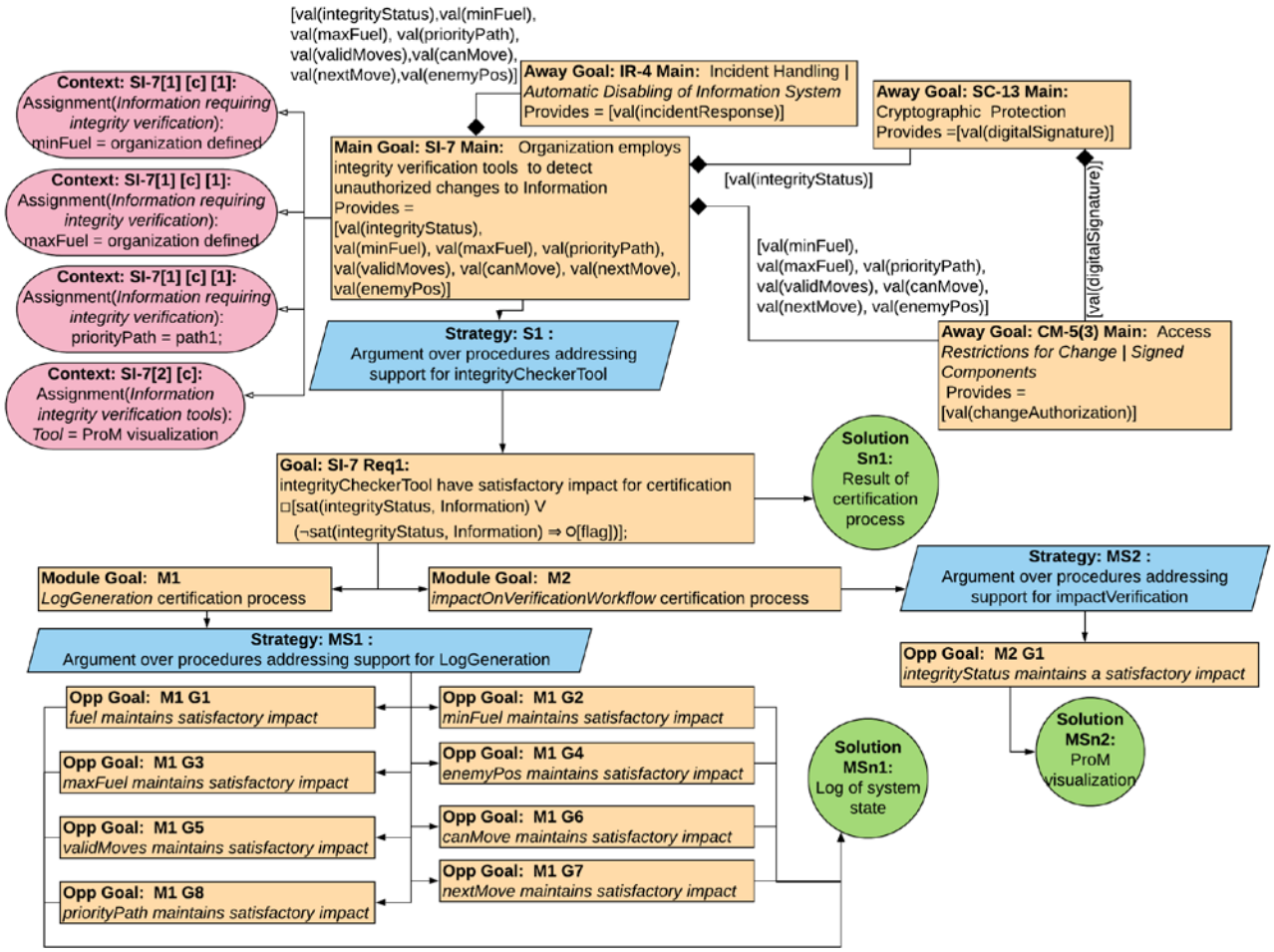

Figure 29: Security Assurance Case for SI-7

4.2.3.1 Change set for adaptation changes

To perform an assessment of the adaptation, we developed the Adaptation Operator Manager (AOM), which accepts a structured form of change representation and activates adaptation operators to modify security assurance cases (Jahan, 2018a). The AOM considers two high-level types of adaptations: (1) those that make a direct change to the state variables within the code, and (2) those that indirectly alter behavior of the state variables by modifying the code. For type (2), it considers two subcategories: (i) those that introduce new functionality in support of the system's goals, and (ii) those that replace functionality with new functionality that has the same interface. Based on these categories, we define three adaptation operators derived from change representation.

- *ChangeVal* when the adaptation invokes a direct change of state variables to a new state.
- *Support* when the adaptation introduces conditional changes to an existing state variable by including new dependencies on other variables.

- *Substitute* when functionality must be replaced without introducing any new dependencies on other variables.

AOM uses rules shown in Figure 30 to extract change requirements to determine how a SAC should be evolved to meet the adaptation requirements.

Rule 1:
IF *newVar = null & newFunc = null*
THEN *ChangeVal(stateVar, newState, evidence, rationale)*

Rule 2:
IF *newVar ≠ null & newFunc ≠ null*
THEN *Support(stateVar, newVar, newFunc, evidence, rationale)*

Rule 3:
IF *newVar = null & newFunc ≠ null*
THEN *Substitute(stateVar, newState, newFunc, evidence, rationale)*

Figure 30: Rules to adapt security assurance cases

We have manually transformed code changes in the patches as a form of *ChangeSet* that AOM can craft to trigger the compliance assessment process. *ChangeSet is* formulated around state variables affected by the change.

ChangeSet = {(stateVar, newState, changeCond, evidence, rationale, changeImpact)$_1$,…, (stateVar, newState, changeCond, evidence, rationale, changeImpact)$_N$}

Patch 1, which removes the condition to check for current enemy positions, impacts the *enemyPos* variable, and its new state becomes ignored. This patch disables *enemycheck* functionality and has a devastating impact. We have constructed the following *ChangeSet* for patch 1.

ChangeSet = {(enemyPos, ignored, (null, disableEnemyCheck), log of system state, fail to collect target, 0.2)}

AOM activates the *Substitute* operator according to Rule 3 in Figure 30, since the change doesn't introduce any new dependency but does introduce new state variables. This change applies to the operational goal associated with *enemyPos* as shown in Figure 31.

Substitute(enemyPos, ignore, disableEnemycheck, log of system state, , fail to collect target)

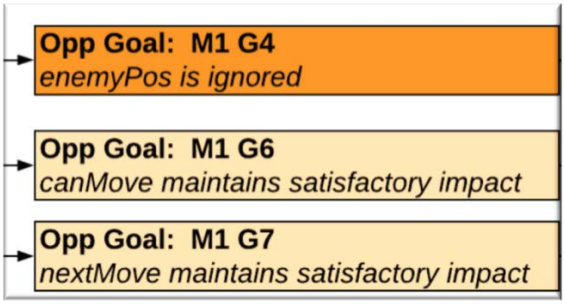

Figure 31: Adaptation change in enemyPos by Substitute operation

Patch 2, which changes the priority path for collecting targets, impacts *priorityPath* and produces a new value for *path2*. This patch does not introduce any new variables or change any functionality. The change has an unconcerned impact on the system's compliance with its requirement. We have constructed the following *ChangeSet* for patch 2.

ChangeSet = {(priorityPath, path2, (null, null), log of system state, fail to collect target, 0.9)}

AOM activates the *ChangeVal* operator according to Rule 1 in Figure 30, as the change only assign a new priority path but doesn't introduce new dependencies or any new state variables. This change applies to the context node of the SAC as shown in Figure 32.

ChangeVal (priorityPath, path2, log of system state, fail to collect target)

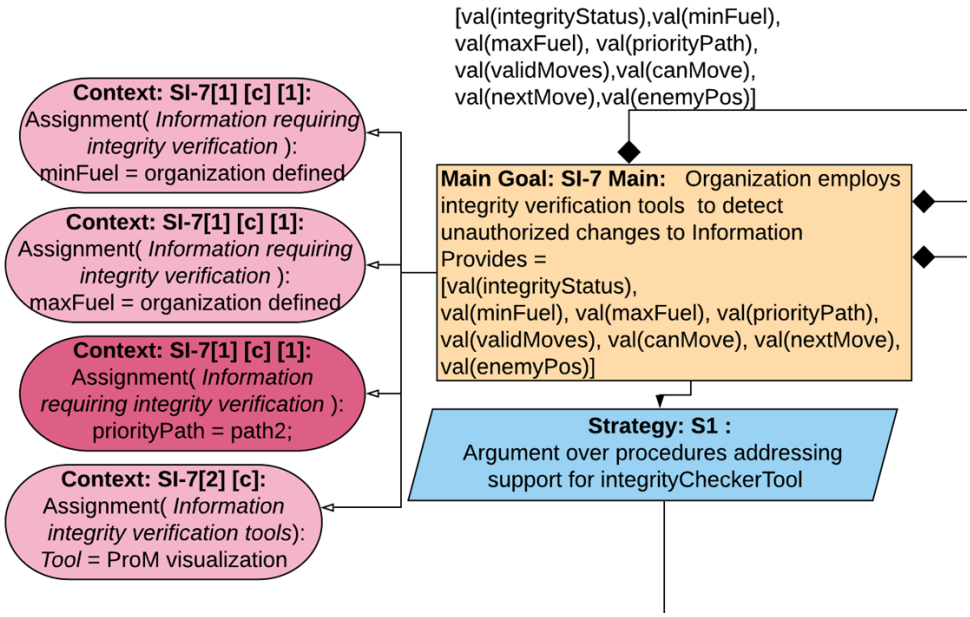

Figure 32: Adaptation of context node to change priority path by ChangeVal Operation

4.2.3.2 Adaptation Risk Assessment

We have defined rules to extract key concerns and conditional impacts from the proof process of the verification workflow (VFlow) (Marshall, 2018b). We rely on these predefined rules to determine the change impact. We have categorized the conditions into three groups, which are 1) Devastation: has high potential to violate the requirements, 2) Worrisome: has potential to violate the requirements in near future; and 3) Unconcerned: less potential to violate the requirements. In Table 5, we have extracted key concerns, which we are termed verification concerns (VCs) as well as their change conditions and impact multipliers.

Table 5: Change impact condition for MMTS

VCs	devastating	worrisome	unconcerned
fuel	Change greater than or equal to $maxFuel - minFuel$, positive or negative.	Change greater than or equal to $\frac{maxFuel-minFuel}{2}$, positive or negative.	Change less than $\frac{maxFuel-minFuel}{2}$, positive or negative
minFuel	Change greater than or equal to $maxFuel - minFuel$, positive.	Change greater than or equal to $\frac{maxFuel-minFuel}{2}$, positive.	Negative change or change less than $\frac{maxFuel-minFuel}{2}$.
maxFuel	Change greater than or equal to $maxFuel - minFuel$, negative.	Change greater than or equal to $\frac{maxFuel-minFuel}{2}$, negative.	Positive change or change less than $\frac{maxFuel-minFuel}{2}$.
enemyPos	ignored		avoided
priorityPath			path2

To perform a verification-based risk assessment of the adaptation changes, the risk assessment approach relies on impact factors from the

- Architecture – flow and priority components,
- Variables of concern (verification concerns) for verifying requirements that may be affected by change, and
- Potential plan ratings based on historical evidence.

The verification-based risk assessment algorithm used by the CPN produces comparison values. The CPN model uses three tokens which traverse the places to collect conflict information.

- Blue token traverses all places to examine every verification concern in *ChangeSet*.
- Pink token represents the *ChangeSet*.

- Red token outputs an alert indicating the number and weight of the issues associated with a plan.

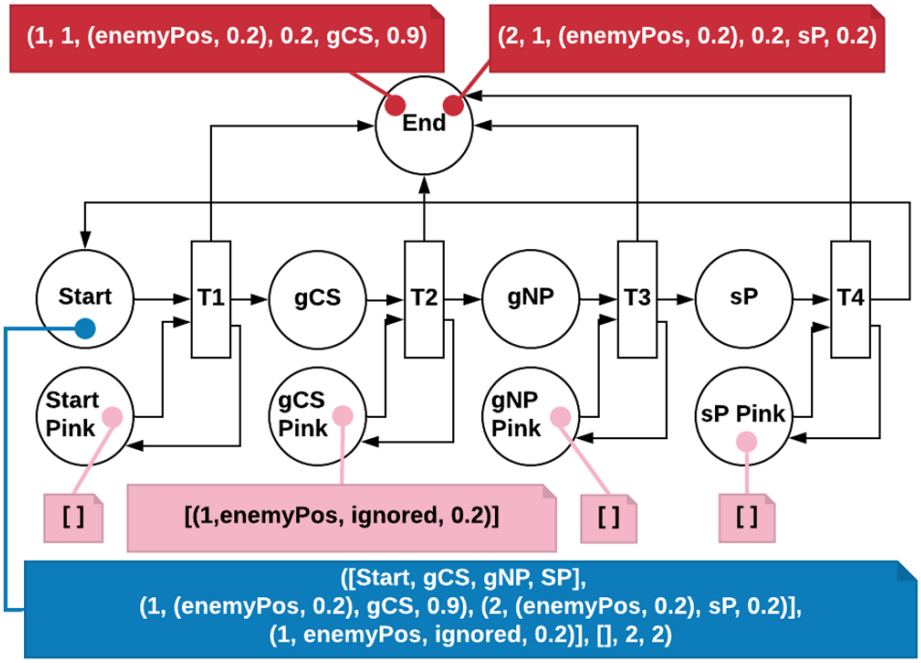

Figure 33: Risk assessment with the change set from patch 1 for requirement MR1

The risk assessment approach identifies two conflicts and introduces two red tokens with the conflict information as shown in Figure 33. The expected utility for patch 1 according to the risk assessment is

$$E[U(patch\ 1)] = (0.75*0.2*0.9*0.2)\ (0.75*0.2*0.2*0.2) + (1*0.2*0.5*0.2)\ (1*0.2*0.2*0.2) + (1*0.2*0.5*0.2)\ (1*0.2*0.2*0.2) = 0.000482$$

On the other hand, changing the priority path in patch 2 only impacts the proof process of requirement MR3.

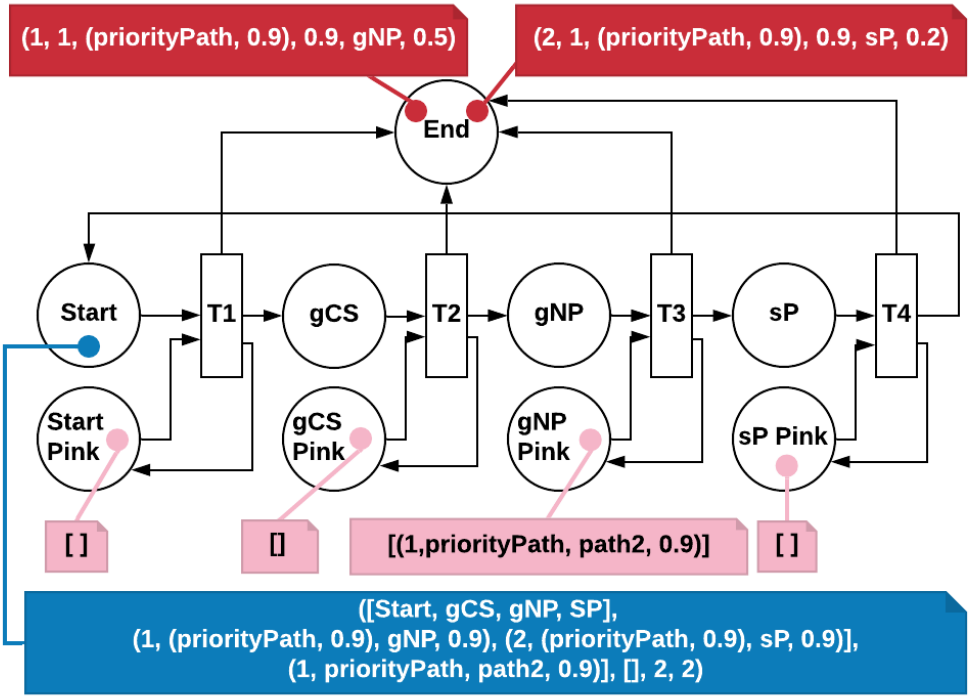

Figure 34: Risk assessment with the change set from patch 2 for requirement MR3

The risk assessment approach identifies two conflicts at places *gNP* and *sP* and introduces two red tokens with the conflict information as shown in Figure 34. The expected utility for patch 2 according to the risk assessment is

$E[U(patch\ 1)] =$ (1*0.9*0.5*0.9) (1*0.9*0.2*0.9) = 0.06561

Patches with higher expected utility have less potential risk when reusing the prior proof process. So, according to the results of the risk assessment, patch 2 is a better option than patch 1 when considering the functional requirements.

4.2.3.3 Adaptation Security Compliance Assessment on Security Assurance Case

Security compliance is assessed by determining the change impact on the SACs. We calculated the Achievement weight and satisficing level to assess the compliance status of the security requirements after considering the functional changes in patch 1 and patch 2. From our verification workflow model and proof process expertise, we determined the impact factors on the state variables (shown in Table 5) that are used to set the achievement weights for operational goals, which propagate upward to high level requirement goals based on the operational goals' importance with respect to the argumentation. The *main* security control's achievement weight contributes to the calculation of satisficing levels across the network of interrelated security controls.

Table 6: Achievement Weight of the goals in SI-7 Security Assurance Case

Goal	Patch 1	Patch 2
Opp Goal: M1 G1	1	1
Opp Goal: M1 G2	1	1
Opp Goal: M1 G3	1	1
Opp Goal: M1 G4	0.2	1
Opp Goal: M1 G5	1	1
Opp Goal: M1 G6	1	1
Opp Goal: M1 G7	1	1
Opp Goal: M1 G8	1	0.9
Opp Goal: M2 G1	1	1
M1	0.9	0.9875
M2	1	1
Goal: SI-7 Req	0.95	0.99375
Main Goal: SI-7	0.95	0.99375

Table 6 shows the achievement weight changes for SI-7 after applying adaptations patch 1 and patch 2 to the security assurance case for SI-7. In Table 7, the satisficing levels of interrelated security controls are shown. Compliance degradation of SI-7 propagates toward the other security controls as they rely on SI-7.

Table 7: Satisficing Levels of Security Controls

	Patch 1	Patch 2
SI-7	0.95	0.99375
IR-4	0.95	0.99375
SC-13	0.95	0.99375
CM-5(3))	0.95	0.99375
Total	3.8	3.975

After calculating the satisficing levels of the applicable security controls within the system, we accumulated the total satisficing levels and ranked the patches to determine the best adaptation plan. With respect to the security requirements, patch 2 is a better option than patch 1 as it maintains a higher satisficing level as shown in Table 7.

In our SAC framework, we employed a coordinated planner (Jahan, 2020a). It coordinates the MAPE-K loop and MAPE-SAC loop, where the MAPE-K loop is dedicated to the assessment of functional requirements and the MAPE-SAC loop handles the assessment of security requirements. In addition, the MAPE-K loop performs the risk assessment of patch changes that affect functional requirements, and the MAPE-SAC loop performs a degree of security compliance assessment on security requirements. The coordinated planner mediates interaction between them to select the best optimal plan for adaptation. When considering both functional and security

requirements, patch 2 is better than patch 1 for our MMTS application. The coordinated planner selected patch 2 as the adaptation plan to be applied to the code and evolves the affected SACs to reflect the selected changes.

5 Conclusion

The technology produced for this effort supports a MAPE-SAC loop interacting with a MAPE-K loop to dynamically provide functional and security assurance as a system adapts. We recognize that coordination rules for planning may be needed to (1) reason about and verify that a concern from one set, or region, of objectives is separate from a concern with the other region of objectives, (2) ensure that the ordering of adaptation triggers does not negatively affect outcomes, and (3) ensure termination of the process if an adaptation cannot be chosen that satisfies both functional and security constraints. We implemented an interaction protocol to support the MAPE-K/MAPE-SAC framework to dynamically maintain functional and security concerns in an adaptive, self-protecting system. We illustrated how such a framework can be used to manage system security requirements, as environmental uncertainty and functional adaptations occur, using several case studies – conceptually on an autonomous rover responding to a potential security incident and fully implemented using two interactive testbeds for an autonomous search and rescue scenario in the event of a natural disaster, while maintaining integrity with respect to security requirements. We introduced the potential for using 3rd party components in the framework by incorporating Enki scenarios for which adaptation may be needed and Darjeeling to produce potential patches representing adaptions.

6 Future Work

Although we have had substantial technological achievements during this project, there is still a great deal of work to be completed to understand the long-term implications of this research. First and foremost is to define the acceptable range of requirement compliance degradation by applying requirement engineering to perform better trade-off analysis and determine the best optimal adaptation. There is difficulty associated with performing a compliance assessment without proper change impact analysis. In the future, we plan to conduct research to enhance the framework with an automated impact analysis approach embedded in the system architecture to decrease reliance on human expertise. We plan to develop a dependency profile to represent interdependencies and assign weights, so that we can determine change impacts for different adaptations. We will apply techniques governing two interacting MAPE loops to large autonomous distributed systems that are exposed to additional sources of environmental and security uncertainty. Additional future work will focus on developing an assurance case patterns catalogue for different system domains, so that our framework is applicable to most domains.

References

(Alexander, 2015) R.D. Alexander, R.D. Hawkins and T.P. Kelly, "Security Assurance Cases: Motivation and the State of the Art," Issue 1.1. CESG/TR/2011/1., University of York, 2015.

(Birman, 1987) K. Birman, and T. Joseph, "Exploiting Virtual Synchrony in Distributed Systems," Proceedings of the Eleventh ACM Symposium on Operating System Principles, 1987.

(Cheng, 2020) B.H.C. Cheng and P.K. McKinley, "X-PLORE: Combining Model-Driven Engineering, Bio-Inspiration And Formal Analysis To Mitigate Uncertainty In High Assurance Software Systems," Final Technical Report, University of Michigan, 2020

(C. Le Goues et al., 2012) C. Le Goues et al., "GenProg: A Generic Method for Automatic Software Repair," IEEE Transactions on Software Engineering, 2012.

(Cox, 2005) J. Cox and E. Durfee, "An efficient algorithm for multiagent plan coordination," In Proc. of the 4th Int'l Joint Conf. on Autonomous Agents and Multiagent Systems, 2005.

(Daniş, 2017) F.S. Daniş, A.T. Cemgil, "Model-Based Localization and Tracking Using Bluetooth Low-Energy Beacons," Sensors, vol. 17,11 2484, Basel, Switzerland, 2017. doi: 10.3390/s17112484.

(Denney, 2012) E. Denney, G. Pai, and J. Pohl, AdvoCATE: An Assurance Case Automation Toolset, Int'l. Conf. on Computer Safety, Reliability, and Security, 2012.

(Goodenough, 2007) J. Goodenough, H.F. Lipson, and C.B. Weinstock, "Arguing security – creating security assurance cases," available at https://www.us-cert.gov/bsi/articles/knowledge/assurance-cases/arguing-security-creating-security-assurance-cases, 2007.

(GSN, 2018) The Assurance Case Working Group, Goal Structuring Notation Community Standard Version 2, January 2018.

(Hawkins, 2015) R. Hawkins, I. Habli, D. Kolovos, R. Paige, and T. Kelly, Weaving an Assurance Case from Design: A Model-Based Approach, IEEE 16th Int'l. Symp. on High Assurance Systems Engineering, 2015.

(ISO, 2013) IT Security Techniques Technical Committee, Information technology – Security techniques – Code of practice for information security, ISO 27002, 2013.

(Jahan, 2018a) S. Jahan, A. Marshall, and R.F. Gamble, "Self-adaptation strategies to maintain security control assurance cases," Proc. of the 12th IEEE Int'l Conf. on Self-Adaptive and Self-Organizing Systems, 2018.

(Jahan, 2018b) S. Jahan, C. Walter, S. Alqahtani, R.F. Gamble, "Adaptive coordination to complete mission goals," IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W), 2018.

(Jahan, 2019a) S. Jahan, A. Marshall, and R F. Gamble, "Evaluating security assurance case adaptation," In Proceedings of the 52nd Hawaii International Conference on Systems Sciences," 2019.

(Jahan, 2019b) S. Jahan, M. Pasco, R.F. Gamble, P. McKinley, B.H.C. Cheng, "MAPE-SAC: A Framework to Dynamically Manage Security Assurance Cases," 1st International Workshop on Self-Protecting Systems (SPS 2019), pp. 146-151, Umea, Sweden, 2019.

(Jahan, 2020a) S. Jahan, I. Riley, C. Walter, R. Gamble, M. Pasco, P.K. McKinley and B.H.C. Cheng, "MAPE-K/MAPE-SAC: An Interaction Framework for Adaptive Systems with Security

Assurance Cases." Published at Future Generation Computer Systems journals, 2020. https://doi.org/10.1016/j.future.2020.03.031

(Jahan, 2020b) S. Jahan, I. Riley, C. Walter, and R. Gamble, "Experimenting with Darjeeling in the MAPE Loop Planning Phase," submitted for publication in 2020 IEEE 6th Workshop on Self-Aware Computing, 2020.

(Jensen, 2009) K. Jensen, and L.M. Kristensen, "Coloured Petri Nets: Modelling and Validation of Concurrent Systems," Springer-Verlag, 2009.

(JT Force, 2015) Joint Task Force Transformation Initiative, Security and privacy controls for Federal Information Systems and Organizations, Technical Report; National Institute of Standards and Technology, 2015.

(Kelly, 2004) T. Kelly, R.Weaver, "The goal structuring notation–a safety argument notation," In Proceedings of the dependable systems and networks 2004 workshop on assurance cases, Citeseer, 2004.

(Kephart, 2003) J.O. Kephart, and D.M. Chess, "The vision of autonomic computing," Computer, vol. 1, pp. 41–50, 2003.

(Langford, 2019a) M.A. Langford, G.A. Simon, P.K. McKinley, and B.H.C. Cheng, "Applying evolution and novelty search to enhance the resilience of autonomous systems," 14th Int'l Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 63-69, 2019.

(Langford, 2019b) M. A. Langford and B. H. C. Cheng, "Enhancing Learning-Enabled Software Systems to Address Environmental Uncertainty," 2019 IEEE International Conference on Autonomic Computing (ICAC), Umea, Sweden, pp. 115-124, 2019.

(Le, 2015) D. T. Le, "Quality Trade-offs in Self-Protecting System," 2015 IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, pp. 152-156, Cambridge, MA, 2015. doi: 10.1109/SASOW.2015.30.

(Lichtenstein, 1985) O. Lichtenstein and A. Pnueli, "Checking that finite state concurrent programs satisfy their linear specification," 12th ACM SIGACT-SIGPLAN Symp. on Principles of Prog. Languages, pp. 97–107, 1985.

(Lipson, 2008) H.F. Lipson, and C.B. Weinstock, "Evidence of assurance: Laying the foundation for a credible security case," available at https://www.us-cert.gov/bsi/articles/knowledge/assurance-cases/evidence-assurance-laying-foundation-credible-security-case, 2008.

(Marshall, 2018a) A. Marshall, S. Jahan, R.F. Gamble, "Assessing the Risk of an Adaptation using Prior Compliance Verification," In Proceedings of the 51st Hawaii International Conference on System Sciences, 2018.

(Marshall, 2018b) A. Marshall, S. Jahan, and R. F. Gamble, "Toward evaluating the impact of self-adaptation on security control certification," 13th Int'l Conf. on Soft. Eng. for Adaptive and Self-Managing Systems, pp. 149–160, 2018.

(Moyano, 2013) F. Moyano, B. Baudry, J. Lopez, "Towards Trust-Aware and Self-adaptive Systems," In: Fernández-Gago C., Martinelli F., Pearson S., Agudo I. (Eds.), Trust Management VII. IFIPTM 2013, IFIP Advances in Information and Communication Technology, vol. 401, Springer, Berlin, Heidelberg, 2013.

(NIST, 2014) NIST, Assessing Security and Privacy Controls in Federal Information Systems and Organizations", NIST Special Publication 800-53A, Revision 4, 2014.

(NIST, 2015) NIST, Assessing Security and Privacy Controls in Federal Information Systems and Organizations, NIST Special Publication 800-53 revision 4, 2015.

(Ross, 2004) R. Ross, M. Swanson, G. Stoneburner, S. Katzke, and A. Johnson, Guide for the Security Certification and Accreditation of Federal Information Systems. NIST Special Publication 800-37, 2004.

(Rushby, 2015) J. Rushby, "The interpretation and evaluation of assurance cases," Technical Report SRI-CSL-15-01, 2015.

(Simon, 2018) G.A. Simon, J.M. Moore, A.J. Clark, P.K. McKinley, "EvoROS: Integrating Evolution and the Robot Operating System," in Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO) Companion, pp. 1386–1393, 2018.

(Vromant, 2011) P. Vromant, D. Weyns, S. Malek, and J. Andersson, "On interacting control loops in self-adaptive systems," Proc. of Soft. Eng. for Adaptive and Self-Managing Systems, 2011.

(Walter, 2018a) C. Walter, I. Riley, R.F. Gamble, "Securing wearables through the creation of a personal fog," In Proceedings of the 51st Hawaii International Conference on System Sciences, 2018.

(Walter, 2018b) C. Walter, "The personal fog: An archirtecture for limiting wearable security vulnerabilties," Ph.D. Dissertation, University of Tulsa, May 2018.

(Walter, 2019) C. Walter, and R. F. Gamble, "Crossing the Adaptation Boundaries of Distinct Testbeds," 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W), 2019.

(Weyns, 2013) D. Weyns et al., "On patterns for decentralized control in self-adaptive systems," Lecture Notes in Computer Science, pp. 76–107, 2013.

(Yuan, 2014) E. Yuan, N. Esfahani, S. Malek, "A Systematic Survey of Self-Protecting Software Systems," ACM Transactions on Autonomous and Adaptive Systems (TAAS), vol.8 no.4, pp. 1-41, 2014. doi: 10.1145/2555611.

List of Acronyms

AC– Access Control

ACC – Adaptive Cruise Control

ACM – Association for Computing Machinery

AFRL – Air Force Research Labs

AI - Artificial Intelligence

AOM – Adaptation Operator Manager

C&A – Certification and Accreditation

CM – Configuration Management

CPN – Colored Petri Net

DDR – Double Data Rate

DoS – Denial-of-Service

FAC – Functional Assurance Case

GB – Gigabytes

GNSS - Global Navigation Satellite System

GPS – Global Positioning System

GPU – Graphics Processing Unit

GSN – Goal Structuring Notation

HRVM – Heart Rate Variability Monitor

IA – Identification and Authentication

IEC – International Electrotechnical Commission

IEEE – Institute of Electrical and Electronics Engineers

IMU - Inertial Measurement Unit

IR – Incident Response

ISO – International Organization for Standardization

LES - Learning-Enabled Systems

LTL – Linear Temporal Logic

MAC – Media Access Control

MAPE – Monitor-Analyze-Plan-Execute

MAPE-K – Monitor-Analyze-Plan-Execute with Knowledge

MAPE-SAC – Monitor-Analyze-Plan-Execute with Security Assurance

Cases ML – Machine Learning

MMTS – Multi-modal Traveler System

MSU – Michigan State University

NIST – National Institute of Standards and Technology

PID - Proportional Integral Derivative

PIN – Personal Identification Number

POCL – Partial-Order, Causal Link

RAM – Random Access Memory

ROS – Robot Operating System

RTK – Real-time kinematic

SAC – Security Assurance Case

SC – System and Communications Protection

SCN – Security Control Network

SDK – Software Development Kit

SI – System and Information Integrity

SIG – Soft goal Interdependency Graph

SSD – Solid State Drive

SUT – System Under Test

TB – Terabytes

TU – University of Tulsa

V&V – Verification and Validation

VANET – Vehicular ad-hoc Network

VC – Verification Concern

VFlow – Verification Workflow