The SuperVolo Story

Authors in Alphabetical Order: Caleb Lucas-Foley, Danielle Stewart, Ed Sandberg, Jim Carciofini, John Shackleton, Kevin Quick, Michal Podhradsky, Rand Whillock, Ryan Peroutka, Tyler Smith

Editors and Reviewers in Alphabetical Order: Andrew Shaughnessy, Johanna Zimmerman, Todd Carpenter

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Table of Contents

I. Abstract 5
II. Foreword: Why are we here? 5
Following Along 10
III. Introduction: Waking up in the Code 10
Organization of the FMAC Approach 19
Part 1: Framing Your Need 20
System Boundary 20
Stakeholders 21
Engineering Process Stakeholders 21
Requirements 22
Measurement 23
Conclusion 25
Part 2: Model the System 25
Describing Dependencies 26
Introducing Models 29
Creating Useful Models 31
Types of Models 35
Domain Models 35
Architectural Models 36
Domain Specific Models 40
Software Models 41
Automated Model Generation 41
Generating AADL Models 42
Generating SysML and UML Tools Models 42
Generating Design Structure Matrices (DSMs) 42
Model Validation 43
Reference Models 43
Conclusion 44
Part 3: Analyze dependencies and change impact 44

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Change Impact Analysis 47
Architectural Dependency Change Impact Analysis 48
Software Dependency Change Impact Analysis 51
Domain Separation Analysis 57
System Resource Change Impact Analysis 62
Regression Testing 63
Conclusion 64
Part 4: Contain - Designing a Change Ready Architecture 64
The Cost of Change 65
Reduce the Amount of Change Propagation on Existing Avenues 69
Method: Pick the least impact change 69
Removing Avenues of Change Propagation 70
Method: Break Dependencies with Interoperability Standards 71
Method: Code Generation 72
Method: Address Problematic Dependencies in the Architecture 73
Method: Contain Components 75
Method: Software Refactoring 75
Method: Choose Safe Programming Languages and Static Analysis 76
Detect Change Impact Propagation in Earlier Phases 76
Method: Hardware in the Loop Testing 77
Method: Architecture Centric Virtual Integration Process (ACVIP) 77
Conclusion 79
Case Study Overview 80
Part 5: Case Study - Selecting a New GPS 80
Frame: Mission Requires new GPS 81
Model: SuperVolo and GPS Design Artifacts 82
Creating AADL Models 85
Tutorial: Running Taphos 85
Portable Systems Integration Lab 89
Analyze: Implementation Options and Change Impacts 92
Software Change Impact Analysis 92

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Tutorial: Inspecting Dependencies with Taphos 98
Commentary on Caller vs Callee Functions 104
Conclusion 104
Contain: Refactor, Software Changes, and Results 104
Conclusion 105
Part 6: Case Study - Implementing a Partitioned Architecture 105
Frame: Mission Requires new Rapid Sensor Integration 105
Model: SuperVolo Design Artifacts 108
Separate Tasks into Scheduleable Processes. 109
Add Protected Shared Memory Components. 112
Assign MILS Domains to Applicable Hardware Components 114
Create End-to-End Flows and Assign CIA Values 114
Analyze: Measuring the Impact of the Design Updates 115
Time partitioning 115
Space Partitioning 118
Tutorial: Running Risk Management Framework Mixed Criticality Analysis 121
Contain: Software Changes and Results 132
Conclusion 133
Part 7: Case Study - Add Software Defined Radio to SuperVolo 133
Frame: Mission Requires Secondary Navigation System 133
Model: New Components and Candidate Integration Sites 134
Analyze: Evaluate impact of integration options 136
Software Integration Options 136
Risk Insertion and Worst Case Error Impact 136
Runtime performance impact 139
Contain: Generate and Integrate Code 142
Generating Source Code with TangramPro 142
Conclusion 144
Conclusion 144
Appendices 144
A. Acronyms 144

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

B. Bibliography 145
C. V-MESA GPS Analysis 147
Introduction 147
SuperVolo GPS Results 149
VOLTA Learner Deployment 154
GPS Learner Details 156

I. Abstract

This book describes a method for transforming a legacy cyber physical system into a modular, rapidly changeable, modern capability. We describe an approach for rigorously defining and capturing dependencies between components in a cyber physical system architecture called *Frame Model Analyze Contain (FMAC)*. This process is part of our larger *Rigorous Digital Engineering (RDE)* methodology. We show how to use those dependencies to qualify and quantify **change impact**. Change impact is the effect (planned or unplanned, direct or indirect) of a change on a system's functionality, airworthiness, or other capability.[1] We demonstrate how to use those change impact metrics to plan where and how to break dependencies and apply modular technologies. Finally, we walk through case studies that show specific applications of these methods to a representative system. The audience for this book is tech leads working on cyber physical systems. It is meant to be a guidebook and a collection of case studies and tutorials. If you are an architect responsible for maintaining a cyber physical system, then this book is for you.

II. Foreword: Why are we here?

When it comes to cyber physical systems, all change comes with risk. System owners, whether commercial aircraft certifiers or military airworthiness authorities, have to manage that risk and decide which risks are acceptable and which are not.

The current practices of software development in the Defense Industrial Base (DIB) are *not working*. DIB engineers have been doing extensive (as measured by dollars and hours) software testing for many years, but such testing is not resulting in high quality software. The GAO reported that in 2020 **23% of all software defects in the F-35 were found *after* software delivery to the test aircraft.**[2] The F-35 experiences expensive schedule delays because the

[1] The DO-178C standard uses "Change Impact" specifically in the context of *software* change impact, but we use it more broadly here.
[2] https://www.gao.gov/products/gao-21-226 full report, p35

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

current practices do not consistently catch major defects, and finding defects after deployment is more costly and time intensive to fix.

> *Delays in capability development and delivery increase the risk that capabilities will be out of date by the time they are delivered, capability development costs will be higher, and capabilities will be delivered to the fleet with deficiencies. Ultimately, this leads to warfighters waiting longer for the capabilities they need to achieve their missions.*[3]

Software capabilities are critical to modern flight systems. Circumstances often call for new or updated software. Each new capability, every new sensor, comes with new software requirements and, more importantly, software *impacts*. A given change may not cause problems in isolation, but may cause problems when *integrated* into a larger system. Cyber-physical systems like the F-35 frequently run over budget and beyond schedule – the 2024 Government Accountability Office (GAO) report indicated that software issues are a major driver of cost and schedule overruns:

> *Problems with aircraft software supporting the radar and electronic warfare systems have been especially prevalent, with some test pilots reporting that they had to reboot their entire radar and electronic warfare systems mid-flight to get them back online. Program officials stated that early versions of radar and in-flight systems software can commonly experience rebooting issues. However, even after being nearly a year delayed, TR-3 software continues to be unstable, according to test officials.*[4]

For some types of change, the impacts, risk mitigations, and approval processes are well understood. For example, FAA approval is required for any change to an aircraft that affects its exterior profile (e.g., flying with a window open or a new protruding device). For such a physical change, the change impact may be easily understood in the context of decades of flight data. The mitigations (e.g., adjusting flight ceiling) may be similarly straightforward for physical changes to an aircraft. For other changes, such as software changes to add new autonomy capabilities, the impacts, risk mitigations, and approval processes are unclear, ambiguous, or entirely absent. Assessing the safety ramifications of cyber-physical changes is a hard problem. A zero day[5] in a fielded mission system might require grounding an entire fleet until the flawed software can be patched. The United States should not allow such a repair to take months or even weeks. To stay competitive in the geopolitical landscape, we *must* be able to change software in hours.[6]

---

[3] https://www.gao.gov/assets/gao-21-226.pdf p35
[4] https://www.gao.gov/products/gao-24-106909, full report page 19
[5] A zero-day exploit is a cyberattack vector that takes advantage of an unknown or unaddressed security flaw in computer software, hardware or firmware. (https://www.ibm.com/think/topics/zero-day)
[6] For a fictional but realistic example scenario with this type of need, see *2034* by Admiral (ret.) James Elliot Ackerman and Admiral (ret.) James Stavridis USN, which describes a hypothetical conflict in which American technology is compromised.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

A 2019 Government Accountability Office (GAO) study predicted the costs of *sustaining* DoD Software at $3 billion per year.[7] The risks inherent in cyber-physical system change are driven in large part by the *relationships* that components in a system have to one another. These relationships can take many forms and exist in multiple dimensions, such as information, power, control flow, and many more. Components can depend on other components, like a software application that relies on services made available by its host operating system or a mission computer relying on its host chassis for power. We call relationships in which one component depends on another *dependencies*. Components can also have mutual dependencies, such as flight control software that relies on a Full Authority Digital Engine Control (FADEC) for information, while the FADEC simultaneously relies on flight control software for instruction. Similarly, data flows can depend on multiple components that work together to transmit information. Dependencies may also be cyclic, resulting in components that are difficult to separate from one another. Due to the complexity and chosen architectures of existing systems, these many dependencies can form a complex graph of relationships that quickly overwhelm traditional manual approaches to manage.

The result of all these dependencies is that making a change to one component within a system very often impacts that component's dependencies, with cascading effects throughout the system. We should assess the direct and indirect impacts of change. If we cannot understand the impacts of a change, then we cannot make judgments about its associated risks, and we cannot fully understand the implications of future changes. If we can understand the implications of change, then we reduce the risk of cost and schedule overruns that delay fielding of capabilities.

For this guide and demonstration, we used a SuperVolo Vertical Take Off and Landing (VTOL) Uncrewed Aerial System (UAS) as our example/case study (see Figure II.1). We chose the SuperVolo because it runs open source software and is large enough (about a 10' wingspan) to have real safety and security concerns. The SuperVolo uses a mix of Commercial Off the Shelf (COTS) and custom hardware and software, drawing on the CubeOrange and Ardupilot projects.

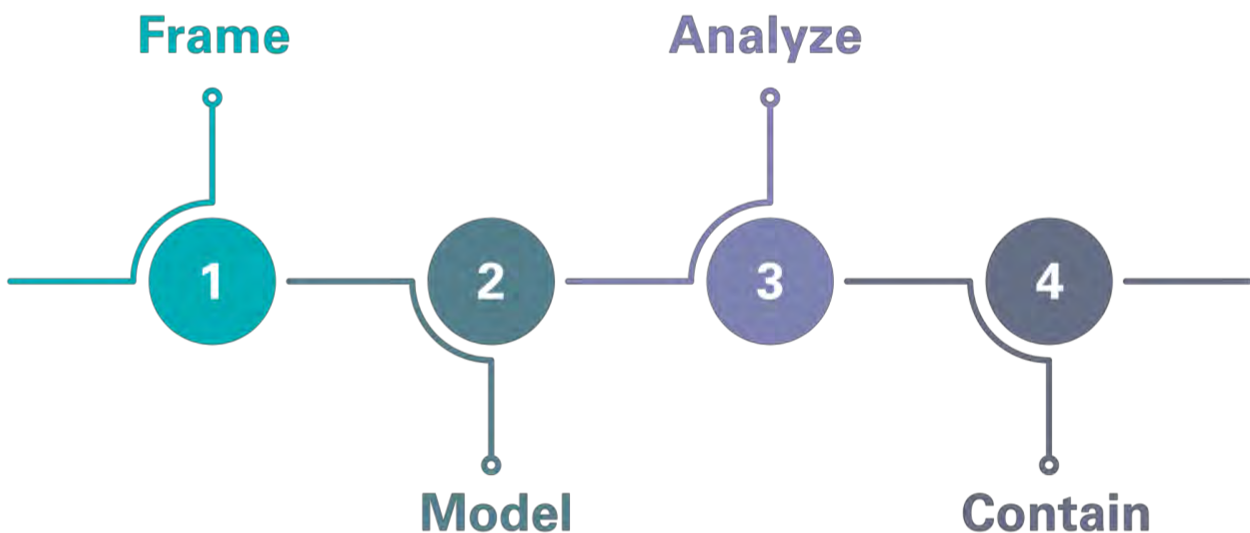
[7] https://www.gao.gov/products/gao-19-173

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Figure II.1 SuperVolo (Photo)

Like any other team tasked with maintaining a legacy system manufactured by a third-party, we started this project lacking the institutional knowledge of an Original Equipment Manufacturer (OEM). Our task was to *understand* the SuperVolo, use that understanding to make changes to increase its changeability and capabilities, and then share the story of gaining that understanding and making changes.



This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Figure II.2 The FMAC Process

This book describes the Frame, Model, Analyze, and Contain (FMAC) process we developed for changing cyber physical systems (see Figure II.2). This process is part of Galois's Rigorous Digital Engineering (RDE) methodology. The FMAC process includes **Framing** the need before selecting tools or making models; **Modeling** the relevant technology to inform decisions; **Analyzing** the models to reveal non-obvious information and generate documentation and implementation artifacts; and **Containing** sources of risk and variability to minimize the impact of changes.

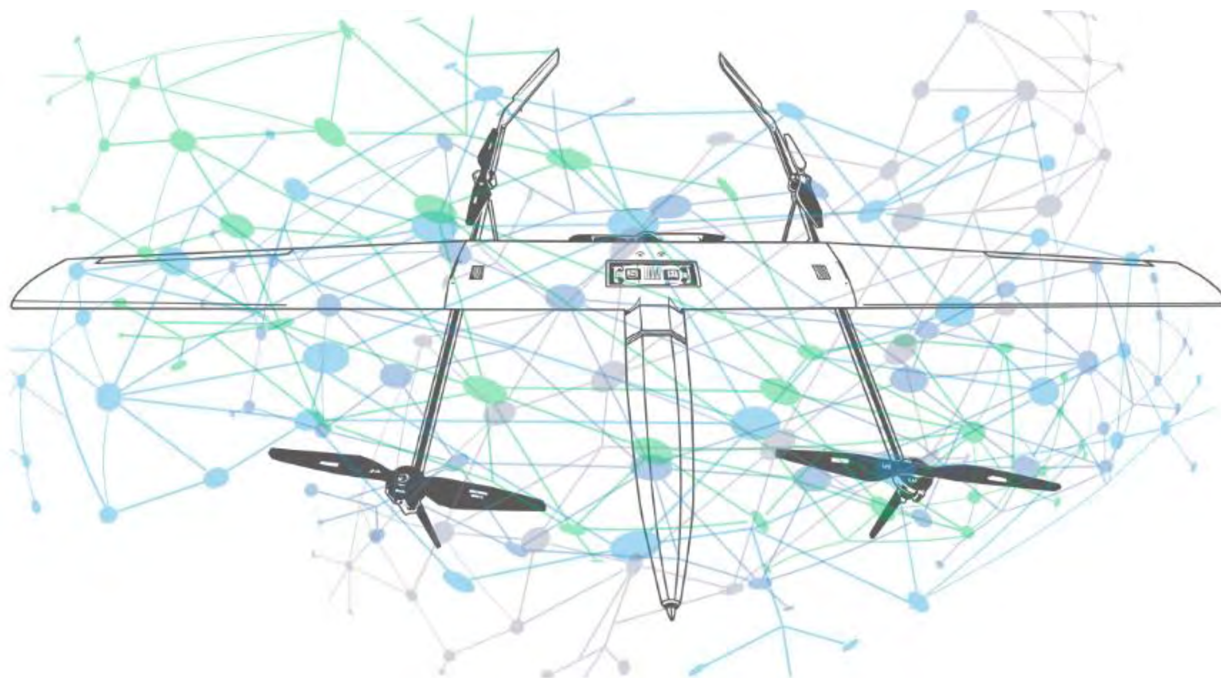


Figure II.3 Abstract illustration of components and dependencies

A cyberphysical system like the SuperVolo has many kinds of hardware and software components (colored ellipses in Figure II.3) and multiple dependencies between those components along which a change impact can propagate (colored lines in the figure above). These dependencies are not necessarily orthogonal, since a single change to a component may propagate change impact via multiple dependency types. A component may also transform a change impact from one dependency type to another. This book is about how to change a cyberphysical system architecture and how to make changes that lessen the impact of future change.

You are probably not working on a SuperVolo. You may not even be working on an aircraft. The good news is that RDE and the FMAC process are generalizable solutions. If you are currently

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

tasked with understanding, maintaining, and managing change for a complex cyber physical system, then this book is for you.

Following Along

The tools, examples, and source code used in the tutorials in this book are available to U.S. Government staff. Many are open source and available to anyone (see Table II.1).

Table II.1

Artifact	Type	Location
Ardupilot Source Code	Code	Github
LLVM Binaries for Ardupilot	Binary	Link TBD after public release
FASTAR	Tool	CAMET
MADS	Tool	CAMET
RMF	Tool	CAMET
SuperVolo Models	Model	Link TBD after public release
SysML to AADL Bridge	Too	CAMET
Taphos	Tool	Link TBD after public release

III. Introduction: Waking up in the Code

You toss and turn, clenching your eyes tight to hold on to sleep. Something hard is pressing into your shoulder.

Why is this bed so hard? You wonder to yourself. Begrudgingly, you crack open an eye and blink away the fog of sleep.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

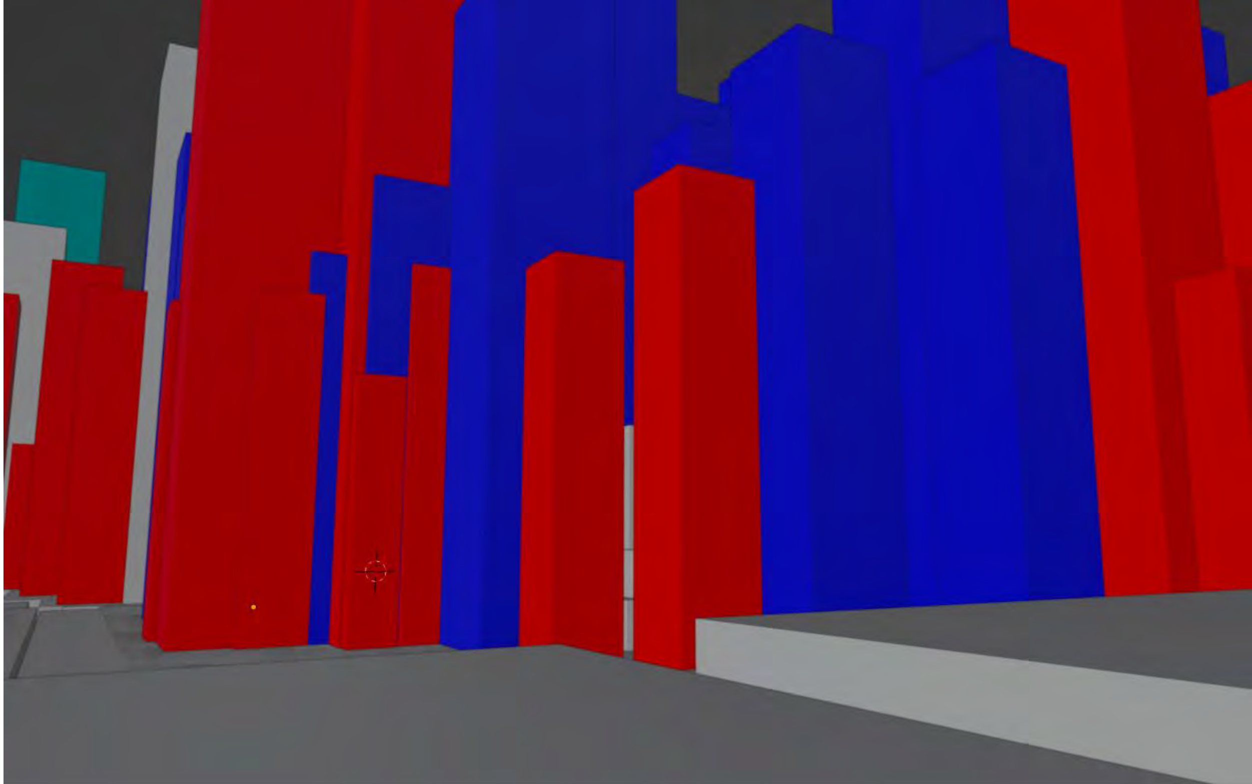


Figure III.1 A confusing view

“Where am I?” you wonder aloud. The world comes slow into focus. You are on a hard, bare surface, surrounded by what appear to be multi-colored buildings. Some are small enough you feel like you could wrap your arms around them, others seem to stretch upward toward infinity. You open your other eye, and your memory starts to coalesce as sleep fades.

I’m in the code. You think to yourself. *I opened this up last night. I needed to implement a redundant location tracking system.*

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

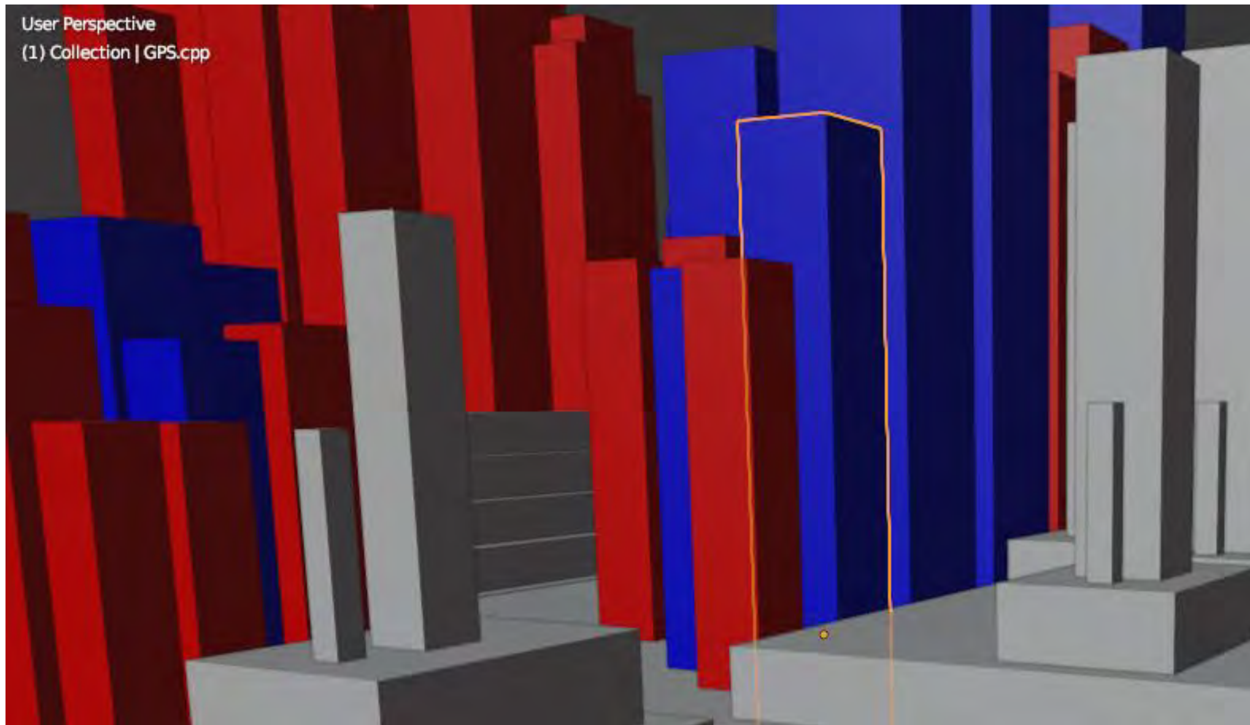


Figure III.2 Getting Oriented

More alert now, you focus on specific buildings. *Aha!* Focusing on a particular blue structure, you notice you can see more detail. Its name appears. `GPS.cpp`.

It's a C Plus Plus file.

Looking closer still, you can see its content and the content of files nearby.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

```
AP_GPS.h
44
45  class AP_GPS_Backend;
46
47  /// @class AP_GPS
48  /// GPS driver main class
49  class AP_GPS
50  {
51      friend class AP_GPS_ERB;
52      friend class AP_GPS_GSOF;
53      friend class AP_GPS_MAV;
54      friend class AP_GPS_MTK;
55      friend class AP_GPS_MTK19;
56      friend class AP_GPS_NMEA;
57      friend class AP_GPS_NOVA;
58      friend class AP_GPS_PX4;
59      friend class AP_GPS_SBF;
60      friend class AP_GPS_SBP;
61      friend class AP_GPS_SBP2;
62      friend class AP_GPS_SIRF;
63      friend class AP_GPS_UBLOX;
64      friend class AP_GPS_Backend;
65
66  public:
67      AP_GPS();
68
69      /* Do not allow copies */
70      AP_GPS(const AP_GPS &other) = delete;
71      AP_GPS &operator=(const AP_GPS&) = delete;
```

Figure III.3 A C++ file in ArduPilot

“Location tracking and GPS,” you say aloud. “Let’s do this.”

You crack your knuckles and begin to type.

“Wait!” A tinny voice shouts. “What are you doing?”

“I’m implementing location tracking,” you shout back, annoyed at the interruption.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

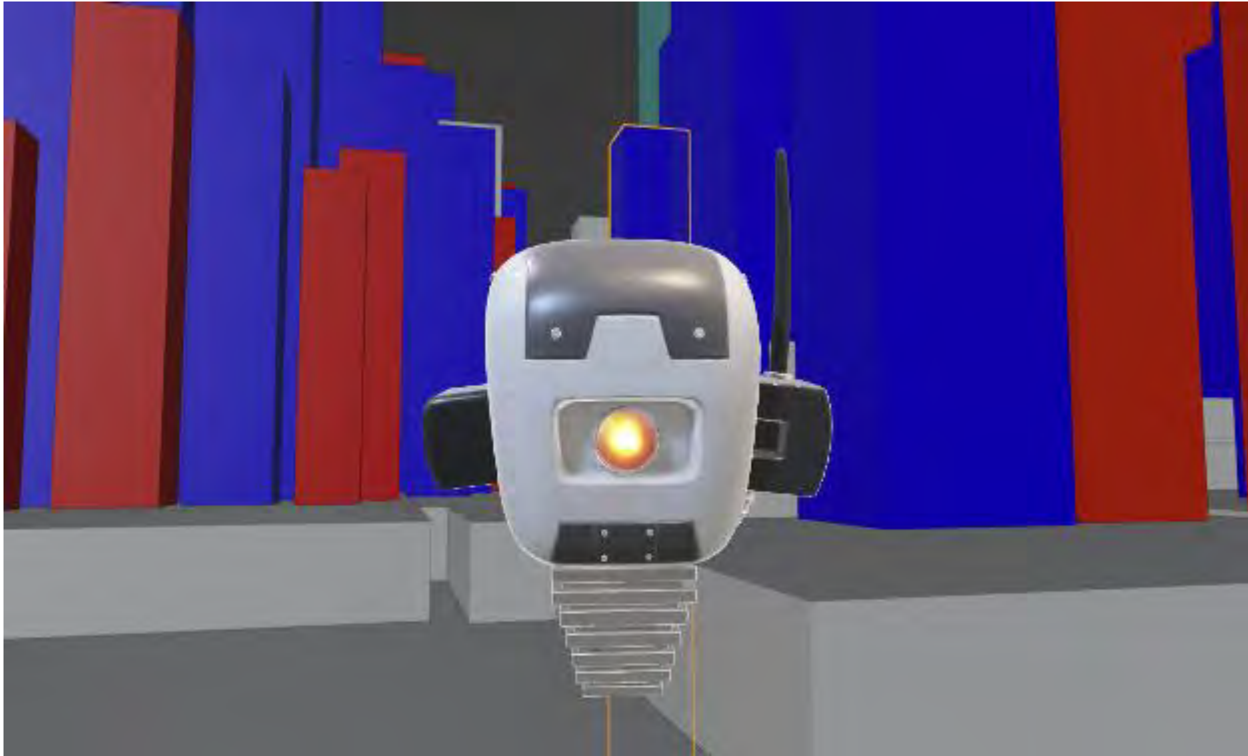


Figure III.4 FMAC, the robot

You follow the voice with your eyes and see a robot hovering in the space between two buildings. It glides over toward you. “Do you know what that code does?”

“Not really. From the name, I assume something to do with the attitude and heading reference system. I’m going to update it to use a new message set.”

“You can’t just change code!”

“If it breaks something, I’ll fix it,” you say back. *Who is this to tell me how to do my job?* “People have been writing software this way for decades. Learn by doing, you know?”

“Suppose you change this file, how will you know your change was successful?”

“The tests will pass. Or fail. Either way, I’ll know.”

“Do any other pieces of software rely on this file?”

“I don’t know. We’ll find out when we run unit tests.”

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

“And if your unit tests have gaps?”

“We’ll find out in functional testing.” You are starting to get annoyed. “Look, I have a lot of work to do. Can you get out of my way?”

“Why are you working on this file?” The robot asks, changing tactics.

“I…I just woke up here.”

“And where is ‘*here’*?”

“In the SuperVolo flight control software,” you say through gritted teeth. *How dense is this robot?*

“That is the *what*, not the *where*. Take my hand.”

The robot clearly has no hands. You are about to angrily point this out when you find yourself lifted into the air as if hoisted by an invisible harness.

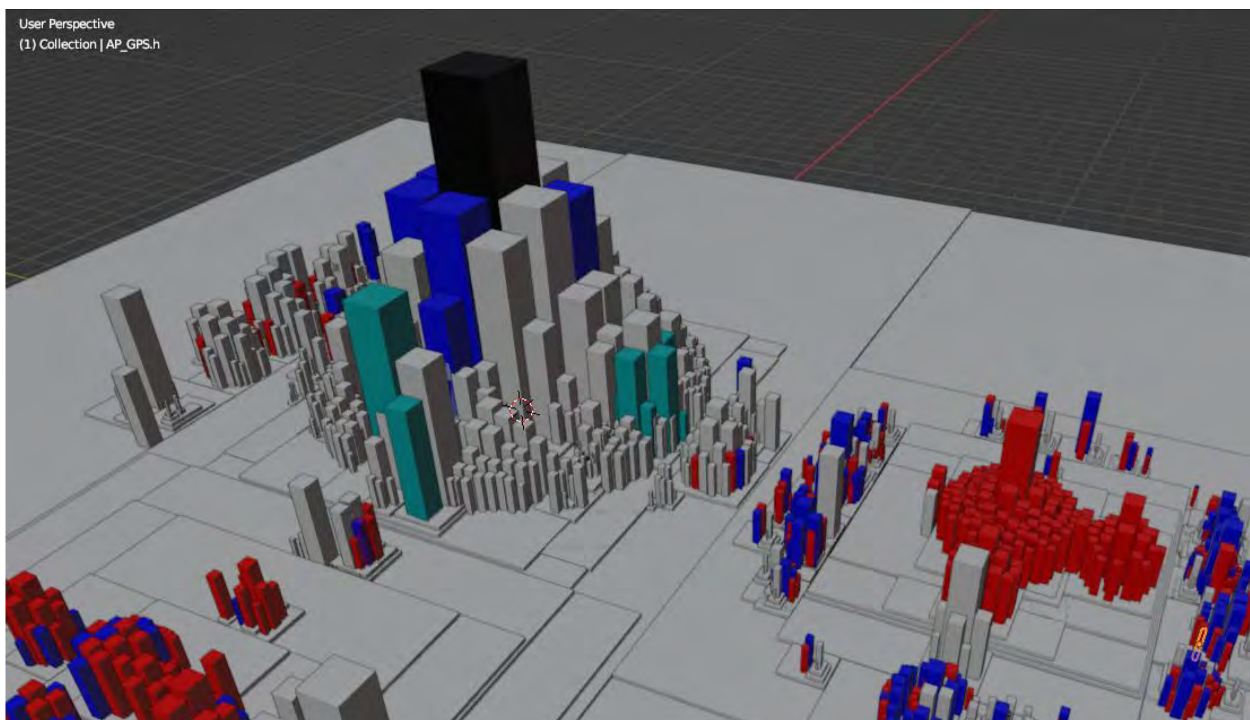


Figure III.5 Automatically generated cityscape 3D model of the Ardupilot source code. `AP_GPS.h` *is selected in the bottom right corner.*

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

A vast landscape stretches out before you. You're high in the air, high enough to see the tops of most of the buildings. You keep your eyes fixed on `AP_GPS.h`.

"My name is *FMAC*." The robot says from beside you in the air. "It is my job to help you see the whole picture. It's my job to help you *understand*."

Without warning you are flying higher and higher. The city becomes a tiny dot in the distance far below you. The gray of the pavement starts to take shape. It is a wing!

Figure III.6 3d Model of the SuperVolo in Blender, Generated with Polycam

"Here we are," FMAC says. "We're at the top."

"The top of what?"

"Of what I can show you, today at least."

"That's the SuperVolo."

This view looks familiar. You've seen the SuperVolo many times, even gotten to turn it on and experiment with it. "The code runs in the SuperVolo. Got it."

'Indeed it does. Let's look from another perspective."

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Together you descend again, racing toward the gray of the wing with reckless abandon. Abruptly you feel a sense of *sliding* and the world blurs for a moment, then snaps into focus.

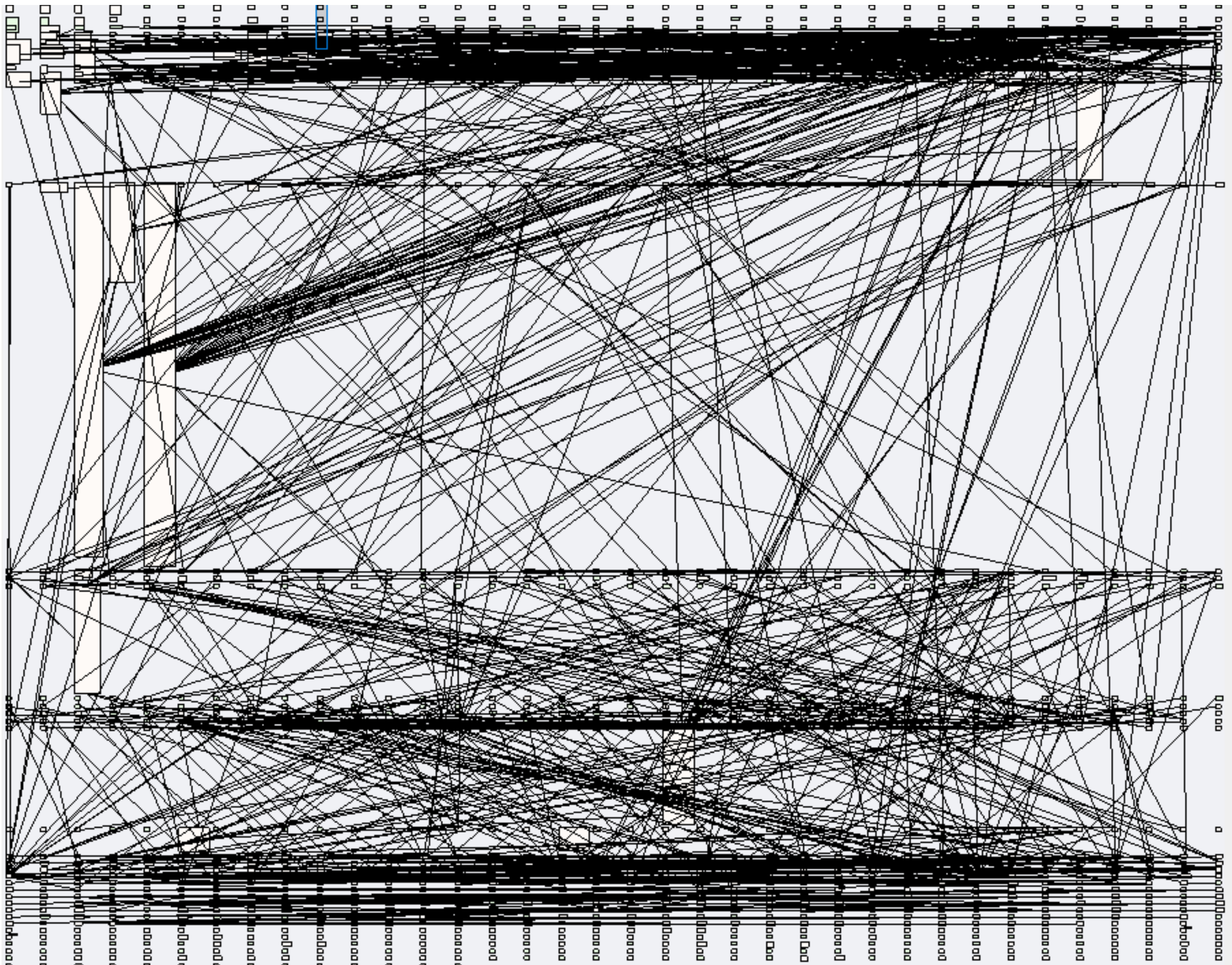


Figure III.7 Top level view of some of the components and relationships in Ardupilot, the flight control software for SuperVolo. This graph is generated by Sparx Enterprise Architect.

This time it is not a city you see, but a vast spider web of interwoven cords.

“What is this?” you ask, now genuinely curious.

“These pieces of software are interconnected. Every piece of software, every line of source code, relies on other code.” FMAC idly picks up a thread, hefting its weight. “Lots of pull here. Looks like this one is from `AP_AHRS` to `AP_GPS`.”

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

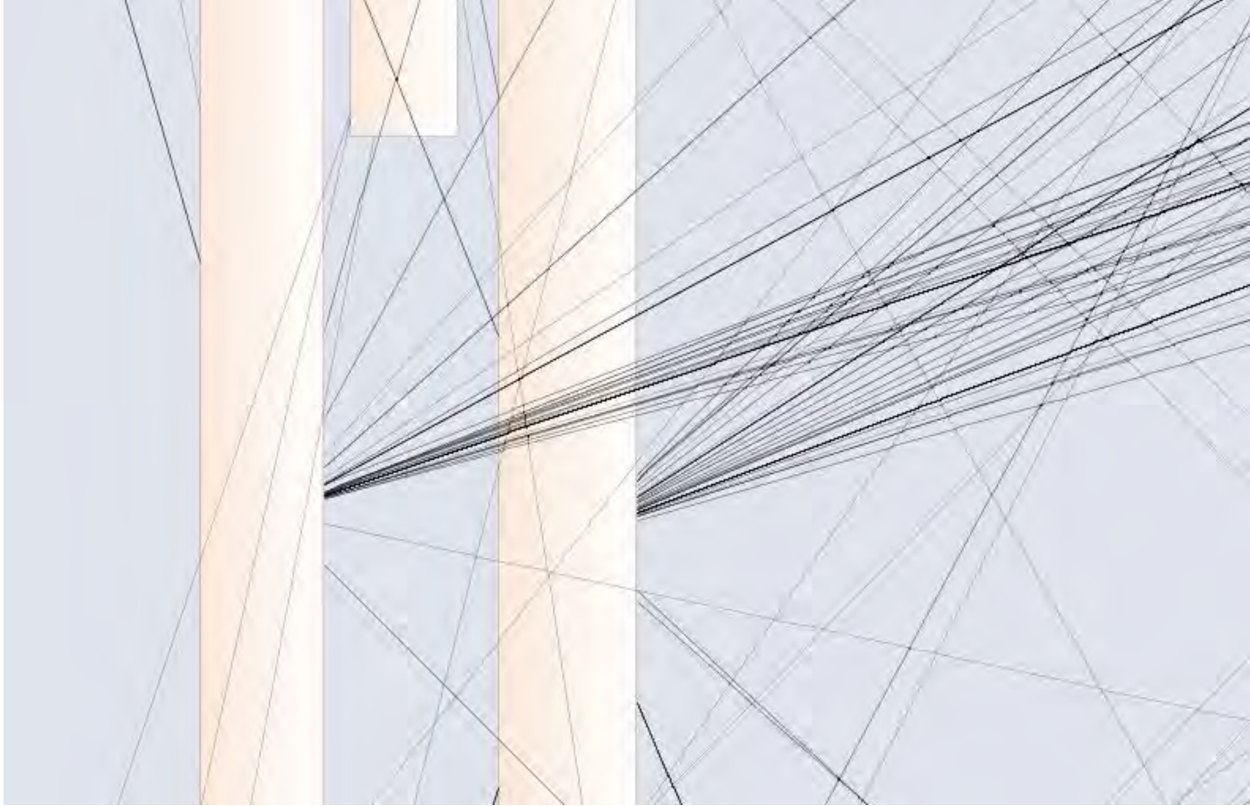


Figure III.8 Excerpt of a large UML Diagram generated by Sparx Enterprise Architect Showing Relationships Connecting Software Components

“Each of these lines is a dependency from one piece of software to another,” FMAC continues. “When you change one end of a dependency...” Here, FMAC drops the line dramatically, “you may affect the other.”

The line FMAC had been holding falls onto a mass of similar lines stretching out in all directions.

“These lines are all dependencies on `AP_GPS` FMAC starts.

“There are dozens of these,” you interject. “So if I change `AP_GPS`”

“...You might affect the behavior of each of these other components,” FMAC finished, gesturing to the pile. “Each one will need to be tested and validated.”

‘But that could take months!”

“Indeed. The interconnections between pieces of software that have not been carefully designed, architected, and maintained to be composable are difficult to find and even more

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

difficult to untangle. If you want to change a system in hours rather than months, you're going to need my help."

"Let's get started."

Organization of the FMAC Approach

At this point, we set aside the whimsical notation of FMAC as a helpful robot. However, we encourage you to keep the imagery from the introduction in mind. Cyberphysical systems are complicated; the FMAC approach can help you build and maintain such systems. Below is an outline of the topics to be covered:

- **Frame**: Determine the system boundary, stakeholder needs, and prospective changes to your system.
  - **System boundary:** identify what is inside and outside the scope of your system
  - **Identify stakeholders:** identify what people and organizations motivate your changes
  - **Identify requirements:** Define what you need to accomplish
  - **Measurement:** Define how you will measure results of changes
- **Model:** Create abstractions of your system and its components to help make decisions about changes.
  - **Dependencies:** Models formalize elements and relationships, particularly dependency relationships
  - **Artifacts:** Models of existing components should make use of existing artifacts like documentation or source code
  - **Domain models:** Create models that identify the terminology of your system
  - **Architecture models**: Create models that identify the boundaries of your system and its components
  - **Domain Specific Models**: Create models that define relationships between parts of your system
  - **Software Models:** Create models of the software components of your system
- **Analyze:** Use your models to determine the potential impacts of change
  - **Software impact risk**: Evaluate models of software components to determine the change impact on one or more software components
  - **Performance impact**: Evaluate models of hardware and software to evaluate performance impact of a change
- **Contain:** Make changes that minimize change impact
  - **Reduce Change Impact Propagation**
  - **Remove Avenues of Change Impact Propagation**
  - **Detect Change Impact Propagation Earlier**

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Part 1: Framing Your Need

A model is, by definition, an approximation of reality and will never be exact. Still, when designed properly, these approximations can be useful. You should have a clear idea of what you want to *gain* from any model before you start spending effort creating it.

All models are wrong, some models are useful. - George Box[8]

Framing is the most important step in cyber-physical systems engineering. Framing is often overlooked because it does not provide the immediate satisfaction of writing code or bending metal. We frequently learn that large projects have started creating models without a clear understanding of *why* they are making models. You must have a clear idea of how you plan to use your models before you start making them and framing will help you do this.

A model *never* perfectly characterizes all aspects of a system (by definition, a model is an abstraction).

Framing is the process of defining the context for your activity. To frame your need, you should:

- Define the boundary of your system
- Define the stakeholders
- Define the current and desired characteristics of interest for your system
- Define measurements by which you will evaluate changes to system characteristics.

System Boundary

Frame → System Boundary

Picture a cyber physical system relevant to you. Perhaps it is a car, airplane, industrial controller, or medical device. What is part of the system? What is not part of it, but related to it? The answer to this question is the system boundary. Think about how you might need to change that system to keep it operational, add a capability, or address an issue. In later chapters we'll use several additional requirements to illustrate strategies for rapidly and reliably changing cyber physical systems via case studies. For example, our first case study discusses the process of replacing the Global Positioning System (GPS) in the SuperVolo. The *System Boundary* is a description of what things are part of the system, or may be part of it. In our case, that means our system boundary includes the SuperVolo, its ground control systems, and the candidate replacement GPS units. We call the things inside of the system boundary *system elements*.

[8] https://en.wikipedia.org/wiki/All_models_are_wrong

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Stakeholders

Frame → Identify Stakeholders

The concept of a "stakeholder" is foundational to software and systems architecture. A stakeholder is an:

> *Individual or organization having a right, share, claim, or interest in a system or in its possession of characteristics that meet their needs and expectations.*[9]

Framing your needs means surveying and prioritizing the concerns of all system stakeholders. Ask questions like, "what does the system need to do that it does not currently do?" The answers will motivate your decisions on whether and how to change a cyber-physical system.

Engineering Process Stakeholders

Some stakeholders are involved in the design, development, testing, and deployment of the system. For these stakeholders, ask questions about the *processes* used for the system, such as:

- What are the steps you take today when modifying your system?
- Which parts of the process are difficult, and which are easy?
- Which parts are time-consuming, and where could automation accelerate the process?

This book is particularly concerned with parts of the engineering process that introduce or manage risk. These are often the parts of the process that address non-functional requirements such as safety or security:

- Finding and fixing defects
- Creating or updating documentation
- Creating or executing test plans
- Certification of safety
- Certification of airworthiness
- Conducting cybersecurity assessments

These parts of the process are often the least exciting and may not apply to the stakeholders' immediate capability needs, but failure to complete them can result in catastrophic costs or injuries (as well as massive sustainment costs) and even reduced deployed capabilities. A study by Carnegie Mellon Software Engineering Institute found that large avionics efforts could save

[9] https://csrc.nist.gov/glossary/term/stakeholder

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

7.8% of costs by exercising model-based methods (like those we recommend here) to discover integration defects early.[10]

Requirements

Frame → Identify Requirements

Before starting the actual system modeling itself, organizing modeling priorities to achieve the desired outcomes for the particular system design makes the process more efficient and yields better results. We begin by selecting the most important system requirements for evaluation, usually identifying a subset of requirements from a larger list that has been captured during a prior formal design phase. “Most important” in a given use case might be the requirements that are closely associated with mission success. Another factor might be the requirements that are perceived to have the highest risk, cause the greatest change, or are most unclear. This isn’t saying that the other requirements may be ignored or be discovered later on to have more significance than originally anticipated. It’s merely a way to scope effort and find a starting point.

In the broadest categories, requirements break down into either functional requirements or non-functional requirements. Functional requirements define the composition of the system and its functionality, such as the major components that comprise the system (hardware and software) as well as the list of component interfaces and operations. Functional requirements are captured in the composition of the design model, which reflect the components required in the system design and their relationship to one another.

Non-functional requirements define the attributes of the system with specific metrics on how the system operates. Non-functional requirements are captured in the design model as attributes and annotations assigned to specific components or design artifacts in the model. For some workflows, such requirements may not be firm and permanent values but rather *derived requirements* that represent budgets (e.g., power capacity) used to perform trade studies as the design becomes iteratively more refined.

Often the requirements most difficult to evaluate are derived requirements. Derived requirements include those that are immutable refinements of first order requirements. They may also represent instances of designs based on prior design decisions. From a modeling and analysis perspective, it is important to include derived requirements in the modeling process, otherwise you risk missing key design questions that may arise later in the design process.

As a convention, each requirement is a statement in plain language that describes a single item or necessity in the design. A requirement statement that contains multiple issues or metrics should be divided into separate requirement statements, if possible. Also, requirement statements formalize the words must, should, and may:

- Statements with the word *must* indicate that the requirement is not optional and must be enforced in order to complete the design successfully.
- Statements with the word *should* indicate that the requirement is highly suggested, but not completely necessary in order to complete the design.
- Statements with the word *may* indicate that the requirement is optional.

[10] https://insights.sei.cmu.edu/library/roi-analysis-of-the-system-architecture-virtual-integration-initiative/

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Details of the processes and tools to capture, organize, maintain, and evaluate requirements is outside the scope of this document.[11]

We have a requirement for this book:

> *The SuperVolo story and FMAC Process must provide a reference and guide for cyber physical systems engineers to make changes faster and more reliably.*

Engineering Process Stakeholders might have similar concerns that can motivate derived requirements for application of FMAC to the development of your system, such as:

- The FMAC Process must reduce the cost to find and fix defects when changing a cyber physical system.
- The FMAC Process must reduce the cost to create or update documentation when changing a cyber physical system.
- The FMAC Process must reduce the cost to create or execute test plans when changing a cyber physical system.
- The FMAC Process must reduce the cost to certify safety when changing a cyber physical system.
- The FMAC Process must reduce the cost to certify airworthiness when changing a cyber physical system.
- The FMAC Process story must reduce the cost to conduct cybersecurity assessments when changing a cyber physical system.

Once you have defined the boundary of your system, enumerated the concerns of its stakeholders, and prioritized those concerns, you are ready to apply FMAC to address some stakeholder concerns.

Measurement

Frame → Measurement

Once the requirements of most interests are identified, the next step is to define how each selected requirement will be verified. In other words, what is needed to confirm that a requirement has been satisfied? For functional requirements, the representation of specific components, interfaces, operations, or component relationships in the model may be enough. Analysis may be used to verify that the composition of the system in the model is correct with respect to the required functionality, and/or meets a desired level of completeness.

Non-functional requirements in a model-based engineering context can be verified through analysis. For example, consider a non-functional requirement that states, “The delay between

---

[11] Example requirements capture tool: https://www.ibm.com/docs/en/engineering-lifecycle-management-suite/doors/9.7.0

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

the time the temperature sensor updates its measurements and the enabling or disabling of the fan device must be 1 second or less." A latency or timing analysis could verify that this requirement is valid in the system design context and should move forward as the design matures. If analysis shows that the requirement is not feasible, or puts other requirements in jeopardy, then the requirement or the design should go through one or more rounds of additional refinement, evaluation, and analysis.[12]

For this step of the modeling process, the system designers define the analysis methods needed for each non-functional requirement in the list. Once the analysis method is defined, the specific tool to perform the analysis is also identified (more detail coming up in Part 3). Common types of model-based design analyses include:

- Analysis of physical properties, such as size, weight, and power (SWaP), against budgets.
- Analysis of resource utilization for processors, memories, and networks (busses) components and systems.
- Latency analysis for software components and information flows.
- Timing analyses for systems and subsystems.
- Security/safety analyses that identify that the system composition has not violated certain security or safety measures, such as partitioning and data isolation.
- Interface analyses that verify that hardware and software connected components include the correct matching attributes and data types.
- Standards-based analyses that ensure that specific components and interfaces have the proper annotations to satisfy certain design standards.
- Model checking analyses that verify that the model has a level of correctness and/or completeness to satisfy other model analyses or project specific needs.

Analyses provide qualified or quantified data about a system. Metrics define what those data mean in terms of whether or not requirements are met. System design analysis requires hard numbers to achieve results with high confidence, and analysis results are more easily evaluated when its input metrics are more concretely defined for the selected requirements.

Often the metrics are taken directly from the text of the requirement itself, such as "The system will disable the fan when the temperature sensor registers 25.0 degrees Celsius or less for at least 10 seconds." For other requirements, the metrics may be derived by aggregating the metrics of sub-components or decomposing metrics at the high system level. The units and syntax of the attributes applied to the model are typically specific to the target domain or analysis tool.

[12] There are other types of analysis not covered in this book, such as simulation or model checking for behavior analysis.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

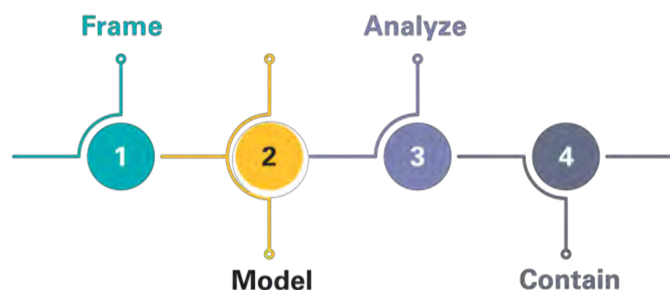
© Galois Inc. 2024, 2025

Metrics may address architectural features of the design as well. For example, a requirement that states, “Subsystem Z will only support security level 3,” will need a security analysis to evaluate whether or not the architecture will satisfy it. For additional discussion of measurements, particularly measurements related to organizational and business concerns (e.g., cost, schedule) we refer to other research, such as *Measuring Digital Engineering Effectiveness for Embedded Computing Systems*.[13]

Conclusion

Before you change a system or its development process, you should have a clear idea of your desired results and how you will measure them.. This understanding is called a *framing* of the system and is the foundation of all subsequent activities.

Part 2: Model the System



Cyber physical systems are among the most difficult to build and maintain because of the scale and diversity of their dependencies. Although pure software systems are largely restricted to software dependencies, and pure physical systems are restricted to physical dependencies, cyber physical systems, by definition, contain both types There are many dependencies of both types to consider, such as mass (how are controls laws implemented in software affected by component mass?) or inter-process communication (how does the latency of communication between two components change based on their physical proximity?).

In the introduction, we discussed the importance of understanding both *what* is in a system and *where* it is in relation to other parts of the system. In cyberphysical systems, the notion of where a component is in an architecture includes more than the physical location - it includes all of the relationships between that component and other components. By modeling a system we create a map of these relationships to help us understand where each component exists relative to others and how it might affect them.

Our examples until now have focused on the relationships between components in an architecture. However, there are relevant pieces of information beyond the physical bounds of a system such as requirements. We use the term **System Element** to describe anything with a relationship to some part of the system architecture. A component is a System Element. A requirement is a System Element.

13
https://www.researchgate.net/publication/361317212_Measuring_Digital_Engineering_Effectiveness_for_Embedded_Computing_Systems

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Describing Dependencies

Model → Dependencies

Many of the relationships that constitute the architectural “location” of a system element are dependencies. Dependencies are best described using transitive verbs. *Uses*, *includes*, *runs on*, *stored in*, are all examples of verbs that express a dependency between one component and another. When drawing dependencies in diagrams, we typically use an arrow that matches the subject-object orientation of the verb (e.g., in the diagram below, **A Uses D**).

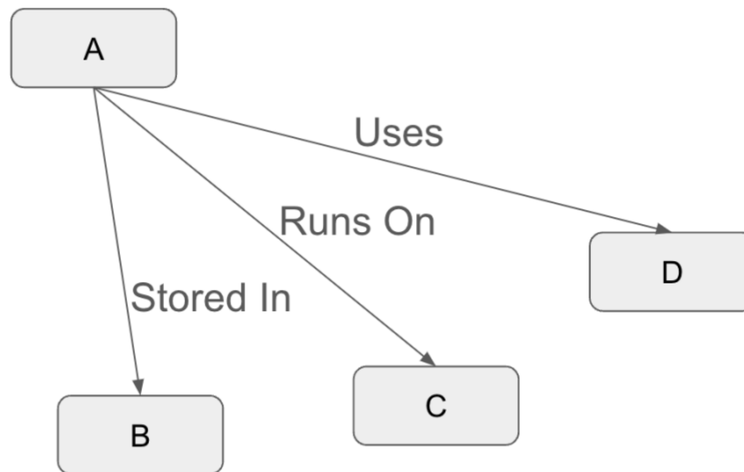


Figure 2.1 If any dependency exists from A to B, then a change to B may affect A.

Dependencies are impactful relationships between system elements. For example, a requirement to limit the weight of cables on an aircraft has a direct impact on the avionics backplane architecture design. Table 2.1 lists many of these types of dependencies.

Table 2.1: Types of Dependencies

	Dependency Category	Dependency	Example
1	Cyber	Direct use (import / call)	Explicit `#include` or `import` statement. Verb: *Foo uses Bar*.
2	Cyber	Indirect use (nested import/call)	Runtime dependency (e.g., via a shared software interface). *Verb: Foo uses IBar at compile time, Bar at runtime*.
3	Cyber	Indirect use (system state)	A software application that expects a global `location` variable to be initialized and updated. Verb: *Foo uses the*

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

			global value of location updated by Bar.
4	Cyber	Build Chain	Use of a specific compiler version or build environment (e.g., `CMake`). *Verb: Foo requires gcc*.
5	Physical	Electrical use	Use of a shared power supply between two otherwise unrelated components. Verb: *Foo uses Supply-A.*
6	Physical	Physical mounting	The form factor of a hard-point to which a sensor is attached. Verb: *Foo-Hardware is mounted at Point-1*.
7	Physical	Sensing	A hall sensor depends on the physical proximity of the sensed element. Verb: *Foo-Sensor detects the location of Sensor-H.*
8	Physical	Actuating	A physical actuator that needs an anchor point to move a control surface. Verb: *Foo-Hardware is anchored to Anchor-1.*
9	Cyber-Data	Information Flow	A software process reliant on data from a physical sensor. Verb: *Foo uses data from Sensor-A*.
10	Cyber-Data	Information Criticality	A software process reliant on information protection from a cross domain solution or isolator. Verb: *Foo is isolated from low criticality data by Isolator-R.*
11	Cyber-Behavior	Event flow, handshakes, etc	A software application that expects a heartbeat from a health monitor at a regular interval. Verb: *Foo senses a heartbeat from Bar.*
12	Stakeholder-Use	Workflow	A software application that only works when a user performs initialization steps manually. Verb: *Foo acts on user input.*
n	…		

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Counting lines of direct use software dependencies (item 1 in the table above) is one of many ways to assess the impact of a change. As discussed in the introduction, a change to `AC_PrecLand_IRLock` might affect other components, such as `Nav_EFK2_Core` which has a direct use dependency (dependency type 1) to it. What about `Nav_EFK2_Core?` A change to `Nav_EFK2_Core` might require a change to `AC_PrecLand_IRLock` , which might in turn require further downstream changes, which might in turn require further changes.

Lattix is a commercial tool that provides this dependency exploration capability for the internals of software components.[14] Figure 2.2 shows a snapshot of the types of dependencies about which Lattix can reason.

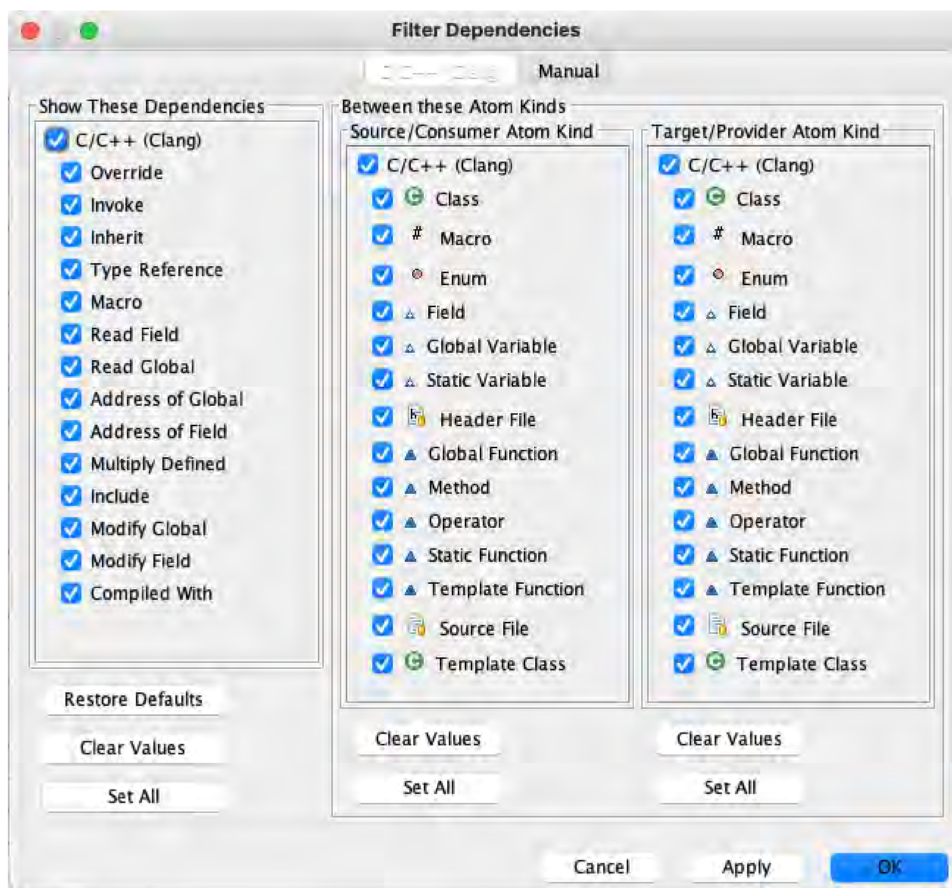


Figure 2.2 Cyber Dependencies supported by Lattix

Things get even trickier when we consider the physical system. What if the change to `AC_PrecLand_IRLock` requires replacement of a physical sensor? Will that replacement require a new physical mounting? Will it have electrical implications? What about interactions

[14] https://www.lattix.com/

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

between distributed and multi-threaded software components executing across a network on contended processing resources?

To an un-aided software engineer, this landslide of change propagation can be hard to see. Experienced systems engineers work through this challenge by gaining a deep understanding of a system over time, through experience, study, and trial and error. But even seasoned engineers need assistance. We'll talk about ways to break these chains of dependencies later in this book. For now, let's talk about how to find and describe them.

Models are a useful tool to help understand dependencies and use them to analyze change impact. Models are formal representations of a system that help to understand the structure, semantics, and dynamics of a system and communicate about those characteristics with others. Structure specifies the architecture (components, interfaces, and bindings) of a design. Semantics specifies how the individual components act and interact, such as via messaging in a particular run-time execution environment. Dynamics specify the time-varying characteristics of the system.

Taphos is a tool developed by Galois that can analyze relationships between software components from intermediate representations (IRs) like LLVM bitcode. In addition to evaluating software dependencies, Taphos can "lift" models of software that we can re-use in a variety of contexts. We used Taphos to create models of the software in the SuperVolo that we combined with manually created models to describe cyberphysical system dependencies.

By describing the dependencies between system elements in a cyberphysical system, we can better understand where they reside in relation to the remainder of the system. Once we understand where a system element resides in an architecture, we can understand how it affects, and is affected by, the system elements around it. To quantify and qualify these effects, we use models.

Introducing Models

Model → Types of Artifacts

A model is an abstraction that helps engineers understand aspects of a system. Just as there are many types of dependencies between elements in a system, there are many types of models to help convey the structure and nature of those system elements and their dependencies. In the table below we describe a variety of models and their relative "elevation," where elevation refers to the viewpoint of the model relative to the details of a system (i.e., high elevation shows lots of context and little detail, low elevation shows lots of detail but little context).[15]

[15] See Adam B V-Spells Slides (slide 8)

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Table 2.2 Types of Models

Row	Model Type	Describes	Elevation
0	Domain	Terminology and Concepts	100,000 ft
1	Requirements	System Functional and Non-Functional Requirements	40,000 ft
2	SysML	Physical System Composition	20,000 ft
3	AADL	Software and Hardware relationships, Timing, Execution Environment	10,000 ft
4	Computer Aided Design (CAD)	Physical Structure	10,000 ft
5	Simulation and Behavior (E.g., Application Data Flow)	Runtime Behavior of a System	10,000 ft
6	Data - Spatial	Scale and Relative Location of Data Elements (e.g., Software Structure)	5,000 ft
7	UML Software Models	Object-Oriented Software Source Code	1,000 ft
8	Source Code	Actual code	100 ft
9	LLVM Bitcode	VM Executable Instructions	50 ft
10	Assembly Code	Processor Executable Instructions at the Instruction Set Architecture (ISA) level	25 ft
11	Hardware	Electrical (copper, silicon, RF, etc) transmission and storage of information	0 ft
n	…		

In the introduction to this book, we saw several of these examples. You “woke up” in a 3d spatial model derived from the SuperVolo source code (#6), zoomed out to see a CAD model of the Supervolo (#4), and switched to a software dependency graph in UML (#7) to see software

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

dependencies. Each of these is a different way to describe where a system element is in an architecture.

Models are different from, but closely related to, **views**. A model is a data structure that stores information – that information and how it is stored has semantics (whether explicit or implicit semantics) that define particular meanings of each aspect of that stored information. A view is a method of presenting that data to a stakeholder. The Systems Engineering Body of Knowledge (SEBoK) defines a view as, "A representation of a system from the perspective of a viewpoint."[16] A SysML Block Definition Diagram (BDD) is a view. A Design Structure Matrix (DSM) is a view.[17]

Sometimes people make views without explicit model semantics (e.g., "PowerPoint modeling"). **When you create a view without a model, you have to trust the audience to infer the model correctly**. This can lead to confusion, as we found in our role assisting in the Army's Future Attack and Reconnaissance Aircraft (FARA).[18]

Models should contribute to understanding, communication, and application of *architecture*. The architecture of a system is a combined description of its stakeholders, stakeholder concerns, components, and how those system elements relate to one another to meet the needs of stakeholders.[19] Use modeling languages with specific semantics so that the stakeholders in Part 1, the analyses in Part 3, and the containment strategies in Part 4 all operate using the same interpretations of the information.

Creating Useful Models

A brief recap: *Where* a system element is in an architecture is defined by its relationships to other components in the architecture. A dependency is a type of relationship between two components in which one component somehow *uses* the other. When a dependency exists between two components, one can affect the other. A model is a semantically precise description of components and relationships between components in an architecture. For implementing change-ready systems, models that describe dependencies are particularly important.

Now we'll talk a bit more about how to create these models and use them to orient and measure change impact. To make a model of a system, start by listing the resources you have on hand with respect to the target system. What do you know about your system? What documentation is available? Do you have physical access? Can you test it in a lab? In the field? Do you have

[16] https://sebokwiki.org/wiki/View_(glossary)
[17] For more on DSMs, see "Design Structure Matrix Methods and Applications" by Eppinger and Browning
[18] https://cdn.prod.website-files.com/673b407e535dbf3b547179dd/676ed3550a1ef3ec00f5cbf3_Model-Based-Engineering-Support-for-Future-Attack-Reconnaissance-Aircraft-Final-Report.pdf
[19] https://sebokwiki.org/wiki/System_Architecture_Design_Definition

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

source code? LLVM Bitcode? The list in Table 2.3 describes what was available “out of the box” for the SuperVolo.

Table 2.3 Common System Artifacts

	Resource Type	What we Have for SuperVolo
1	Source Code	Ardupilot Source Code
2	LLVM Bitcode	Derivable from Ardupilot Source Code
3	Executable Bitcode	Derivable from Ardupilot Source Code
4	Documentation	User guides
5	Bill of Materials	Photos of the system internals from which we can get part numbers
6	Test Cases	Ardupilot Unit Test
7	Physical Asset	We have a SuperVolo in our lab
n	…	

As is often the case for legacy systems, the information we had on hand was not sufficient for decision making, so we created additional models. In some cases, the information may be available but in the wrong format (e.g., paper documents, Word and Excel documents that rely on natural language lacking a consistent, well-defined ontology). We recommend compiling standards-based, machine-analyzable resources such as those described in the table below. Such artifacts facilitate model-based analyses and workflows, as we’ll describe later.

Table 2.4 Models We Made for SuperVolo

	Resource Type	What we Made for SuperVolo	Notes
1	SysML Models	Hand-written SysML combined with Taphos-extracted AADL lifted to SysML with the SysML to AADL Bridge	Systems Modeling Language (SysML) is standardized by the Object Management Group (OMG).
2	AADL Models	Taphos-Extracted AADL from Ardupilot LLVM Bitcode	Architecture Analysis and Design Language (AADL) is standardized in SAE International AS5506D
3	Software Dependency Matrix	Taphos-extracted dependency matrix from Ardupilot LLVM Bitcode	Software dependency matrix is based on DSMs.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

	Resource Type	What we Made for SuperVolo	Notes
4	Information Dependency Matrix	MADS-derived information dependency matrix based on the SysML Models	

Most legacy DoD systems were built with prior airworthiness artifacts (full requirements documented, requirements traceability, proof of V&V, meeting notes, change board actions, etc). So, this list of resource types could be much larger for your system.

Taken together, all of these artifacts (both those given to us and those we derived) provide the basis to build a representation of the SuperVolo architecture. We use that architecture to understand the SuperVolo, quantify and qualify the impacts of change, and to decide our engineering plan.

Modeling for change-ready cyberphysical systems is about describing *where* each component resides in an architecture, which we describe in terms of each component's relationships to others. Start from the highest level (the most "zoomed out") perspective: a domain model (described in Table 2.2) that describes the most significant 10 to 20 system elements in your architecture. For the SuperVolo we did this by creating a SysML Block Definition Diagram (BDD) describing the major components of the SuperVolo.

Based on the objectives identified from stakeholder input in Part 1, select which lower level modeling language(s) are likely to meet stakeholder needs. For example, if you anticipate making physical structure changes to your system, a CAD model may be appropriate. For the SuperVolo we planned to make software changes, so we created AADL models. Using the semantics appropriate for each selected modeling language, model the components of your architecture and the relationships between those components and others in the architecture.

The task of modeling (whether by authoring models or translating non-model artifacts into models) and *relating* all of these artifacts to one another is sometimes called *creating a digital thread*. We'll touch on the concepts and approach of a digital thread in this book, but will not provide a comprehensive guide. Product Lifecycle Management (PLM) tools like Siemens TeamCenter or Dassault 3DS™ provide capabilities for tracking and relating elements of a digital thread. Tools and standards like OSLC and [Intercax Syndeia] can help connect disparate elements in a digital thread.

Relationships between system elements can be expressed at different levels of fidelity as a system design progresses. For example, in an initial low fidelity system description the relationship between two components may be modeled in AADL as a `connection` with an empty `feature group`. This is equivalent to a block box description that says that there is an

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

information sharing relationship between the two components but no details are expressed about the data exchanged. Subsequent model updates can add details to the connections, expressing data directions (in or out) and specific data formats. Since most of the analysis tools support mixed fidelity (i.e., different components can be at different fidelity levels), the level of design details currently defined for a subsystem or component should not prohibit it from inclusion in a system model.

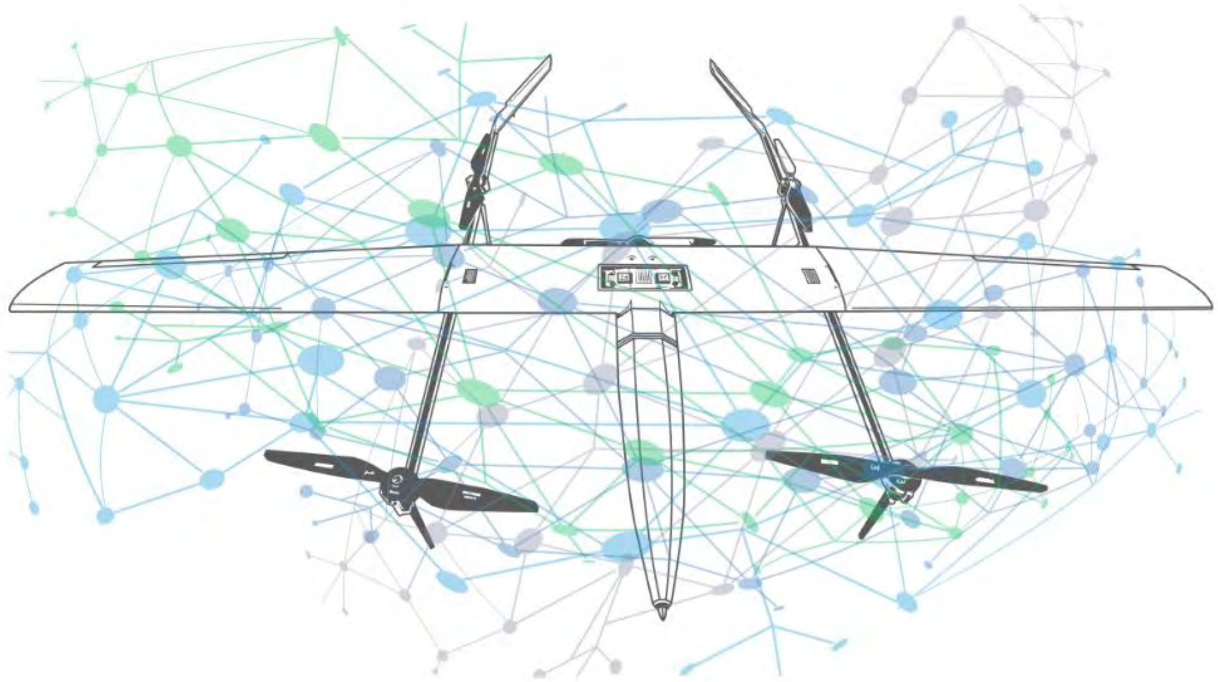


Figure 2.2 Abstract representation of SuperVolo elements and connections between those elements. A pathway that traverses these components and connections is a digital thread.

Overall system model details often include:

- Enumeration of requirements and the dependencies between requirements and other modeling artifacts.
- Structural representation of components (hardware and software), interfaces, and connections (physical and logical).
- The binding of software components to hardware components such as busses or processors. For example, the software that embodies a particular operation or function is hosted on a specific processor.
- Information flows that begin at one software component and end at another software component.
- Specification of execution semantics, such as operating systems on processors and scheduling mechanisms on busses.
- The metrics defined in previous steps applied to appropriate components.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

- Additional annotations that further address requirements, such as standards compliance or requirements traceability.

As shown in Figure 2.2, we connect these modeled information sources together to form a complete picture of the system. We can traverse this modeled representation for a variety of purposes, which we'll discuss more in Part 3.

Types of Models

Domain Models

Model → Domain Models

A type of dependency we have not yet discussed, but have extensively used, is dependency on language and terminology. Up to this point we have *used* language extensively (this is a book, after all!)

Rigorous Digital Engineering (RDE) starts with domain models because we need a design lexicon to effectively collaborate with one another. For any of the models discussed in this section to be useful, they need to be rooted in a common *domain* of understanding.

A domain model is therefore something that defines the nouns (roughly speaking the endurants) and verbs (roughly speaking the perdurants) – and their combination – of a language spoken and used in writing by the practitioners of the domain. Not an instantiation of nouns, verbs and their combination, but all possible and sensible instantiations

Dines Bjørner & Yang ShaoFa[20]

To create a domain model, start by writing out a glossary of terms applicable to your system. A variety of languages are available for formalizing this glossary so that you can use (and reuse) terms across other artifacts.

You can define a domain model with general-purpose architecture modeling tools like SysML 1.x, or with domain specific languages like Lando.[21]

---

[20] https://www.imm.dtu.dk/~dibj/2024/primer/primer.pdf

[21] You can find an extensive domain model written in Lando in this report: https://www.nrc.gov/docs/ML2232/ML22326A307.pdf

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Architectural Models

Model → Architectural Models

Architectural models describe the components of a system and how they relate to each other and to the system's stakeholders. For this project, we used SysML 1.x to describe the SuperVolo architecture. SysML.1 models are one of the best places to get oriented in a cyberphysical system because they define the context and boundaries of the system and its environment. SysML is standardized by the OMG. There are a variety of tools available for creating, viewing, and editing SysML models. For this guide, we used Dassalt's MagicDraw and Sparx's Enterprise Architect. Free tools, such as the Eclipse-based Papyrus, are also available. For a full introduction to SysML modeling, we recommend Lenny Delligatti's book *SysML Distilled*. For the purposes of this book, we introduce the core SysML concepts and set of SysML diagrams that we use most for describing the SuperVolo.

A SysML *model* is a *database* containing *elements* and *relationships* between those elements. Many of these relationships are the **dependency relationships** that are the focus of this book. A SysML *diagram* is a particular view of the contents of that database; a SysML diagram is not "the model," rather, the diagram is just a visual representation of some of the model. There are a variety of *types* of elements that you can use when creating a SysML model. For this book, we'll primarily use *blocks*, *parts, ports*, *connections*, and *interactions*.

Table 2.5 SysML Views (Partial List)

View Name	Description	Use for SuperVolo
Use Case Diagram	Describes a systems boundary and its stakeholders.	Communicate intended use of the system.
Block Definition Diagram (BDD)	Describes categories of things and the relationships they have to each other.	Communicate component definitions and boundaries.
Internal Block Diagram (IBD)	Describes the internal and external interconnections of a system, subsystem or component.	Communicate component composition.
Sequence Diagram	Describes a series of information propagations through and architecture.	Communicate information flow or control flow between components.

There are other SysML views (e.g., activity diagram, state diagram, parametric diagram) defined in the SysML standard. For information on these, see the references mentioned earlier.

For our SuperVolo work, we started by creating SysML models to flesh out our understanding of the functionality of the SuperVolo and its notional missions. Several of our engineers were

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

trained on SuperVolo operation and were able to inform this discussion. For example, below are two SysML diagrams describing 1) use cases for the SuperVolo and 2) component structure of the SuperVolo. The former establishes *stakeholders* and *use cases* for the SuperVolo. It shows a system boundary (dotted lines) and various uses of the system by stakeholders. These uses are yet another form of dependency - a Sensor Operator depends on the “Collect Data" function to accomplish his or her job.

Figure 2.3 SysML Use Case Diagram

The component structure diagram describes the physical composition of the SuperVolo. Black lines with solid black diamonds indicate *composition* relationships. The nature of dependency represented by a composition relationship varies, but at least one dependency exists between the parent and child components (e.g., the SuperVolo Airframe is part of the SuperVolo Aircraft Platform, thus the airframe has a physical dependency on the platform, because it must reside within the platform).

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Figure 2.4 SysML Block Definition Diagram

We can add more detail to these relationships in a variety of ways. For example, we can use AADL relationship stereotypes to associate components with one another and provide information about the type of dependency (and thus potential change propagation avenues) represented by the relationship. Inside the SuperVolo Aircraft Platform block, we can see that there is a part called PixelHawk of type Cube Orange.

In [Part 6](#), we describe using models of the Cube Orange to specify time and space partitions for its computing resources. For example, the CubeOrange internal block diagram below shows the Actual Processor Binding relationship between an AHRS partition and the main processor on the Cube Orange board. This dependency means that a change to the (physical) main processor could affect the (software) AHRS Partition. Examples of such changes may include performance configurations (e.g., clock rate), or binding additional partitions to the processor.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

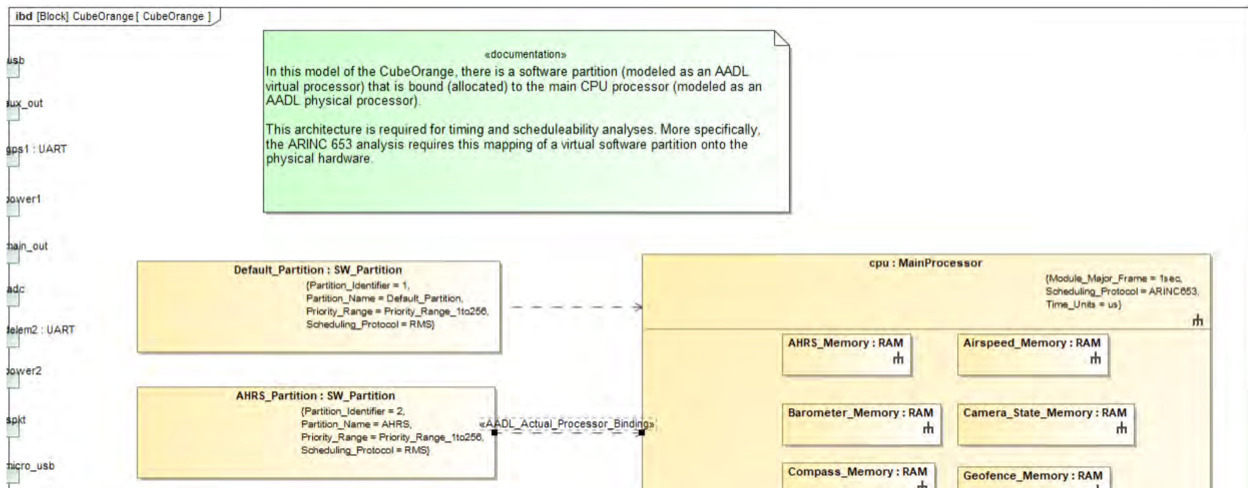


Figure 2.5 SysML Internal Block Diagram Example

Could a change to the `AHRS partition` affect the main `processor`? It depends! In [Part 4](#) we'll talk about separation architectures that reduce the risk of such propagations.

What we have defined so far lets us express relationships across several layers and representations of the SuperVolo architecture. We can use **requirement matrices** in SysML or tools like Intercax Syndeia to store and reason about relationships that cross layers of the architecture and models. Relationships that cross levels of the architecture (such as relationships between a component design and its requirement) are often used for **traceability** to meet certification requirements.[22]

A **design structure matrix** is a method of concisely describing the relationships between a collection of components in an architecture. The "elevation" of a design structure matrix depends on the level of detail in the selection of components that make up the rows and columns of the matrix (that is, a design structure matrix can describe software structure at a low level, or component relationships at a high level).

[22] For example, the ARMY MILITARY AIRWORTHINESS CERTIFICATION CRITERIA (AMACC) calls for a Component Verification Matrix (CVM) relating components to their method of airworthiness verification.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Domain Specific Models

Model → Domain Specific Models

Domain specific models describe specific facets of a system architecture in a *domain specific* manner to facilitate analysis or automation.

The **Architecture Analysis and Design Language (AADL)** is a domain specific language for describing cyber physical models with consistent terminology and semantics.[23] Unlike SysML 1.x, the AADL is purpose-built to facilitate automated analysis of an architecture for emergent properties, such as the worst case latency of an information flow. In later chapters, we will discuss several tools for such analysis, and how we use them in our evaluation and development strategy for the SuperVolo.

There are several available tools for authoring or viewing AADL models. Carnegie Mellon University Software Engineering Institute (CMU SEI)'s Open Source AADL Tool Environment (OSATE) is an open source option. Galois's Curated Access to Model-based Engineering Tools (CAMET) Base Pack is proprietary (but free to the U.S. Government) and is built on OSATE. STOOD, from Ellidiss, is another option. Collins Aerospace and its collaborators developed a set of AADL tools called BriefCASE.[24] Galois's CAMET SysML to AADL Bridge provides a plugin for authoring AADL Models in Dassault Cameo/MagicDraw.

The canonical definition of an AADL model is a textual representation written in the AADL (unlike SysML 1.x, in which models are typically stored as a database). As with SysML 1.x, AADL allows definition of a variety of kinds of elements. For this project, we used many of them, including `systems`, `components`, `ports`, `connections`, and `flows`.

You can author AADL models directly in AADL text, via an AADL graphical editor, or by annotating a SysML 1.x model with AADL *stereotypes*. For this project, we used the latter approach, as we already had a SysML 1.x model created to establish system context. You can see those stereotypes in the SysML models shown earlier in this section, such as `<<AADL_System>>`. We used both methods when working with the SuperVolo.

As with SysML 1.x and UML, you can create multiple views of an AADL model, which the AADL calls *diagrams*. For this project, we primarily used AADL structural diagrams that show the composition and connectivity of components in an architecture. AADL structural diagrams show content similar to that of a SysML IBD.

---

[23] AADL is standardized by SAE International. For an introduction to the AADL, we recommend Julien Delange's book, *AADL in Practice*.

[24] https://loonwerks.com/projects/case.html

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

In the future, we plan to use SysML 2.x heavily, as it will combine many of the capabilities and features of SysML 1.x and AADL.

Software Models

Model → Software Models

Software models describe software and its use. The Unified Modeling Language (UML) is a domain specific modeling language for describing software, specifically object oriented software.[25][26] For the SuperVolo, we used SysML to describe the overall system (including both physical and software components). We use UML to describe some pure-software components. For this project, we used Sparx Enterprise Architect for UML.[27]

Whereas SysML 1.x deals in generic terms like blocks, which can represent physical or logical elements of an architecture, UML is intended for describing object oriented software concepts such as `classes`, `interfaces`, and `associations`. As UML concepts are closely associated with software, there are many tools that can generate UML diagrams by parsing source code. We used Sparx Enterprise Architect for this purpose (that is how we created the spider web diagram in the introduction).

UML also excels at modeling data structures. Many of the standards we'll discuss later use UML to document data structures for interoperability. We use UML *views* like class diagrams to describe data structures specific to software, and to express relationships between software components, such as usage relationships between software components.

Automated Model Generation

Model → Model Generation

We can often create models from existing artifacts. This is useful because generating a model reduces the risk of human error (which is itself a source of change impact propagation!) and reduces the effort required to create a useful model. When we refer to "generating" (rather than "writing" or "creating") we mean automated generation.

[25] Martin Fowler's UML Distilled is a good place to start.
[26] UML is standardized by the Object Management Group (OMG).
[27] It may surprise you to learn that SysML 1.x is based on UML. In fact SysML is a derived extension of UML (that describes object oriented software) for the purpose of capturing a system model.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Generating AADL Models

AADL Models are commonly used for analysis of software and hardware, and can be created either by hand or through automated tools, For example, Galois's Taphos tool can create a variety of types of models (e.g., **AADL** and **Design Structure Matrix**) by analyzing LLVM bitcode. Galois's SysML to AADL Bridge can generate AADL Models from **SysML Models**. We use Taphos and the SysML to AADL Bridge extensively in the case studies in Part 5, Part 6, and Part 7.

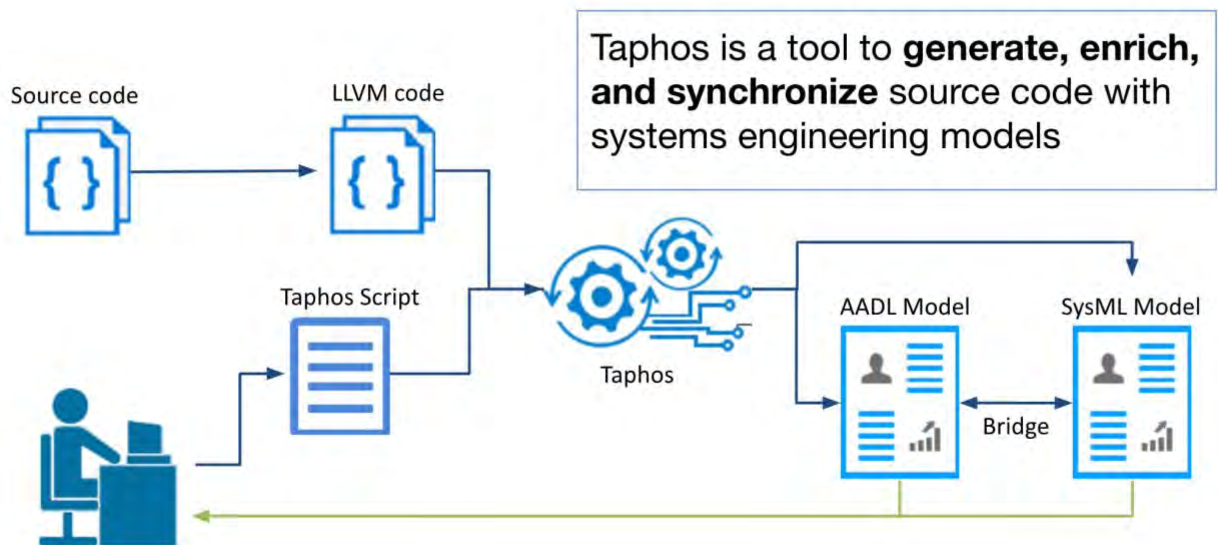


Figure 2.6 Taphos Workflow

Generating SysML and UML Tools Models

SysML and UML tools like *Dassault Cameo* or *Sparx Enterprise Architect* often provide tools for generating models (typically **Model → UML Models**) from source code. The UML diagrams shown in the Introduction were generated using Sparx Enterprise Architect. Galois's SysML to AADL Bridge can also generate **Model → AADL Models** from **Model → SysML Models**.

Generating Design Structure Matrices (DSMs)

DSMs can show a variety of different types of components and relationships, and there are a variety of tools for generating DSMs with different types of content. In this book, we use Cameo SysML Matrices to generate DSMs of mixed hardware and software components described in **Model → SysML Models**. We use Lattix and Taphos to generate DSMs for software.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Model Validation

Model → Model Validation

Models should accurately represent the system and should be both internally and externally consistent (that is, a model should not contradict itself and should not contradict the system it describes).

There are a variety of methods for verifying and validating models, including:

- For **Domain Models**, Lando provides a grammar and syntax consistency checker.
- For **Requirements Models**, NASA's FRET tool provides an environment for formalizing requirements and checking them for internal consistency.[28] Galois's FRIGATE provides a similar capability for formalizing mission plans and checking them for internal consistency.[29]
- For **CAD Models**, a variety of commercial tools are available for size and weight testing, strength testing, and more.
- For **SysML 1.x Models** and **Model UML Models**, tools like Dassault's Cameo support semantic checking via Object Constraint Language (OCL) and analysis with simulation tools or parametric diagrams. Galois's SysML to AADL Bridge also allows analysis via conversion to AADL (discussed more in Part 3).
- For **AADL Models**, CMU SEI's OSATE provides AADL syntax and semantic validation. Galois's CAMET Library provides a variety of analysis tools for performance and security (discussed more in Part 3). Collins' Resolute tool provides support for assurance cases[30], and Collins' AGREE supports assume-guarantee reasoning for designs in AADL models.
- **Design Structure Matrices** and **Spatial Data Model** are generally generated directly from source material and do not need validation.

Reference Models

Model → Reference Models

Most software engineers start by copying and pasting a "Hello, World!" program. Doing so alleviates the need for engineers to memorize boilerplate code and allows them to focus on their immediate need. The same approach works for modeling system architecture - by starting from a simple, well understood example, you can quickly get to the important parts of modeling. In the same vein, high level languages, software frameworks, and application ecosystems allow developers to reuse existing functionality and avoid re-creating common capabilities (for

[28] https://software.nasa.gov/software/ARC-18066-1
[29] https://www.galois.com/project/frigate
[30] https://loonwerks.com/tools/resolute.html

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

example, software application developers should use existing libraries for user authentication, rather than implementing such features themselves).

Reference Architectures, such as the Government Avionics Reference Architecture (GARA), provide a mechanism to reduce or mitigate human and organizational change impact. Reference architectures can reduce the effort required to create or identify model-based assets, such as those defined in this section, and can provide guidelines for consistency across projects and systems.[31] A reference model provides a reference point for describing part of a system architecture, such as by specifying desired attributes for domain-specific systems to support late analysis (as described in Part 3). A reference model can be any type of model, such as a domain model. A reference architecture provides some combination of stakeholder needs and a description of components that meet those needs.

A standard reference model is a model for which a notion of *conformance* is defined, meaning that it is possible to quantitatively or qualitatively assess whether a system model *conforms* to a reference model (that is, the reference model is *standardized*). A standard reference architecture is an architecture for which a notion of conformance is defined, meaning that it is possible to quantitatively or qualitatively measure whether a system architecture conforms to a reference architecture. As we'll discuss in Part 4 (**Contain → Remove Avenues of Change Impact Propagation**), using a reference model can reduce or remove avenues of change impact propagation.

Conclusion

In this section, we discussed approaches and tools for building models of a system that include the information required to address stakeholder needs (as identified in Part 1). We discussed modeling standards and how different modeling techniques allow us to represent components and the relationships between them. Next, we can conduct *analysis* to learn non-obvious things about our system using these models, particularly how a change impact propagates between components in our architecture.

Part 3: Analyze dependencies and change impact

So far we have described how to frame the engineering activities by defining the system boundary and engaging stakeholders and how to collect information about the system into models. In this section, we'll talk about how we use that information.

Analyze → Propagation

[31] https://dodcio.defense.gov/Portals/0/Documents/Ref_Archi_Description_Final_v1_18Jun10.pdf

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

In the summer of 2024, a defect in a CrowdStrike software update disabled airlines, hospitals, and bank systems around the world.[32][33] These systems all had direct or indirect dependencies on the changed software, and the flaw in the software change, which caused a kernel panic, propagated to many of the dependent systems, stranding people in airports and shutting down operations across industries.

When we analyze dependencies to determine change impact, it is useful to think in terms of *highly contagious* defects. Highly contagious defects are errors in a component that spread to other components by any possible means (imagine an illness spreading). Dependency relationships are the only way errors propagate; if two components are unrelated, then one cannot affect the other. The 2024 Crowdstrike error was an example of such a defect.

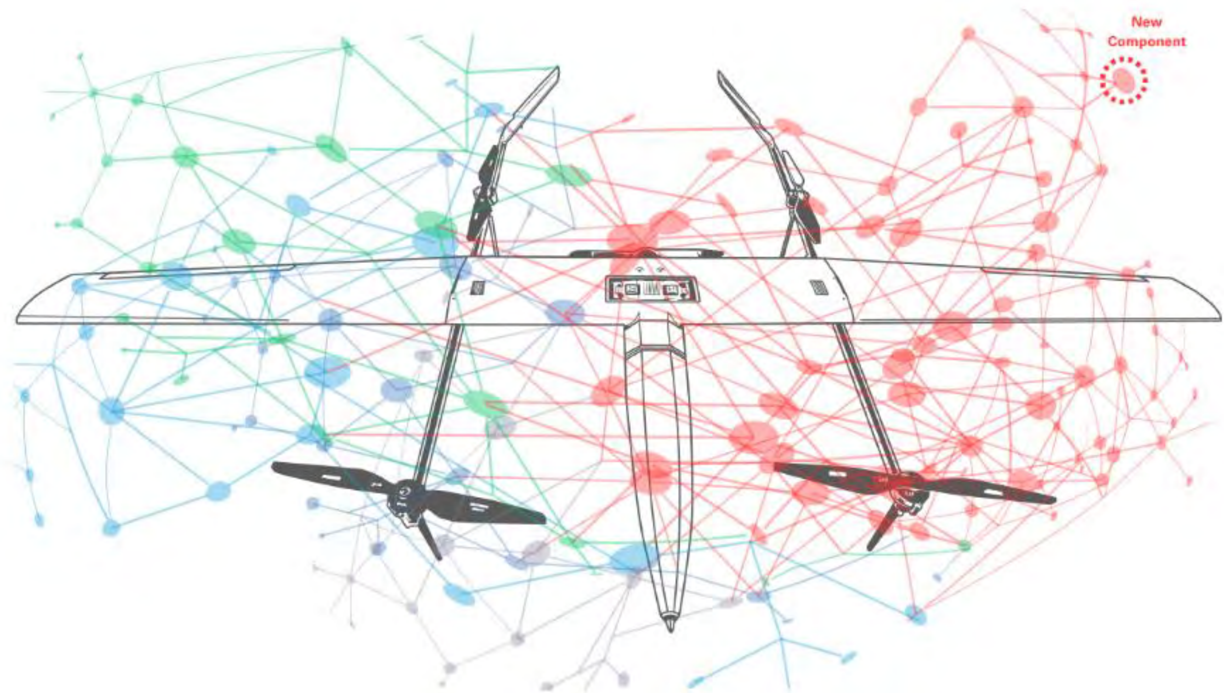


Figure 3.1 An abstract visualization of the change impact of a highly contagious defect propagating throughout an architecture.

Highly contagious defects can happen by accident, as with Crowdstrike, or on purpose, as with malicious software.[34]

Thus, when we consider a component change for a safety or security critical cyber-physical system, we ask, "**what is the absolutely worst possible impact this change could have if it**

[32] https://krebsonsecurity.com/2024/07/global-microsoft-meltdown-tied-to-bad-crowstrike-update/
[33] https://www.crowdstrike.com/falcon-content-update-remediation-and-guidance-hub/
[34] https://www.wired.com/story/mirai-botnet-minecraft-scam-brought-down-the-internet/

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

contains a highly contagious defect?" We can measure this by considering each dependency relationship to be a mechanism for propagating the contagion of the highly contagious change. We call these mechanisms ***avenues of change impact propagation***.

Propagation means some effect of a change in one component manifests in another. Propagation can follow any type of dependency relationship unless mitigated by some kind of *isolation* enforcement. Change impact is propagation of effects of a change from one component to others. Change impact analysis is the evaluation of such possible effects. In this section, we'll discuss how to use the models we created earlier to conduct change impact analysis and make decisions about our system.

Under-defined change impact is a major source of cyber physical system challenges. We hypothesize that it is a significant contributor to the cost and schedule overruns discussed in the introduction. Contemporary cyber-physical systems rely on experienced systems engineers who have an internal understanding of the system architecture and its various dependencies gained through years of experience. However, such humans are a limited resource and system complexity is reaching a point where no individual person can understand the entire system. Sustainment becomes challenging when those experienced engineers retire or are no longer part of the sustainment mission. If we want to make changes fast and make them well, then we need two things:

1. An architecture abstraction that enables rapid and thorough understanding of change impact, e.g., by allowing automated analysis to identify the impacts. (the models from Part 2).
2. An architecture that minimizes change impact (the focus of Part 4).

There are as many methods of analyzing models as there are for specifying models (likely more!). Generally, we consider any automated approach for revealing non-obvious information about a system to be "analysis." No analysis tools can tell you information that is not somehow present in your model, although it may be able to indicate the absence of information necessary to answer a query. Analysis results will only be as good as the information you put into your model, hence the importance of model verification and validation, as discussed above.

In this section we introduce several analysis methods central to the aims of FMAC, all of which are part of **change impact analysis:**

- Architectural dependency change impact analysis
- Software dependency change impact analysis
- Domain separation analysis
- Regression testing and analysis

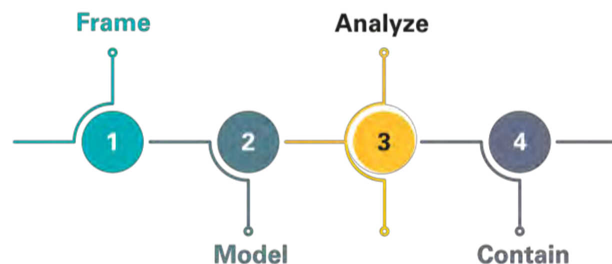
This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Change impact analysis helps us evaluate our system architecture to make decisions about how best to meet stakeholder needs. Change impact analysis helps us compare and contrast alternative approaches to implementing a change in terms of how that change affects the system. Armed with change impact analysis results, we can make rapid, informed decisions.

Change Impact Analysis

Analyze → Change Impact Analysis



With the inter-component dependencies in our system specified in one or more models, we can now assess the impact of a change. For example, in the case of the SuperVolo we know there is a physical dependency between the SuperVolo platform and the CubeOrange processor, and that there is a processor binding relationship between the CubeOrange processor and the AHRS task.

AHRS Task → CubeOrange → SuperVolo Platform

If we want to make a change to the SuperVolo platform, we need to determine its impact on related components. For example, if we opted to change the physical structure of the SuperVolo platform such that the CubeOrange board no longer physically fits, we might have to use a different computing system. That, in turn, might mean we have to modify the AHRS task to run on the new computing system. This simple propagation of effects is an example of *change impact*. With the models we created in the previous step and automated analysis tools, we can simulate a desired change and receive alerts or visualizations of these effects. Using this information, we can address potential issues before they become costly problems

The process for assessing the impact of change is not new. For example, the Army Military Airworthiness Certification Criteria (AMACC) requires aircraft configuration changes to be assessed for impacts to system airworthiness.[35] DO-178C provides a software-centric definition of Change Impact Analysis. We recommend a broader, cyber-physical definition that includes both software and hardware change impacts.[36]

---

[35] AMACC Rev B section 4.4 says, "Once airworthiness has been determined for the aircraft configuration, that configuration must be brought under configuration control to preclude unauthorized changes. Any subsequent changes to the configuration must be assessed for impacts to the system airworthiness."
[36] https://www.rtca.org/do-178/

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Table 3.1 Change Impact Terminology

Term	Definition
System Element	Any part, component, design artifact, or other asset associated with a system.
Change	Any modification to a System Element.
Change Impact	Any effect of a change.
Change Impact Analysis	Manual or automated evaluation of the effect of a change on system elements.
Change Impact Propagation	The effects of a change to one system element on other system elements.
Change Impact Propagation Avenue	A relationship between two system elements through which change impact can propagate (a way one system element can affect another). **Dependencies** are one common type of change impact propagation avenue.

Architectural Dependency Change Impact Analysis

Analyze → Architectural Dependency

One way to explore change impact is through a *Design Structure Matrix (DSM)*. SysML tools (like Cameo) include the capability to automatically generate such matrices by evaluating relationships between components.

The matrix below shows dependencies between elements of the SuperVolo SysML architecture. We can read this matrix from row to column, meaning the row “Ardupilot_AHRS” depends on several columns, including “Camera Mount Up” and “Read Airspeed.”

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

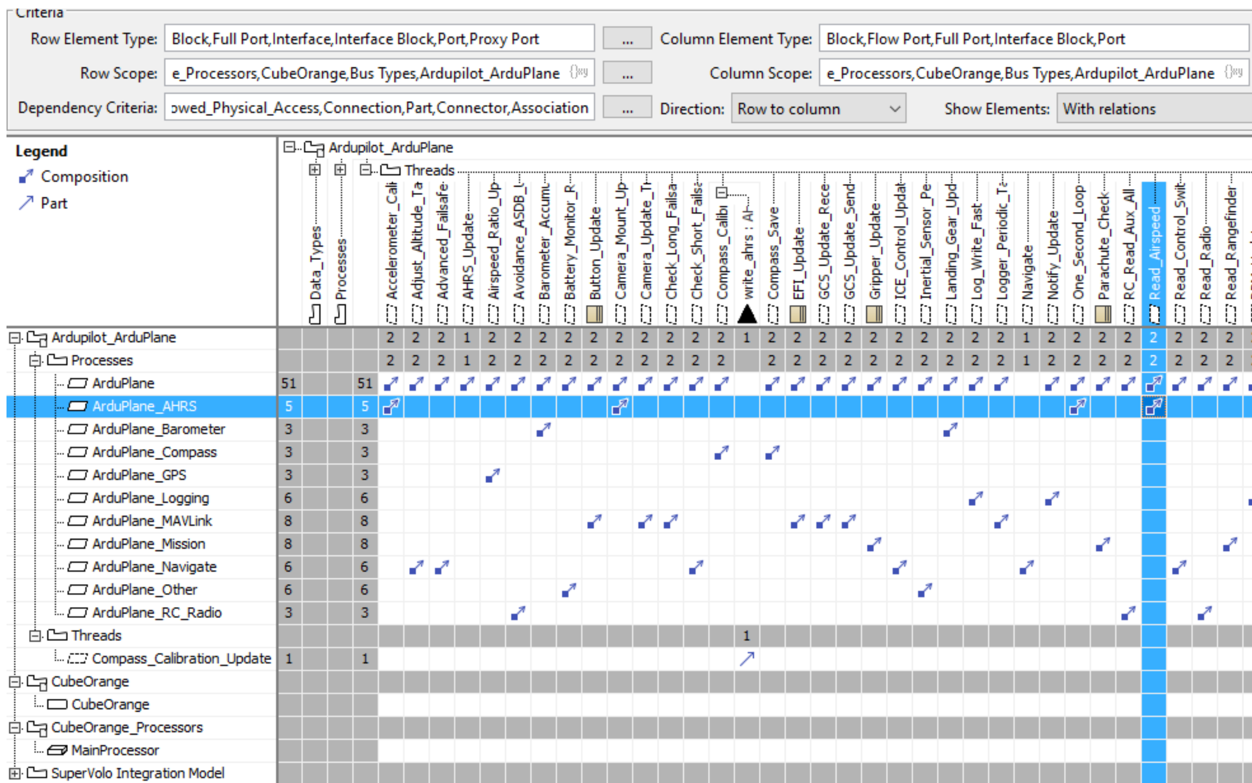


Figure 3.2 Dependency Matrix in MagicDraw/Cameo showing components used by Arduplane_AHRS

We can get a simple count of the number of dependencies by looking at the cardinality on the left - Arduplane_AHRS has 5 dependencies on other components (see Figure 3.2). ArduPlane, by contrast, has 51.

What if we want to invert the picture and ask, “what depends on this component?” To do so, we read the matrix from column to row. For example, in the highlighted column we read, “The thread Read_Airspeed is used by two processes, Arduplane and Arduplane_AHRS (see Figure 3.3).” If we want to see the nature of the relationship, we can hover over an individual cell.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

	Landing_Gear_Upd… (cut off)	Log_Write_Fast	Logger_Periodic_T…	Navigate	Notify_Update	One_Second_Loop	Parachute_Check	RC_Read_Aux_All	Read_Airspeed	Read_Control_Swit…	Read_Radio	Read_Rangefinder	RPM_Update	Scheduler_Update_	Set_Servos	Stabilize
	[illegible]	2	2	1	2	2	2	2	2	2	2	2	2	2	2	2
	[illegible]	2	2	1	2	2	2	2	2	2	2	2	2	2	2	2

Composition **from** ArduPlane_AHRS **to** Read_Airspeed

Figure 3.3 Dependency Matrix in MagicDraw/Cameo Showing Usages of Read_Airspeed

We can read this chart recursively to build a test and accreditation plan for any particular component we change. For example, if we change the thread "Read_Airspeed" we know we need to test ArduPlane_AHRS and ArduPlane. But we can't stop there. We also have to test anything that depends on ArduPlane_AHRS and ArduPlane!

Legend
- Composition
- Part

	Ardupilot_ArduPlane	Data_Types	Processes	ArduPlane_AHRS	ArduPlane_Barome	ArduPlane_Compas	ArduPlane_GPS	ArduPlane_Logging	ArduPlane_MAVLink	ArduPlane_Mission	ArduPlane_Navigat	ArduPlane_Other	ArduPlane_RC_Rac
Ardupilot_ArduPlane													
Processes													
Threads													
Compass_Calibration_Update	1												
CubeOrange													
CubeOrange													
CubeOrange_Processors													
MainProcessor													
SuperVolo Integration Model				1	1	1	1	1	1	1	1	1	1
Components				1	1	1	1	1	1	1	1	1	1
Flight Control Software Secure	10		10										
Operational Environment													
SuperVolo Aircraft Platform													
SW_Protected_Space													

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Figure 3.4 Dependency Matrix in MagicDraw/Cameo showing Components used by Flight Control Software

The only downstream dependency from these two is the Flight Control Software (Secure), which includes ArduPlane_AHRS (see Figure 3.5), and is then used by the SuperVolo Aircraft Platform.

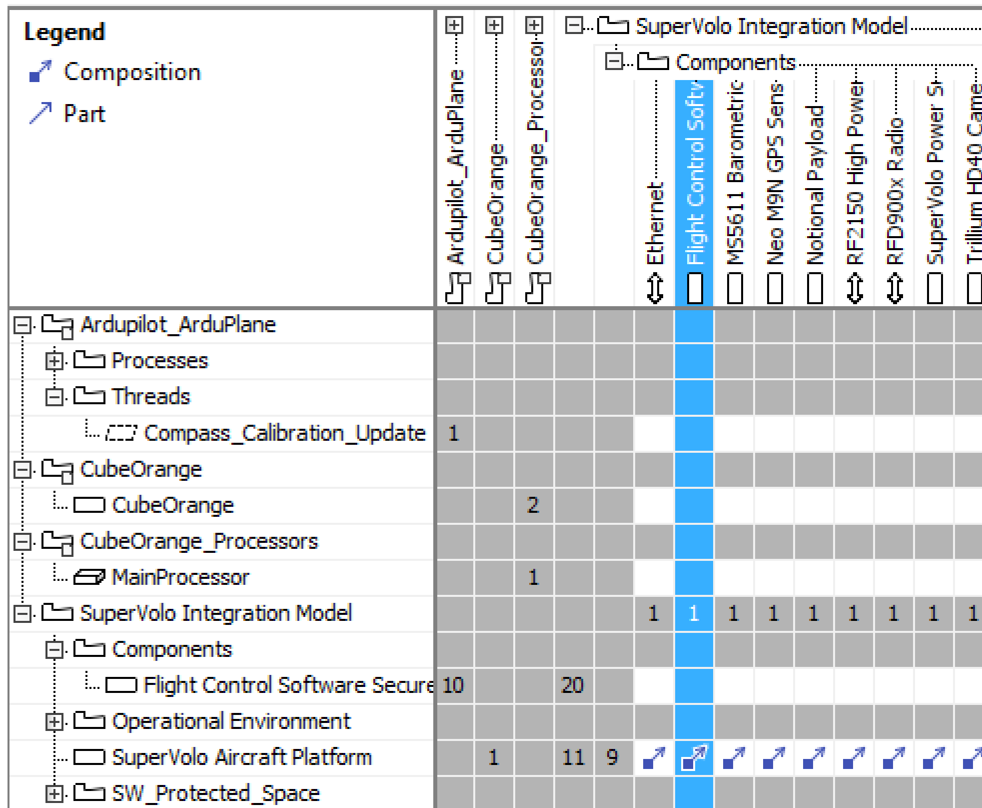


Figure 3.5 Dependency Matrix in MagicDraw/Cameo Showing Usage of Flight Control Software

All together, this means we can use the system structure as defined in our SysML model to scope the impact of changing `Read_Airspeed` to:

- Read_Airspeed (itself)
- ArduPlane_AHRS
- ArduPlane
- SuperVolo Flight Control Software
- SuperVolo

Software Dependency Change Impact Analysis

Analyze → Software Dependency

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

There are many relationships that are not directly obvious from a top-down assessment of an architecture. Even relationships extracted by parsing source code may not be complete. The Taphos tool enables us to evaluate dependencies in compiled code by *lifting* a model of the system from its bitcode.[37]

Taphos can consider both direct and indirect dependencies between software components, where a software component is a function or a class. A direct dependency (could also be called *explicit*) is an `#include` statement, `import` statement, or similar piece of code or configuration in which one software component explicitly depends on another.

An indirect dependency is a build/execution time association, such as the use of a global variable that is written by another component. For example, if a navigation thread uses a global location variable, and that location variable is updated by a gps thread, then there is an indirect (or *implicit*) dependency from navigation to gps

The dependency matrix in Figure 3.6 (generated by Taphos) shows several such dependencies between functions.

[37] Taphos reads an LLVM bitcode file to extract the LLVM Intermediate Representation (IR), and it then analyzes the LLVM IR.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

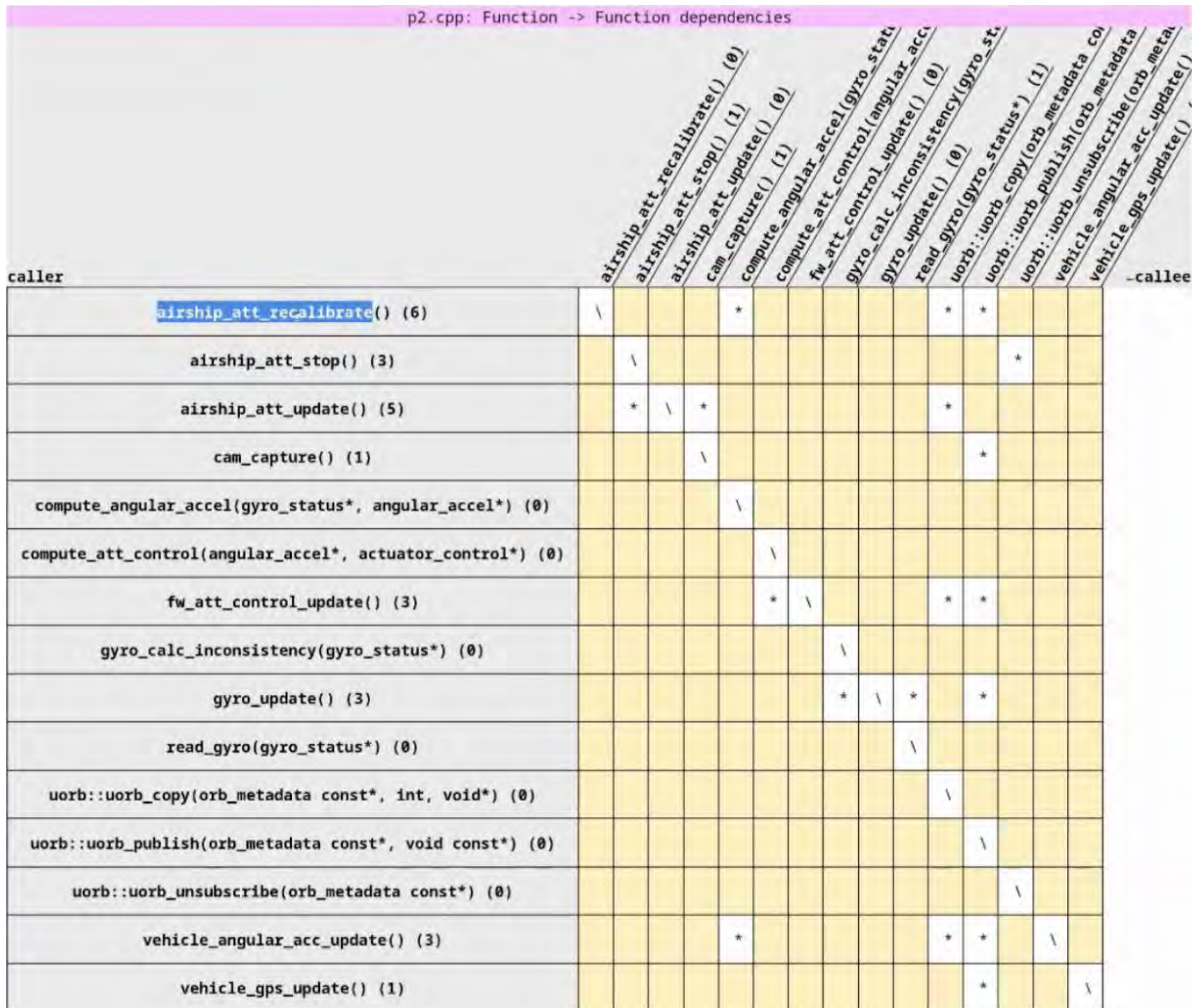


Figure 3.6 Taphos Dependency Graph Output

Structure analysis tools like Lattix can be helpful both in understanding an architecture as it exists presently, and in evaluating approaches for improving it. For example, Figure 3.7 is a design structure screenshot from Lattix showing dependency relationships between elements of the Ardupilot code base.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

$root		1	2	3	4	5	6	7	8	9
		APMrover2	AntennaTracker	ArduCopter	ArduPlane	ArduSub	Tools	benchmarks	libraries	tests
APMrover2	1	3%							1	
AntennaTracker	2		1%							
ArduCopter	3			5%						
ArduPlane	4	1			4%					
ArduSub	5					3%				
Tools	6						2%		1	
benchmarks	7							.1%	3	
libraries	8	351	172	714	492	308	181		82%	
tests	9								12	.1%

arduplilot / ...ects.DFAMS.git.ardupilot

Figure 3.7 Dependency Matrix in Lattix

Here we can see that the majority of the implementation for Ardupilot is in the `libraries` directory, which is *used by* `Arducopter`, `Arduplane`, etc. We can see at a glance that no dependencies exist *between* `Arducopter` and `Arduplane`, as those cells are empty.

Automating Software Dependency Evaluation

Analyze → Software Dependency → Call Radius

The dependency evaluation we walked through in this section was verbose for the sake of clarity. Tools like Taphos provide automated capabilities for recursively exploring a dependency matrix to identify dependent components nested at various levels of depth – one of those is the function "Call Radius".

The fidelity of a naive SysML-based DSM is limited by the information in the model: We can only assess impacts when we have information available about the dependencies between the affected components. Some dependencies, like the containment dependency we used to build this test list, are obvious. Other dependencies, such as information flows between software threads in the same memory space, are less clear. For example, what if the `Read_Airspeed`

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

uses a shared variable in memory that is also used by `Read_Rangefinder`? In that case, one could affect the other. Such a relationship is not apparent from evaluating composition alone, and presents a significant risk.

To address this risk, we can explore more subtle software dependencies using Taphos. For example, using the Taphos `call-radius` function we can see which functions call `AP_Airspeed::get_airspeed() const` (hereafter just `get_airspeed`) up to a radius of 2. Using a .json file `Llvm_Link.json` exported from Taphos, we use the script `taphos_json_info.py` with the `radius` command and target the `get_airspeed` function, as shown in Listing 3.1.

Listing 3.1 Taphos Call Radius Tool for get_airspeed with Radius 2

```
docker run -it -v $(pwd):/example -w /example taphos:0.7.1.0 taphos_json_info.py  Llvm_Link.json radius
"AP_Airspeed::get_airspeed() const" 2
Call Radius (2) for AP_Airspeed::get_airspeed() const
    Called from layer 2 :
        AP_AHRS_DCM::update          airspeed_estimate           AP_Hott_Telem::loop         generate_LTM
        update_GPS_10Hz
        check_sensor_failures                          passthrough_wfq_adaptive_scheduler
        NavEKF2_core::SelectTasFusion                  NavEKF2_core::controlFilterModes
        NavEKF3_core::controlFilterModes               Plane::verify_command
        set_servos_controlled

        ^--- 12 layer2 callers
  ↓
    Called from layer 1 :
        drift_correction             estimate_wind               calc_velandyaw              send_EAM
        send_Sframe                  verify_vtol_takeoff
        AP_AHRS::airspeed_estimate                     AP_AHRS_DCM::airspeed_estimate
        check_sensor_ahrs_wind_max_failures            NavEKF2_core::readAirSpdData
        NavEKF2_core::detectFlight                     NavEKF3_core::detectFlight
        airspeed_ratio_update                          GCS_MAVLINK::vfr_hud_airspeed
        GCS_MAVLINK_Plane::vfr_hud_airspeed            Plane::suppress_throttle

        ^--- 16 layer1 callers
  ↓
    ====> AP_Airspeed::get_airspeed() const ====>
 ↓
    Calls to layer 1 :


        AP_Airspeed::get_airspeed(unsigned char) const
        ^--- 1 layer1 callee
  ↓
    Calls to layer 2 :

        ^--- 0 layer2 callees
```

In this output, we see that 16 functions call `get_airspeed` directly and a further 12 functions call those 16. We can also see that `get_airspeed` calls only one function (an overloaded version of `get_airspeed`) which does not make any function calls to other elements of Ardupilot.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

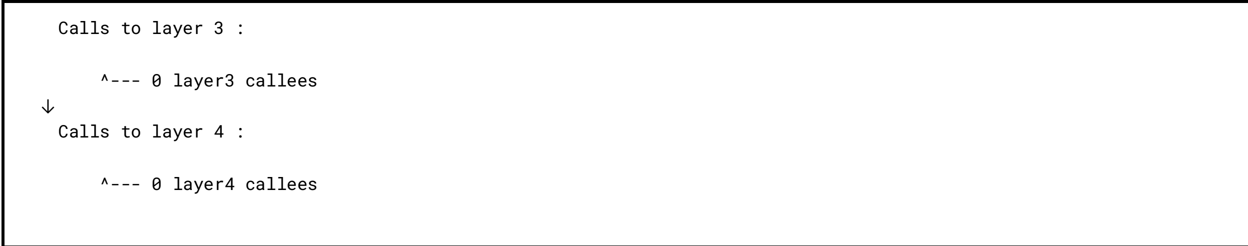
If we want to grow our search, then we can increase the radius parameter provided to Taphos, as shown in Listing 3.2.

Listing 3.2 Taphos Call Radius Tool for get_airspeed with Radius 4

```
docker run -it -v $(pwd):/example -w /example taphos:0.7.1.0 taphos_json_info.py  Llvm_Link.json radius
"AP_Airspeed::get_airspeed() const" 4
Call Radius (4) for AP_Airspeed::get_airspeed() const
    Called from layer 4 :
        AP_Frsky_Telem::loop        get_telem_data            update_logging2           Plane::stabilize
        control_run                 read_airspeed
        AP_AHRS_NavEKF::update                        NavEKF2::UpdateFilter
        NavEKF3::UpdateFilter

        ^--- 9 layer4 callers
   ↓
    Called from layer 3 :
        update_DCM                  AP_Airspeed::update       _get_telem_data           AP_LTM_Telem::tick
        get_speed_scaler            update_is_flying_5Hz      ModeRTL::update           ModeRTL::_enter
        calc_airspeed_errors        read_airspeed             set_servos
        send_SPort_Passthrough                        NavEKF2_core::UpdateFilter
        NavEKF3_core::UpdateFilter                    Log_Write_Control_Tuning
        verify_command_callback

        ^--- 16 layer3 callers
   ↓
    Called from layer 2 :
        AP_AHRS_DCM::update         airspeed_estimate         AP_Hott_Telem::loop       generate_LTM
        update_GPS_10Hz
        check_sensor_failures                         passthrough_wfq_adaptive_scheduler
        NavEKF2_core::SelectTasFusion                 NavEKF2_core::controlFilterModes
        NavEKF3_core::controlFilterModes              Plane::verify_command
        set_servos_controlled

        ^--- 12 layer2 callers
   ↓
    Called from layer 1 :
        drift_correction            estimate_wind             calc_velandyaw            send_EAM
        send_Sframe                 verify_vtol_takeoff
        AP_AHRS::airspeed_estimate                    AP_AHRS_DCM::airspeed_estimate
        check_sensor_ahrs_wind_max_failures           NavEKF2_core::readAirSpdData
        NavEKF2_core::detectFlight                    NavEKF3_core::detectFlight
        airspeed_ratio_update                         GCS_MAVLINK::vfr_hud_airspeed
        GCS_MAVLINK_Plane::vfr_hud_airspeed           Plane::suppress_throttle

        ^--- 16 layer1 callers
   ↓
    ====> AP_Airspeed::get_airspeed() const ====>
   ↓
    Calls to layer 1 :


        AP_Airspeed::get_airspeed(unsigned char) const
        ^--- 1 layer1 callee
   ↓
    Calls to layer 2 :

        ^--- 0 layer2 callees
   ↓
```

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025



```
Calls to layer 3 :

    ^--- 0 layer3 callees
↓
Calls to layer 4 :

    ^--- 0 layer4 callees
```

The information from Taphos's `radius` function both gives general information on the role `get_airspeed` plays in the architecture and on specific usage, both directly and indirectly, in the code base. For the former, we can see that `get_airspeed` is a *leaf* function, in that it is used by many other functions but uses few functions. It would be unwise to make significant changes to `get_airspeed` as that would affect many functions that use it. If we do need to make a change to `get_airspeed`, this output provides the starting point for components to review and test as part of the change process (e.g., a developer changing `get_airspeed` should at a minimum read through all of the layer 1 callers and should ensure automated tests exercise all of the calling functions at layers 1 and 2).

Domain Separation Analysis

Analyze → Domain Separation Analysis

Another way to analyze system element relationships is by looking at domains. A domain is a category. Domain separation analysis is the evaluation of system element relationships based on categorization. For example, we might categorize parts of a system in terms of which are high criticality and which are low criticality. Then we might analyze the system to determine whether any high criticality components have dependency relationships on low criticality components. Such a relationship would be a risk, as we would not want impact from a change to a low criticality component (like an entertainment system) to propagate to a high criticality component (like a flight controller).

To analyze relationships that are not stated explicitly (but may be implied through other means) we can use tools like Galois's CAMET Multiple Analyses for Domain Separation (MADS) tool. In addition to explicitly defined relationships, MADS also evaluates *implicit* relationships between components, such as that between two threads hosted on the same processor. Thanks to semantically precise languages like AADL, the dependency implications of relationships (such as a *binding* between software and hardware) are well defined and analyzable.

In our previous example, we followed the impact of a single modification propagating across three software subcomponents within the system architecture, using a SysML model as our navigation aid. This method is a viable first indication of the impact, but not detailed enough to identify the scale of the impact upon each indirect dependency. We can use the MADS tool, for

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

example, to more effectively probe the system architecture and assess the impact propagation in depth.

With the MADS tool we annotate hardware and software with *domains,* which describe the groups of functionality or criticality that we want to isolate within the architecture. We use MADS by adding MADS AADL *tags* to our SysML model. For example, we may want to separate components with different criticality levels, as described in the USAF airworthiness guidance.[38] We might identify domains for Safety Critical Functions (SCF) as directed by AC-17-01 (see Listing 3.3)

Listing 3.3 AC-17-01 Safety Critical Functions

- **Flight Critical** functions are functions used to achieve and control flight (loss or degradation could directly lead to loss of aircraft).
- **Operation Critical** are SCFs that are used for supporting a non-Flight Critical function that has inherent safety functionality associated with its operation (loss/degradation could directly lead to a consequence of Catastrophic or Critical hazard severity).
- **Indication Critical** are SCFs needed to provide indications to pilot/crew necessary for maintaining safe operation.
- **Emergency Critical** are SCFs that exist purely for the purpose of mitigating risk associated with emergency conditions.
- **Avoidance Critical** are SCFs needed purely to mitigate a potential safety risk.

Alternatively, we could identify domains for individual functions, such as Navigation, or individual capabilities, such as GPS location.

[38] https://daytonaero.com/wp-content/uploads/AC-17-01.pdf page 10

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

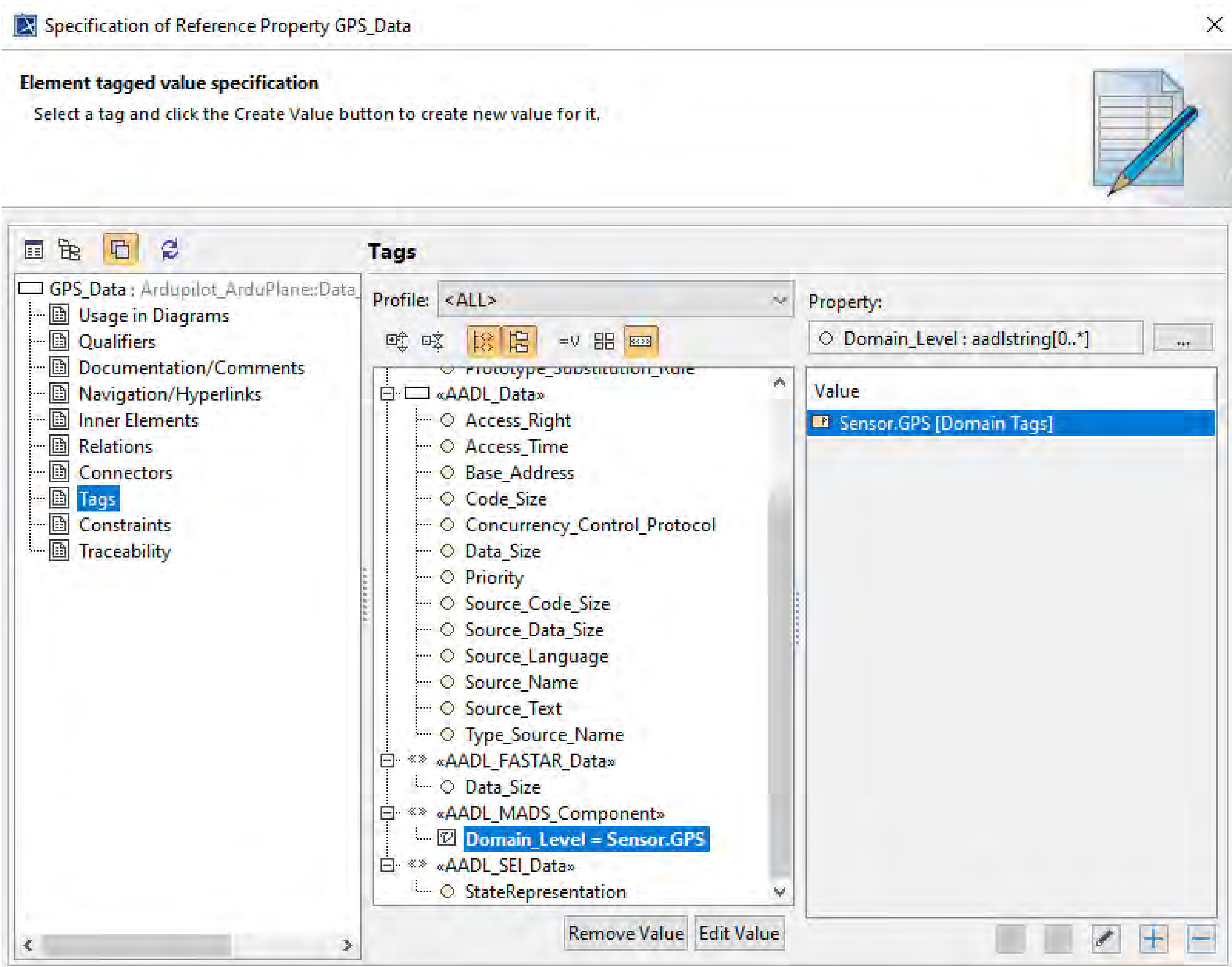


Figure 3.8 MADS Stereotypes in SysML

Failure Modes and Effects Testing (FMET) practices such as those recommended by USAF Airworthiness Processes[39] encourage *isolation* of safety critical functions and redundancy for safety critical capabilities. MADS analysis can determine whether such isolation is present in a system architecture by evaluating dependency relationships between components and comparing the domain labels of each component in a given relationship (See Figure 3.9).

[39] https://daytonaero.com/wp-content/uploads/AC-17-01.pdf page 28

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

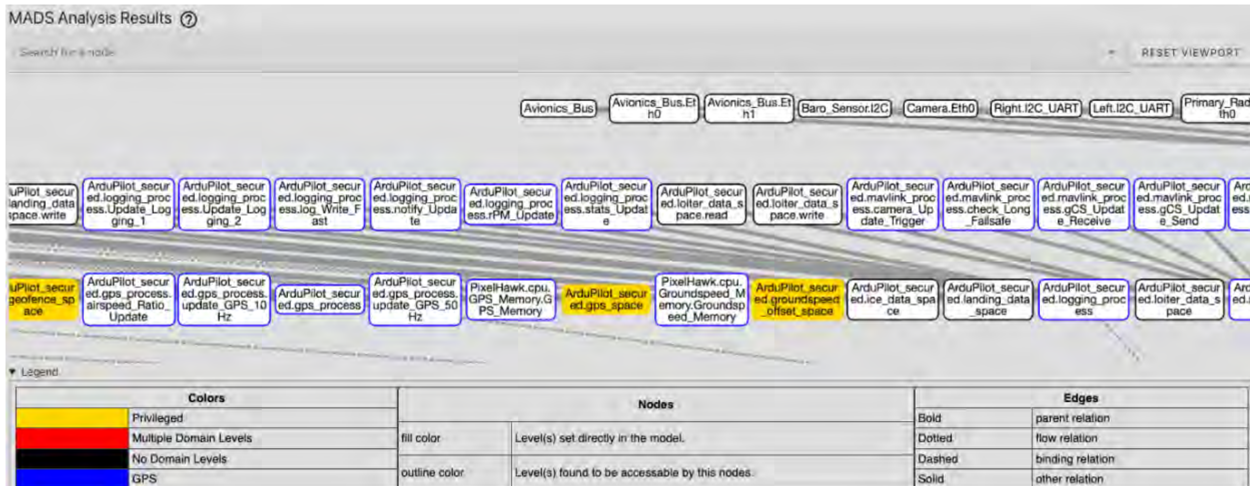


Figure 3.9 MADS Architecture Graph

Once tags are added to specific components in the system model, the MADS analysis traverses the system architecture and determines where isolation domains may overlap. The graph above, an output of MADS analysis, shows a subset of the components with dependencies on the SuperVolo Aircraft Platform. The graph is color-coded to show various isolation domains. In this diagram, we see a variety of components that all have containment (parent) relations to one another and do not have a domain level assigned.

On the bottom row of Figure 3.10, we can see several elements in a privileged domain (we will get to what that means later) and binding relationships that connect software to hardware. We also see an element from the GPS domain (we will also talk more about domains later).

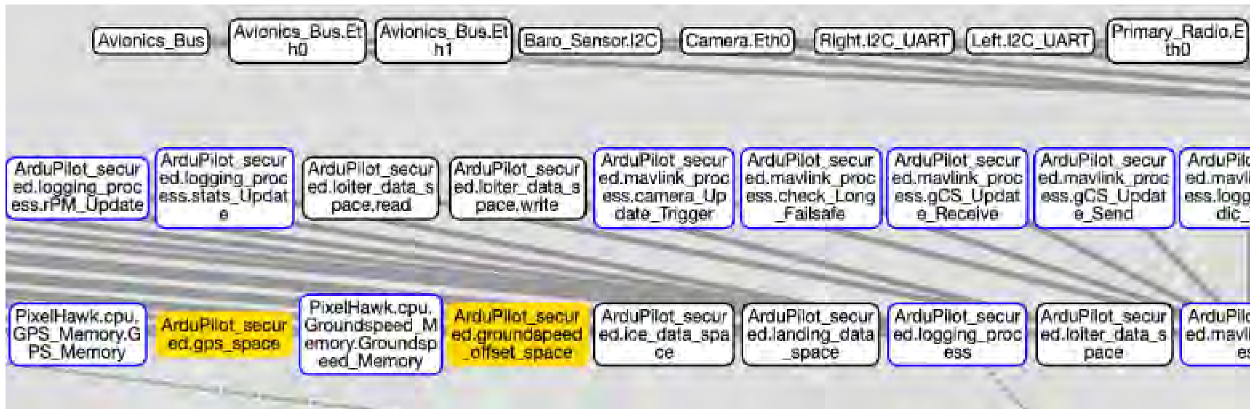


Figure 3.10 MADS Architecture Binding Graph, with two privileged components in gold

Using this output graph, we can quickly quantify the possible impact of a change to an element of our system by enumerating the components that are accessible through composition, data flow, binding, or other relationships. For example, we can see that there are many components including (`ArduPilot_secured.gps_space.write_gps` and

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

`ArduPilot_secuired.ahrs_space.read_ahrs`) bound to the AHRS Partition (see Figure 3.11).

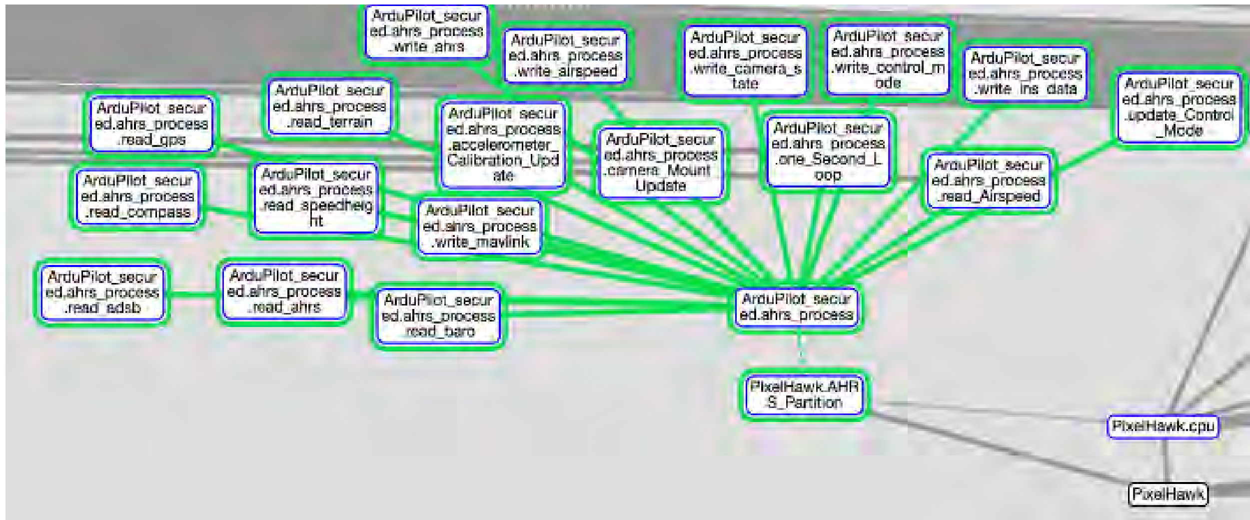


Figure 3.11 MADS Architecture Binding Hierarchy Graph

Using this graph, we can quantify the impact of a change to any given component by enumerating the components which depend on it or its dependents, and we can qualify the impact by assessing the type of dependency that relates them. For example, if we make a change to the computing schedule for the AHRS partition, that change may propagate to any of the green highlighted components in the figure above because they all rely on the AHRS partition.

In this example architecture, we have placed Ardupilot components into proposed *partitions*. A partition is an isolation strategy that provides guarantees of separation for some kinds of dependency relationship, such as resource usage. We'll discuss partitions more in Part 4. A change affecting CPU availability limited to the AHRS partition can propagate to **two threads**. In our reconfigured design, the PixelHawk CPU hosts 12 partitions, which together host approximately 50 threads. Thus a change to the PixelHawk CPU availability could propagate to **fifty threads** and therefore a rigorous testing plan would require all fifty be tested and validated.

Remember, there are *lots* of different ways one component can depend on another. The strategies we present here address *some* of these types of dependencies. For example, isolating two threads in different time and space partitions and executing them on a system that provides time and space separation will prevent them from affecting one another via processor or memory resource contention. If they explicitly communicate with one another (e.g., via inter-partition communication), then a change to one may still affect the other over that specified

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

interface. Such an interaction can be controlled and measured, and is a significantly weaker dependency (which is a good thing) than arbitrary time-and-space interactions via sharing process space.

We noted earlier that `ArduPilot_secured.gps_space.write_gps` and `ArduPilot_secuired.ahrs_space.read_ahrs` are bound to the same partition. A partition does not provide isolation between threads running in it, so a fault in `ArduPilot_secured.gps_space.write_gps` could affect `ArduPilot_secuired.ahrs_space.read_ahrs` as well. Partitions provide time and space isolation (thus breaking time and space dependency relationships) between computing elements; putting `ArduPilot_secured.gps_space.write_gps` and `ArduPilot_secuired.ahrs_space.read_ahrs` would alleviate the risk of change propagation between them. Although this is potentially at the cost of some computing efficiency, it can drastically improve the robustness of the system and reduce sustainment costs with the lower change impact propagation

The CAMET Library Risk Management Framework (RMF) and Multiple Independent Levels of Security (MILS) tools operate in a manner similar to the MADS tool - users label system elements (threads, data flows, processing hardware, etc) according to domains (criticality level, security level) and the tool evaluates the relationships between system elements to identify any improper relationships between system elements in different domains.

What about change propagation within individual software components or between threads? If one thread depends on data from another, they cannot be isolated completely. Within a given thread, there may be software components which may or may not affect one another. To get more clarity on those relationships, we need to dig deeper with Taphos, as described in the prior section.

System Resource Change Impact Analysis

Analyze → Resource Analysis

A common way for a change impact to propagate across components in an architecture is through resource contention. For example, a new piece of software may be free from internal errors, and may be free of errors in its communication with other software components. However, if it requires compute resources, it may cause a *resource contention* problem, in which two or more components require more of a shared resource than is available (in Part 7 we encountered this exact problem!).

Model → AADL Models are intended for analysis of resource contention. Tools such as those in CMU SEI's OSATE and Galois's CAMET provide analysis of competing demands for shared resources. For example, Galois's FASTAR Utilization tool analyzes *binding* relationships

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

between consumers of a resource and producers of that resource, comparing what is available to the demand and alerting the user if demand exceeds supply.

Analysis → Behavior Analysis

Behavior analysis refers to automated evaluation of system element behavior, often in terms of system element state and state change. On the DARPA V-MESA project, Galois developed a method for automated learning of *behavior models* for hardware devices. Behavior models describe how a system acts or how it reacts to different types of input. The V-MESA project explored ways to use such models to identify vulnerabilities or incompatibilities in components (we applied this technology on the SuperVolo GPS, more detail in Part 5 and V-MESA GPS Analysis).

Regression Testing

Analyze → Regression Testing

A test suite is a collection of test cases that can (usually in an automated fashion) exercise a system to determine whether its behavior is correct. A regression test suite is a collection of tests designed to catch error conditions in which a system's behavior ceases to match its specification. Rigorous software development projects often run a regression test suite on the code regularly (the CAMET Library, for example, goes through regression tests roughly every day).

Unlike the previous topics in this section, a regression test suite is not targeted at a particular change. Instead, a regression test suite is a "catch all" designed to find any issues that crop up, regardless of their origin.

A software regression test suite is a test suite that exercises the software of a system and flags potential issues. A model-based analysis regression test suite exercises models of a system and flags potential issues. For cyber-physical systems, we recommend using both a software regression test suite *and* a model-based analysis regression test suite.

We used Gitlab's continuous integration and continuous deployment (CI/CD) tools to automate building and analyzing both the SuperVolo's source code and its models. The Taphos-based dependency matrices shown above are generated automatically as part of this automated workflow.

We used the CAMET Library Continuous Virtual Integration Toolkit (CVIT) tools to create and run an automated set of model-based tests on models of the SuperVolo architecture. For example, our model-based analysis regression test suite evaluates the modeled weight of SuperVolo components against a weight limit, flagging any changes that increase the overall

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

system weight above the limit (this is an example of **Analyze → Resource Analysis**), shown in Figure 3.12.

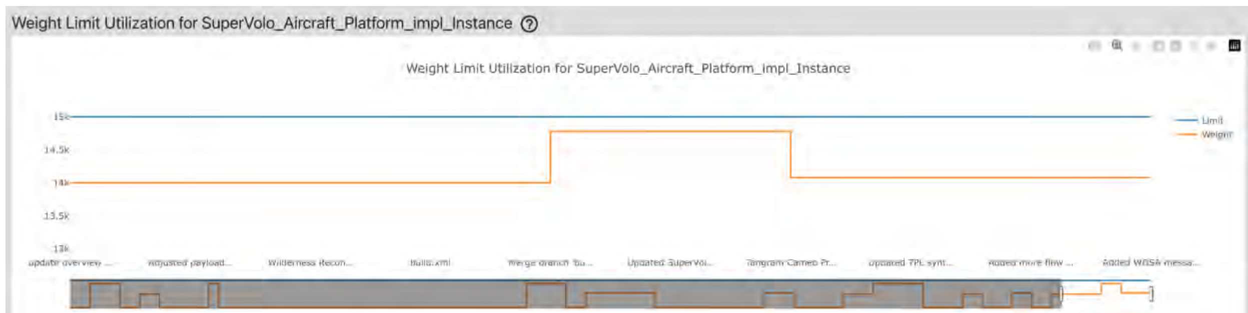


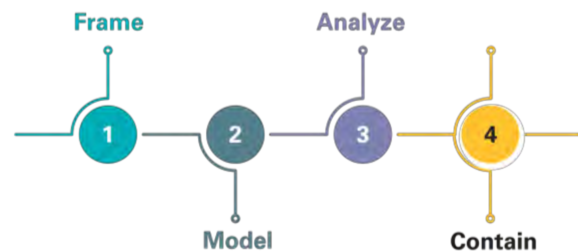
Figure 3.12 Automated regression testing of the SuperVolo architecture against weight limits. Each change to the architecture models triggers a new test.

Automating regression tests (both for software and hardware) are one way to keep the time and cost of testing the systems low. It also catches potential issues early, especially those that have been observed previously. The regression automation is one way to preserve engineering expertise and know-how, so we don't have to keep finding the same bugs in operational systems. Regression testing and analysis is an effective approach to *contain* observed defects and classes of defects to prior phases of development.

Conclusion

Using Galois's Taphos and CAMET tools, we created a digital map of the SuperVolo and used that map to evaluate the impact of various changes to its architecture. In Part 4 we'll use these evaluations to determine a path toward modularity and rapid changeability.

Part 4: Contain - Designing a Change Ready Architecture



In Part 3 we introduced methods for evaluating the *impact* of a change. In this section we'll discuss how to use those methods to choose why, where, and how to refactor an architecture to contain change to reduce its impact by reducing or removing avenues of change impact propagation. If we can reduce or remove avenues of change impact propagation, then we can reduce the need for re-certification.

There are a variety of changes that can prompt recertification. The Defense Acquisition

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

University (DAU) provides a list of example changes that require re-certification. For example, any change to the navigation system effectiveness requires recertification.[40] One objective of removing or reducing avenues of change impact propagation could be to prevent changes unrelated to the navigation system from impacting the navigation system, as such changes would trigger re-certification.

The previous steps, Frame, Model, and Analyze, help identify these sorts of changes in the CPS, as well as the components and dependencies involved. The “FMA” parts of the FMAC process helps us understand the avenues of change impact propagation. In “C” we can go about removing those avenues through a variety of containment approaches, with a particular method chosen to fit the context.

Contain → Refactoring

In the software development world, *refactoring* is a kind of change to an architecture whose purpose is to restructure existing code to improve its maintainability, portability, or other attributes. Many software developers refactor software with the same goals as we’ve described here - to better account for dependencies to reduce the likelihood of defects.[41]

> *Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.*[42] - Martin Fowler

The Cost of Change

Contain → Change Cost

In this case, we’re concerned with refactoring software *and* hardware, and doing so with the aim of reducing the cost of future changes (both money and time). The greater the impact of a change, the higher its cost. This cost is magnified in critical cyber-physical systems, because testing and certifying such systems is expensive, and the cost of problems found in later-stage testing are magnified by testing and certification costs.

If you can implement an architecture that is ready for change (that is, an architecture which minimizes change impact), then you can reduce overall sustainment costs. For example,

---

[40] The Defense Acquisition University (DAU) provides a list here: https://www.dau.edu/acquipedia-article/airworthiness-airworthiness-certification

[41] https://thomas-zimmermann.com/publications/files/kim-fse-2012.pdf discusses software refactoring at Microsoft

[42] https://www.refactoring.com/ is the web site of Martin Fowler, the recognized authority on software refactoring.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

suppose you are maintaining a system that has approximately 5 major changes per year and each change costs about 400 hours of engineering effort. You'll have sustainment costs as shown in blue on the plot below. If you decide to invest in developing a change-ready architecture, you'll make an up-front investment that will pay off in lower costs per-change later (red plot below). Note that after year 3, the sustainment costs for the change-ready architecture are significantly lower (See Figure 4.1).

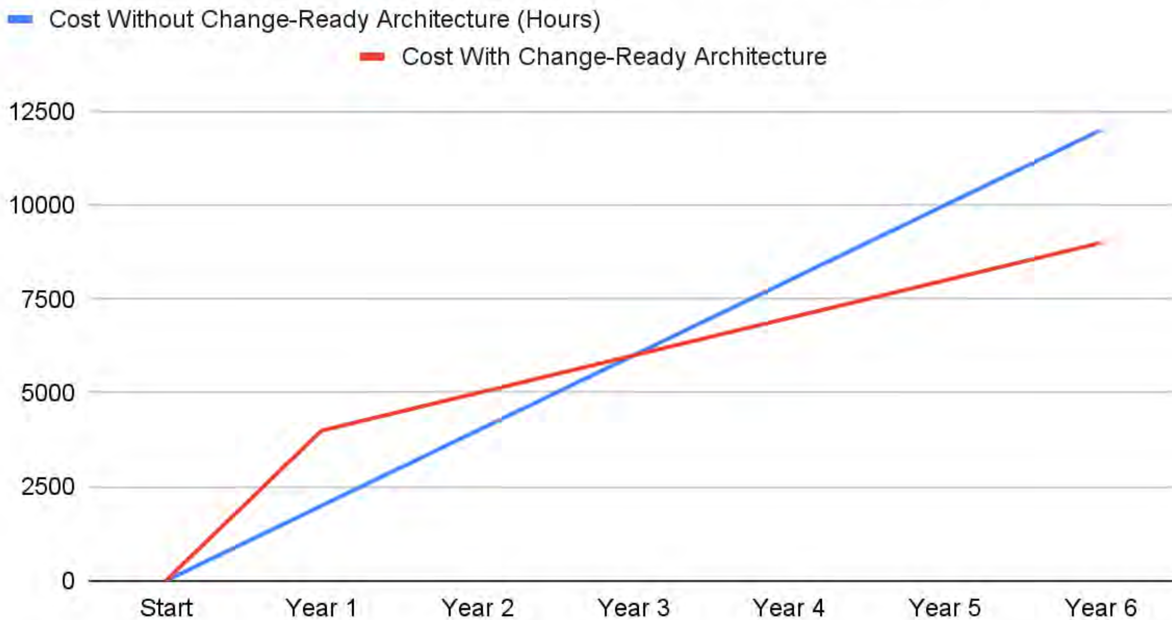


Figure 4.1: Up Front Costs Versus Sustainment Costs for a Hypothetical System

For this project we sought first to define design structure matrices for SuperVolo that allow us to assess a component-to-component dependencies for a variety of types of relationships. Next, we use those matrices to select and incrementally apply changes to the SuperVolo to progressively *decrease* the impact of each subsequent change (putting us on the red track in the chart above).

An architecture is change-ready if the **cost to make changes is low compared to the cost to make an analogous change is a comparable architecture** (e.g., in prior versions of a system). In cyber-physical systems, the cost of change is largely driven by costs incurred at the test and operation stages of development (in contrast to pure-software fields, such as web design, where fixing changes to deployed code is relatively inexpensive). Figure 4.1 and Figure 4.2 show several cost calculations from different cyber-physical system domains:

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

	Method 1 Cost Factors	Software Cost Factors
Requirements	1X	1X
Design	8X	5X – 7X
Build	16X	10X – 26X
Test	21X	50X – 177X
Operations	29X	100X – 1000X

Figure 4.2 Cost factors for software through development phases from Error Cost Escalation Through the Project Life Cycle[43]

The Aerospace Vehicle Systems Institute reports similar costs:

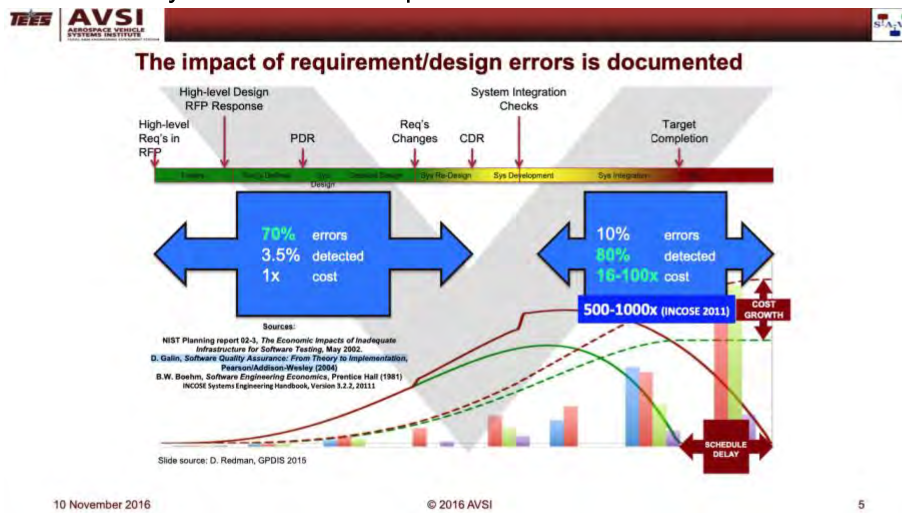


Figure 4.3 From System Architecture Virtual Integration (SAVI) Project: Intermodel Error Checking and Consistency Review and Demonstration An Aerospace Vehicle Systems Institute Project (AVSI) Presented by Greg Pollari (Rockwell Collins) and Nigel Shaw (Eurostep)

The need for low-cost change is a primary driver of the DoD's Modular Open Systems Approach (MOSA), which was written into law in 2017. [44] The principals of MOSA are well aligned with the containment objectives we discuss here.

New components are commonly unit tested (tested in isolation) before being added to a system architecture (top right of following figure). However, studies consistently show that defects found in **integration** and **operational phases** of the cyberphysical system lifecycle are significantly more expensive to fix (center right and left of following figure). These are the lifecycle phases in which most or all of the components in a system are connected and operating together. Defects found during these phases are those in which a change impact has propagated well beyond its source, but its propagation (red in the following figures) has not been detected.

[43] https://ntrs.nasa.gov/api/citations/20100036670/downloads/20100036670.pdf
[44] The FY17 NDAA includes requirements for MOSA in Sec 805-809 as required by Public Law 114-328.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

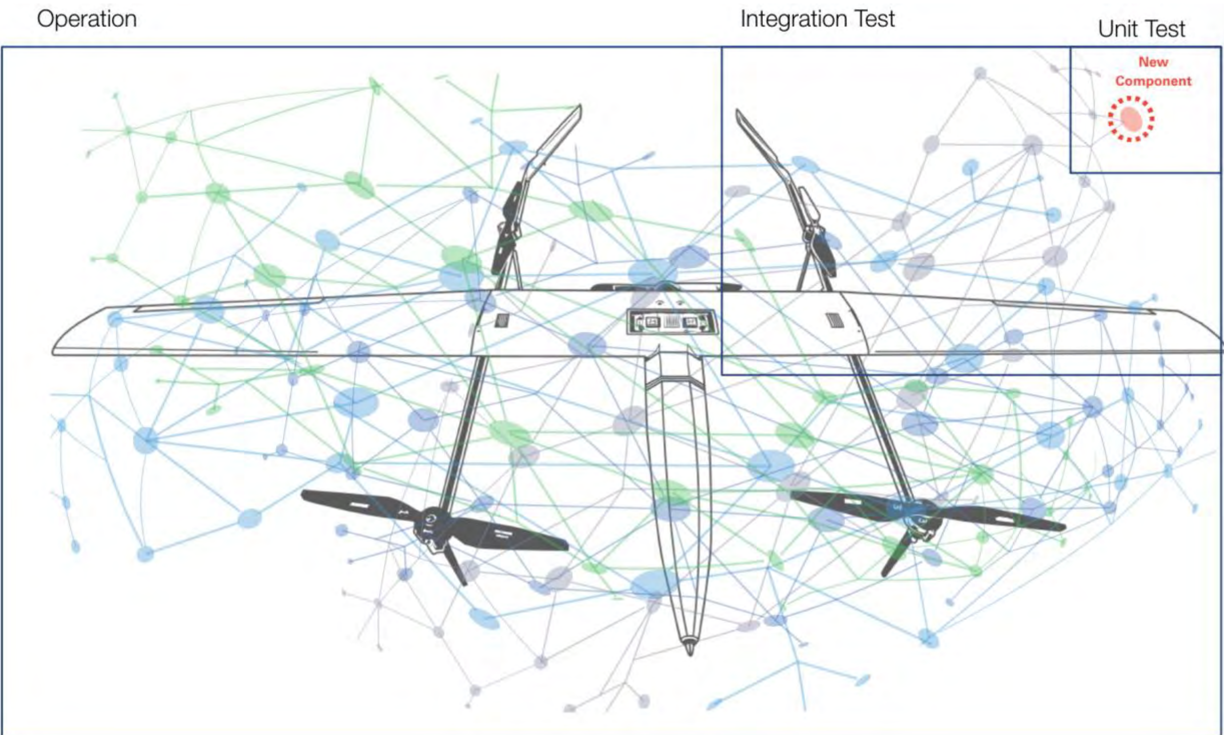


Figure 4.4 Abstract representation of the scope of different phases of test and operation. Note that dependencies between components mean that a change made to a single component can propagate throughout the architecture.

If the impact of a change propagates beyond the boundaries of your testing, you cannot determine whether it will manifest in operation.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

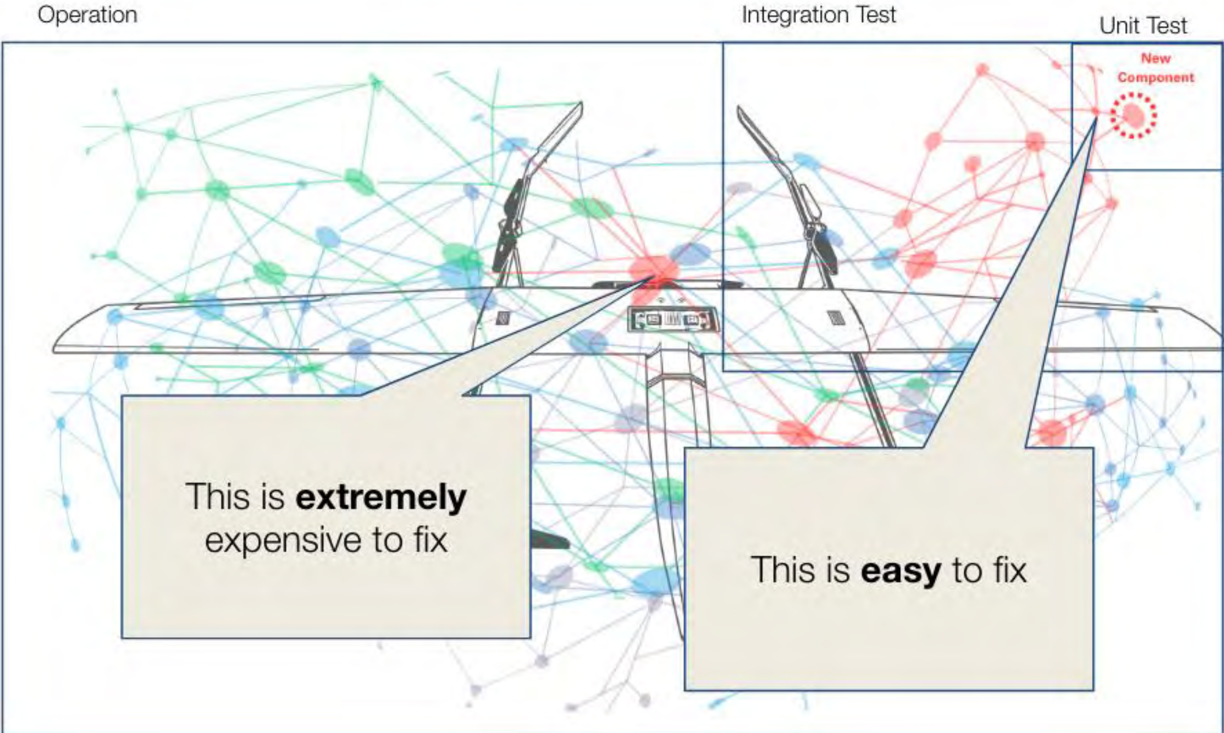


Figure 4.5 Change impact that propagates outside of unit and integration test is extremely expensive to fix.

There are several methods we can use to reduce the likelihood of change impact propagating into the integration or operation phases (recall, the GAO reported that in 2020 23% of all software defects in the F-35 were found after software delivery to the test aircraft. We'll save a lot of time and money if we can get that number down!)

We're left with a few basic options: we can **remove avenues of change propagation**, we can **reduce the amount of change propagation on existing avenues**, and we can **detect change propagation in earlier development stages**.

Reduce the Amount of Change Propagation on Existing Avenues

Contain → Reduce Propagation on Existing Avenues

The first and simplest strategy is to use analysis tools (from Part 3) to evaluate different approaches to making a change and select the approach that minimizes change impact.

Method: Pick the least impact change

Contain → Find the Least Impact

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

The simplest approach to limiting the amount of change impact propagation is to make changes in a minimally impactful manner. For physical changes, this is often obvious. For example, if we need to add a data storage device to the SuperVolo, adding it in a way that does not change the exterior profile of the SuperVolo will likely have much less change impact than making the change in a way that alters the aerodynamic properties of the SuperVolo.

In other contexts, particularly those involving software, the least impact change approach might not be so obvious. Using a Design Structure Matrix (**Model → Design Structure Matrix**), we can organize our view of the architecture by the degree of dependencies on each component. For example, we might consider adding the data storage device by connecting it to the main flight controller. Design Structure Matrices make it easy to evaluate this directly - how many components directly use the flight controller?

We can also evaluate indirect dependencies. For example, using tools like the Taphos Call Radius tool (**Analyze → Software Dependencies → Call Radius**), we can evaluate several possible methods for implementing a change to determine which has the least change impact on dependent components multiple steps away. For example, in Part 7 we discuss a change to add a new location data sensor to the SuperVolo. We identified several candidate locations in the code base where we *could* add code to handle the new sensor, then we use the Taphos Call Radius tool to determine which locations have the highest or lowest degree of connectivity to the rest of the code base (the highest or lowest number of change impact propagation avenues). The location with the lowest degree of connectivity is likely to have the least risk of change impact propagation to other parts of the system.

The simplest way to minimize the impact of a change is to enumerate the possible methods for implementing the change, compare the number and type of dependencies on components likely to be affected by the change, and then pick the method with the smallest number of dependencies (we use this in Part 5).

Removing Avenues of Change Propagation

Contain → Remove Avenues of Change Propagation

Selecting the minimum impact approach for a given change is often viable in the short term, but is also a good way to accumulate *technical debt*. In the long run (as described in **Contain → Cost**), you can reduce cost by modifying larger portions of your architecture to reduce the impact of *future* changes.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

There are three primary methods for changing an architecture to remove avenues of change propagation entirely: **integration of interoperability standards, addressing problematic dependencies in the architecture,** and integration of **privileged components**, such as guards, data isolators, or cross-domain solutions**.**

Method: Break Dependencies with Interoperability Standards

One of the best ways to break dependencies between components is to use standards. Standards are common in both traditional software and cyber-physical systems, but they are important enough to warrant explicit mention here.

Thanks to standards for communication busses in aircraft and automobiles, manufacturers can add or change devices that *use* those communication busses without having to change the busses themselves. Standards like the Portable Component Interface (PCI) standard for computers have become ubiquitous *because they work*. Can you imagine having to re-write portions of your operating system for every new device that you want to connect to it? It would be a nightmare.

Similar themes play out in software-only environments. Standards governing network traffic (such as the TCP/IP protocol) are fundamental to creating systems of networked components that can be readily changed or extended.

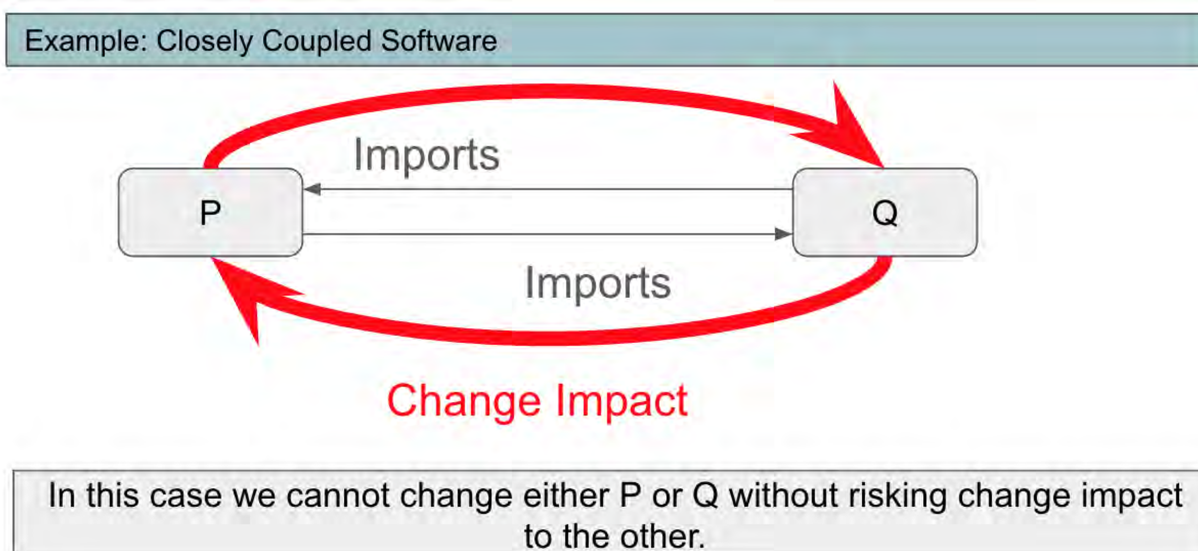
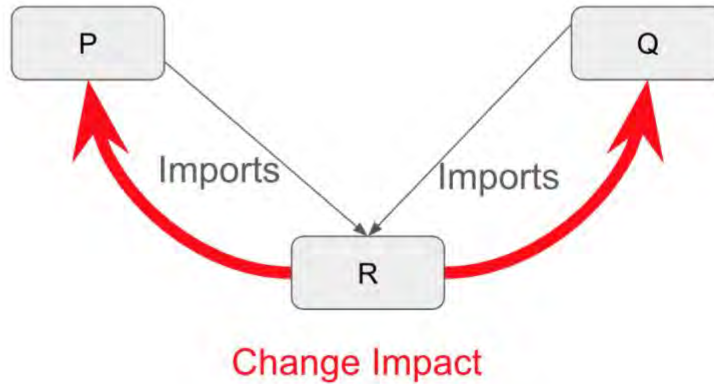


Figure 4.6 Illustration of Cyclic Dependencies Between Software Components

Standards are less common, and often more difficult to apply, in niche domains like avionics. Nevertheless, a variety of standards like ARINC, FACE™, and OMS are available to break inter-component dependencies.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025



In this situation we must refactor P and Q, such as by adding a new component R that both P and Q can depend on.

Figure 4.7 Breaking Cyclic Dependencies through Refactoring

When we add a standard we remove avenues of change propagation between individual components (see Figure 4.6 and Figure 4.7). Now as long as the standard remains stable, the individual components can change with less risk of impacting one another. See Table 4.1 for examples of Avionics standards.

Table 4.1 Examples of Avionics Software Standards

Resource	Description	Notes
ARINC	Standards for aircraft hardware and software	See https://aviation-ia.sae-itc.com/product-categories/arinc-standards
Open Mission Systems (OMS) Standard	Standardized message set for avionics.	
Future Airborne Capability Environment (FACE)™ Technical Standard	Standardized data modeling ecosystem for avionics software.	
Weapons Open System Architecture (WOSA)	Standardized message set for weapons systems.	

Method: Code Generation

Errors can manifest when implementing a change in a variety of ways. Some errors, such as invalid requirements, can be detected using methods such as requirements validation (**Models**

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

→ **Validation**). Other errors can occur as a result of human error. Just as dependencies exist between components in a physical architecture, dependencies exist between design artifacts. For example, the source code implementing a change depends on the design artifacts (e.g., models) describing that change. A human implementing that source code increases the risk, because the code then depends both on the requirement *and* on the human implementing it! We can remove this dependency on humans for the code by generating code directly from the design artifacts. This approach has an upfront cost, but many benefits:

- The avoidance of common errors and bugs introduced by manual entry,
- Automatic traceability to architectural, interface, and communication standards,
- Improved traceability to formal system requirements,
- Improved tracking and propagation of low-level changes, especially updates to data, message, and interface formats,
- Documentation of architectural definition used for system qualification.
- Improved integration with design and implementation testing and verification tools.

Certain modeling tools and languages support code generation. In this context, auto-generated source code is integrated directly into an existing system software build environment with little or no manual modifications. For complex systems at scale, auto-generated code usually gains insertion into specific subsystems or cross-section of a design, instead of replacing the entire system. Often code generation addresses a domain-specific design property, such as communication message or data formats, or a specific design/implementation standard. In total, these benefits work to reduce overall risk when updating and re-qualifying a system design.

For our SuperVolo development (more detail in Part 7), we have integrated into its design the WOSA messaging standard to pass information between certain scheduled avionics tasks. For example, the information could be taken from an eLoran compatible radio. Operations that communicate according to the WOSA standard require functions to serialize and de-serialize message payloads. Because the WOSA standard is extensive, we used the auto-generation tool TangramPro, which supports WOSA messaging, as well as other messaging standards.

By modeling the relevant operations and internal communication paths in TangramPro, we are able to automatically generate source code that serializes/de-serializes the specific WOSA message types needed within the SuperVolo, without manually validating against the WOSA specification. This approach saves hours of manual development, validation, and testing, and reduces the risk of introducing unexpected errors.

Method: Address Problematic Dependencies in the Architecture

Software systems are particularly susceptible to problematic dependencies in their architecture. For example, suppose a system has a software component called “data logger” responsible for logging location, telemetry, and activity data throughout each flight. Such a component might

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

have many *incoming* dependencies, as it could be used to log activities from many components (e.g., sensor control, communication, etc).

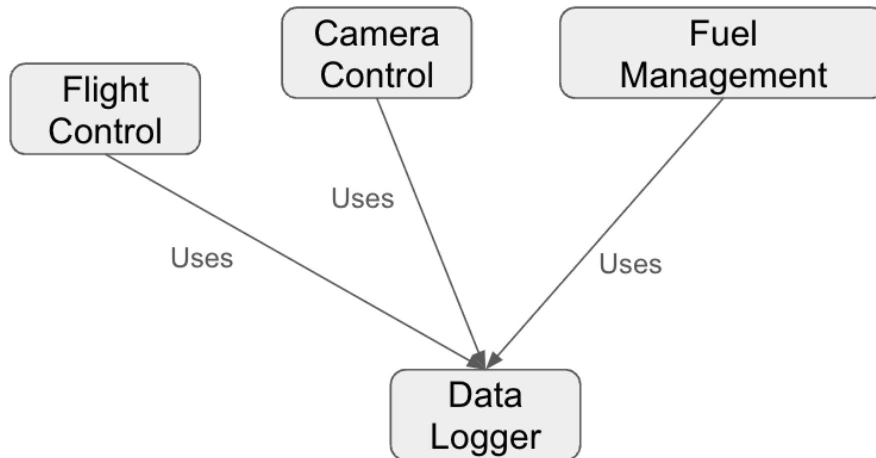


Figure 4.8 Multiple Software Components using the Data Logger

This configuration is not inherently problematic - in fact it's desirable! We want common capabilities (such as data logging) to be written once and used many times. However, sometimes mistakes lead to circular dependencies.

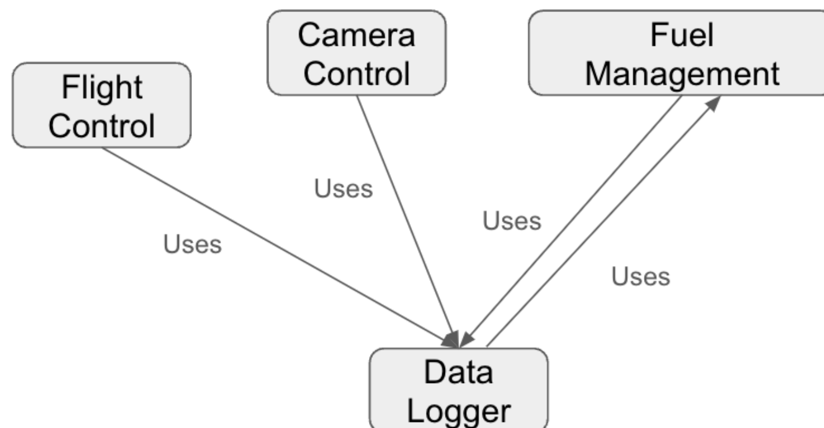


Figure 4.9 Problematic architecture: the data logger depends on the fuel management system.

Suppose a software maintainer opted to use a function defined in Fuel Management directly from the data logger. In doing so, the **maintainer unwittingly added an *implicit dependency* from the Flight Control software to the Fuel Management software**! Now impact from a change to the Fuel Management software could propagate to the flight control software,

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

increasing the risk and requiring increased testing scope. Tools like Lattix can help us find and resolve these problematic dependencies.[45]

Method: Contain Components

There are times the impact of refactoring architecture to reduce / remove dependencies is not feasible. For those cases, there are other technologies for modifying parts of the architecture with guarantees that you will avoid impacting behavior elsewhere.

For example, it is not possible to eliminate dependencies from application software to the operating system on which it runs. However, incorporating standard APIs (e.g., POSIX, ARINC) can mitigate risks by constraining changes to the operating system. Formal methods approaches (such as the provably correct behavior of Sel4) can further mitigate change impacts.

There are many ways to contain a component - for every dimension along which change can propagate, there are mitigations we can apply to eliminate or reduce the change impact. For example, isolating unrelated software components in different memory spaces (as done by most operating systems) mitigates potential change impact propagation pathways between software components.

For the SuperVolo, we opted to reduce change impact propagation avenues by containing components in a separation architecture, discussed in Part 6.

Method: Software Refactoring

Ideally, upgrading a legacy system does not require a large refactoring of existing tasks, particularly tasks captured as scheduled execution threads. A significant re-implementation of a software task risks creating new dependencies and unexpected change impact on the system design. A large refactoring in this context includes modification of data flow access (incoming or outgoing interfaces), modification to software-hardware bindings, and significant modification to computations or functionality. Under certain conditions, however, a large refactoring is needed to address updated architectural, safety, security, performance, or functional requirements. In these scenarios, a task may need its functionality split up into multiple new tasks.

More specifically, reasons for splitting up task functionality include:

- Reducing maintenance of a task with expanded functionality,
- Addressing updated timing or scheduling requirements,

[45] https://docs.lattix.com/lattix/userGuide/Applying_Algorithms#other-algorithms

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

- Improving network, memory, or processor hardware utilization,
- Limiting access to certain directional data flows and/or global variables,
- Limiting access to certain hardware components,
- Enforcing conformity to an API or interoperability standard.

Often in safety-critical systems, separating functionality is done in a way that partitions the memory, execution space, and timing. Such partitioning provides an additional layer of security and safety by preventing tasks from unauthorized memory access or unauthorized execution times. To enforce this partitioning, the software must rely on underlying operating system and/or hardware infrastructure. For example, separation kernels enforce task isolation, strict inter-task communication, and memory access specifications.[46] Standards like ARINC 653 (which a separation kernel may implement) define a protocol for task execution time partitioning.[47] In partitioned systems, the decision on how functionality is mapped to specific tasks, threads within tasks, and timing partitions becomes a crucial activity in the design process or subsequent system upgrades.

Method: Choose Safe Programming Languages and Static Analysis

Programming languages like `C` and `C++` allow functions to have side effects, which are changes to system state that may not be obvious to the caller of the function. That means the impact of a change can propagate both to users (callers) of a changed function and to used (callee) functions of the changed function.

Although maintainers of a legacy system may be daunted by the prospect of changing programming languages, such a change can significantly reduce the risk of change impact propagation. For example, memory usage is one such side effect concern - if a function allocates or deallocates memory, changes to when that function is called can affect the overall system state (since memory is a shared physical resource). Memory safe languages like `Rust` can significantly reduce this risk. Formal analysis tools like `framac` can also help mitigate the risk of side effects for code written in `C`.

Unfortunately the Ardupilot software is written in C++, and a rewrite in another language was out of scope for this study.

Detect Change Impact Propagation in Earlier Phases

In some cases it is not feasible to reduce or avoid change impact propagation. In these cases, the next best option is to detect change impact propagation as early as possible. We used two

[46] https://en.wikipedia.org/wiki/Separation_kernel
[47] https://en.wikipedia.org/wiki/ARINC_653

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

methods for this effort: hardware in the loop testing and Architecture Centric Virtual Integration Process (ACVIP).

Method: Hardware in the Loop Testing

We created a Portable System Integration Lab (PSIL) to provide a platform to test out software and hardware configurations of the SuperVolo Ardupilot system before loading them onto the SuperVolo aircraft. A key feature is to provide feedback on functionality of software and hardware changes without having to replicate all of the aircraft components. This journal documents the design and development of the SIL using digital engineering (DE) methodologies. We used this PSIL extensively for testing configuration changes, particularly those described in Part 5 and Part 7.

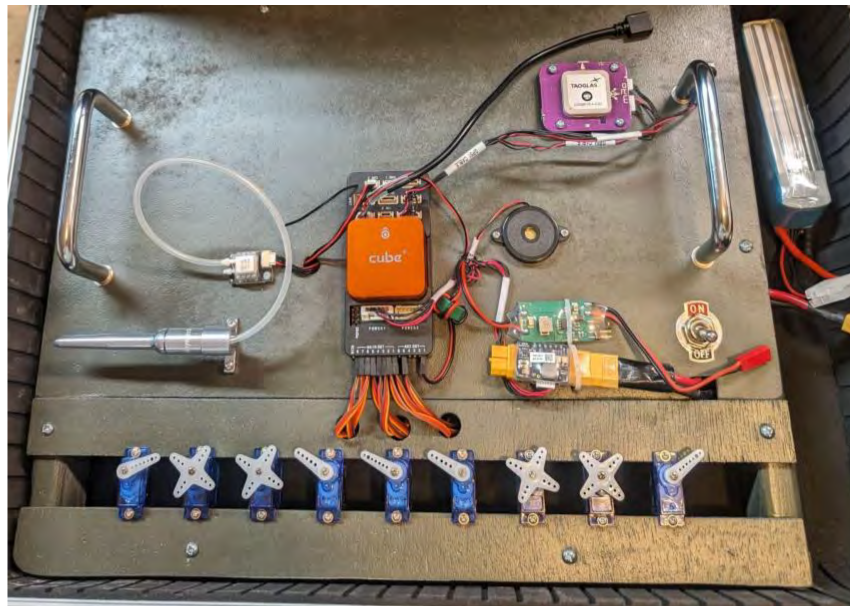


Figure 4.10 SuperVolo Portable System Integration Lab (PSIL)

Method: Architecture Centric Virtual Integration Process (ACVIP)

The Architecture Centric Virtual Integration Process (ACVIP) is a methodology for early model-based evaluation of cyber-physical system architectures to identify defects or risks. ACVIP practitioners use models (like those described in Part 2) and analysis tools (like those described in Part 3) to evaluate candidate system designs or configurations for risk.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

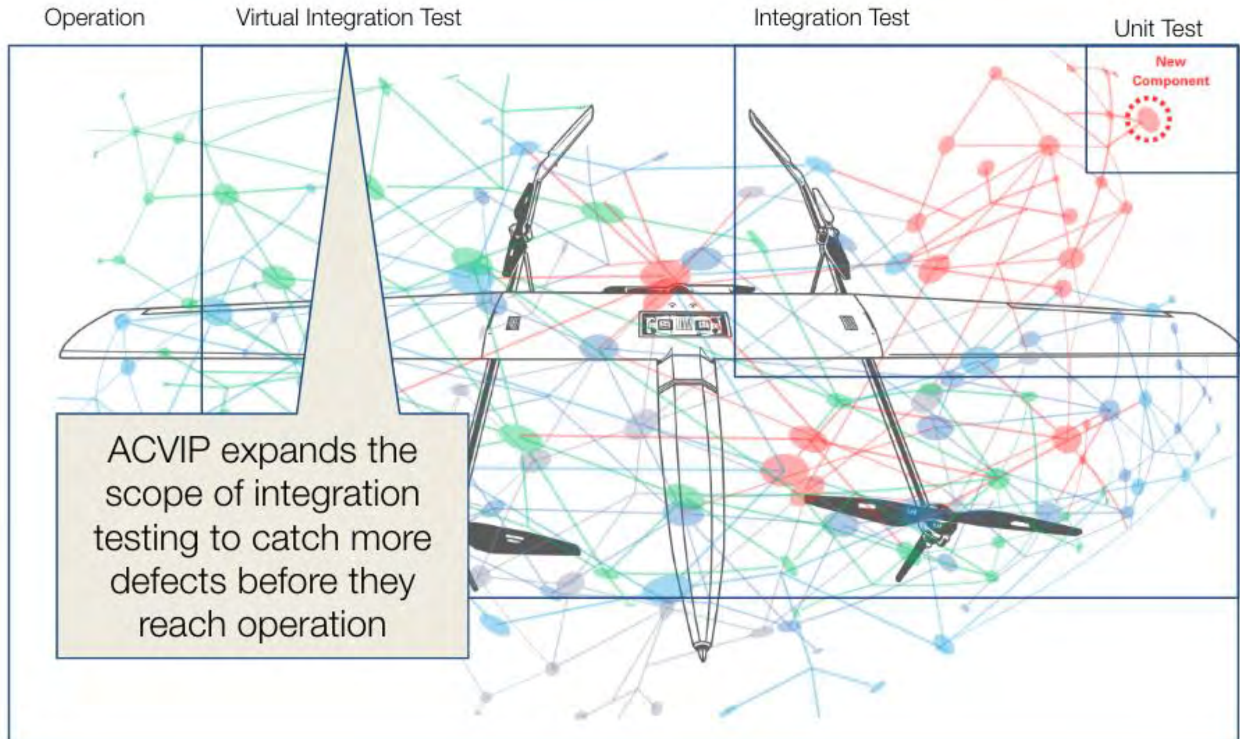


Figure 4.11 ACVIP Expands the Scope of Integration Testing

ACVIP is defined by the ACVIP Handbooks.[48] ACVIP practitioners determine which parts of their architecture are at highest risk for change impact, then select analysis tools to help detect change impact propagation affecting those areas.

[48] The ACVIP Modeling and Analysis Handbook is publicly available: https://cdn.prod.website-files.com/673b407e535dbf3b547179dd/6786ed77930332b4ced3988c_ACVIP-Modeling-%26-Analysis-Handbook_Mar2021_DistA.pdf

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

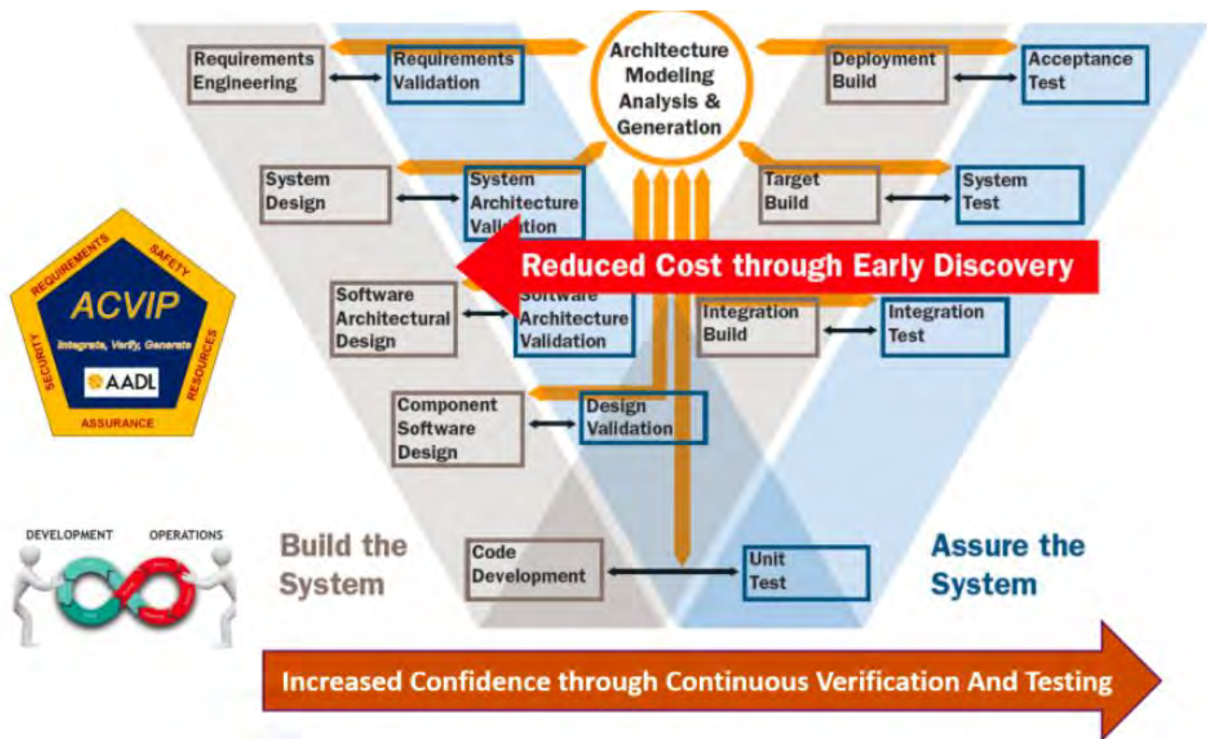


Figure 4.12 ACVIP as part of the Systems Engineering "V" Model, from "ACVIP, A Key Component of the DoD Digital Engineering Strategy"[49]

In our experiments with the SuperVolo we used ACVIP in several places, such as Size Weight and Power (SWAP) budgets, processor performance capacity and scheduling, and domain isolation. We used gitlab continuous integration / continuous deployment (CI/CD) ecosystem to automatically and regularly run ACVIP analysis tools. When you run analysis tools proactively and regularly, you can find and correct unexpected change impacts earlier and save time and money.

Conclusion

In this section we introduced several strategies for *containing* change to reduce its impact. We can use automated analysis tools to detect unexpected change impacts early. We can reduce change impact risk by removing or reducing avenues of change propagation. These methods include use of separation architectures to eliminate change propagation avenues; code generations to reduce variation points and align code with models; standards and interfaces to reduce change at architectural boundaries; and refactoring code to isolate high-change software.

[49] https://insights.sei.cmu.edu/documents/1510/2019_021_001_634975.pdf

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Case Study Overview

In the following three case studies, we demonstrate application of the approaches discussed in Parts 1-4 to a real cyber-physical system, the SuperVolo. In the first case study we demonstrate methods of *measuring* the impact of change using MBSE artifacts. In the second, we demonstrate a workflow for identifying and isolating *domains* of functionality in a cyber-physical system using MBSE artifacts. In the third, we demonstrate use of *standards* and *reference architectures* to limit the impacts of change. Each of these studies describe a different mechanism that contributes to *increasing the speed* at which we can build and maintain cyber-physical systems.

For each of the case studies, the system boundary (**Frame → System Boundary**) was the combination of the SuperVolo, its ground station controllers, and potential additional system elements.

We expect most readers will not be able to follow along with everything in these case studies. However, we strive to include enough detail that you can re-create the results in your own context, and we have included several tutorials with enough detail to follow along directly.

Part 5: Case Study - Selecting a New GPS

The *New GPS* case study explores use of MBSE artifacts to measure the change impact (and associated risks) of replacing the Global Positioning System (GPS) unit in the SuperVolo. In this case study we demonstrate a range of systems engineering tools and methods to explore options for upgrading the GPS in the SuperVolo aircraft. We perform trade studies exploring the various options before making changes to the physical aircraft. To motivate this case study, we hypothesize that users of the SuperVolo had identified a risk associated with the current GPS unit and needed it to be replaced (**Frame → Identify Stakeholders**).

A GPS unit has a variety of dependencies to and from its host system that make changing it a potentially high-risk operation. For example, the physical GPS unit might require power, cooling, and physical mounting from its host system. The GPS unit might depend on a specific bus architecture to communicate. The host system might require specific information from the GPS unit, or might require it at a specific rate. The GPS software on the host may depend on a specific message set supported by the GPS, and might depend on other capabilities of the system.

We used several of the systems engineering tools and methods discussed in Parts 1-4 to explore and analyze options for upgrading the GPS in the SuperVolo aircraft. We designed a Portable System’s Integration Lab (PSIL) to test out a number of hardware configurations. By conducting the Frame, Model, Analyze, and Contain activities with options for a new GPS unit,

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

we show how to characterize the impact of potential changes and make impact-informed decisions.

Frame: Mission Requires new GPS

Frame → Identify Requirements

For this case study, we hypothesize that a flaw has been identified in the existing GPS on the SuperVolo and that it must be replaced. The GPS is a critical component of the SuperVolo architecture, so before selecting a replacement we need to thoroughly evaluate the possible impacts of such a change.

Notably, we do not assume a performance limitation with the current GPS (that is, we need a replacement that matches the existing functionality, not a replacement that adds capability). New capabilities (such as higher refresh rate or higher precision) would be nice to have but are not required. We assessed the available hardware options and selected the candidates shown in Table 5.1.

Table 5.1 Candidate GPS Units

Supplier	Model	GPS Chip	Notes
ARK	RTK GPS	Ublox Zed-f9p	
ARK RTK	GPS	Ubloxk Neo M9N	
CubePilot	Here3	Ublox Neo M8P	
CubePilot	HerePro	Ublox Zed-f9p	
Holybro	M8N GPS	Ublox Neo M8N	
Holybro	M9N GPS	Ubloxk Neo M9N	
Holybro	RTK GPS	Ublox Zed-F9P chip	
mRo	mRo GPS u-Blox Neo-M8N IST8308	Ublox Neo-M8N	This is what is currently in the SuperVolo
Septentrio	various	Ublox competitor	
Sky-Drones	SmartAP GPS	Ubloxk M8N	
SparkFun	different options	Ublox F9P, M8N, M10, etc...	

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

On assessment of the product information on the available options, we determined that there was little variance between the options, as all used the same three Ublox GPS chips. The primary differences were in the interfaces (i.e., i2c vs CAN).

For the purpose of analysis, we consolidated the trade space to three basic options:

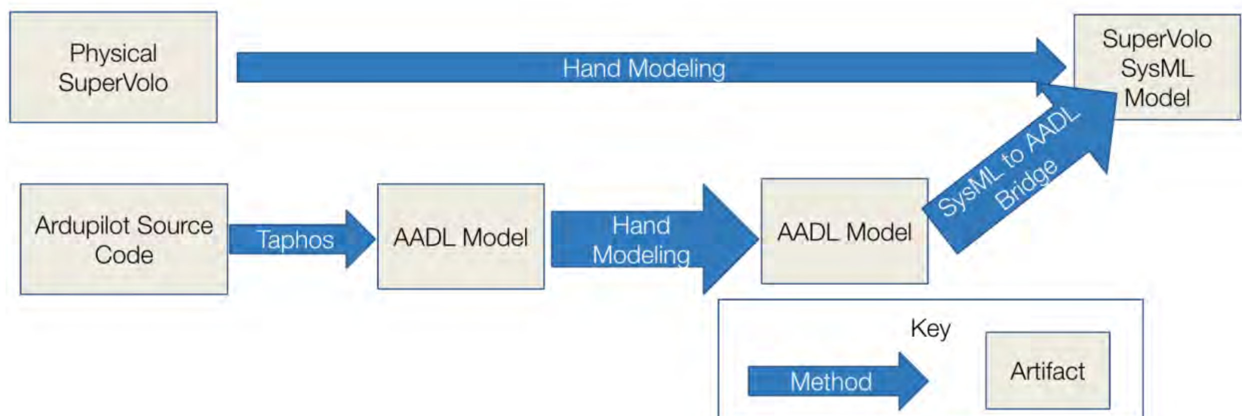
- Cheap/Simple: another i2c M8N module
- More advanced, yet still simple: M9 module with CAN
- Most advanced: RTK solution over i2c/CAN

To decide between these, we decided to focus our measurement (**Frame → Measurement**) on the functional change impact (to what degree does the replacement affect system behavior/capability) and the infrastructure change impact (to what degree does the replacement require changes to other system elements like software?).

Model: SuperVolo and GPS Design Artifacts

After framing our need (replace the SuperVolo GPS with a different model that maintains functionality), we modeled the SuperVolo to establish a foundation of understanding of its architecture, including its physical architecture, its hardware components, and its software architecture. We use this model in the analysis phase to assess the impact of various GPS replacement options.

We began by developing a model of the SuperVolo with sufficient detail to perform the needed analysis and examine the tradeoffs of different GPS options. We created a SysML V1 model (**Model → Architecture Models**) of the aircraft including hardware and software elements, focusing on those that have some relation to the GPS. We created models for some of the hardware components, including the CPU and the GPS, first in AADL (with help from Taphos) and then translated to SysML using the AADL / SysML bridge tool for inclusion in the SysML v1 model (see Figure 5.1).



This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Figure 5.1 Architecture Model Workflow

We use SysML to specify both our Architecture (**Model -> Architecture**) and our Domain model (**Model → Domain Models**). We use the `Documentation` field of each SysML element to provide reference information, forming the glossary for the SuperVolo and its ecosystem (an example shows in the Block Definition Diagram (BDD) in Figure 5.2).

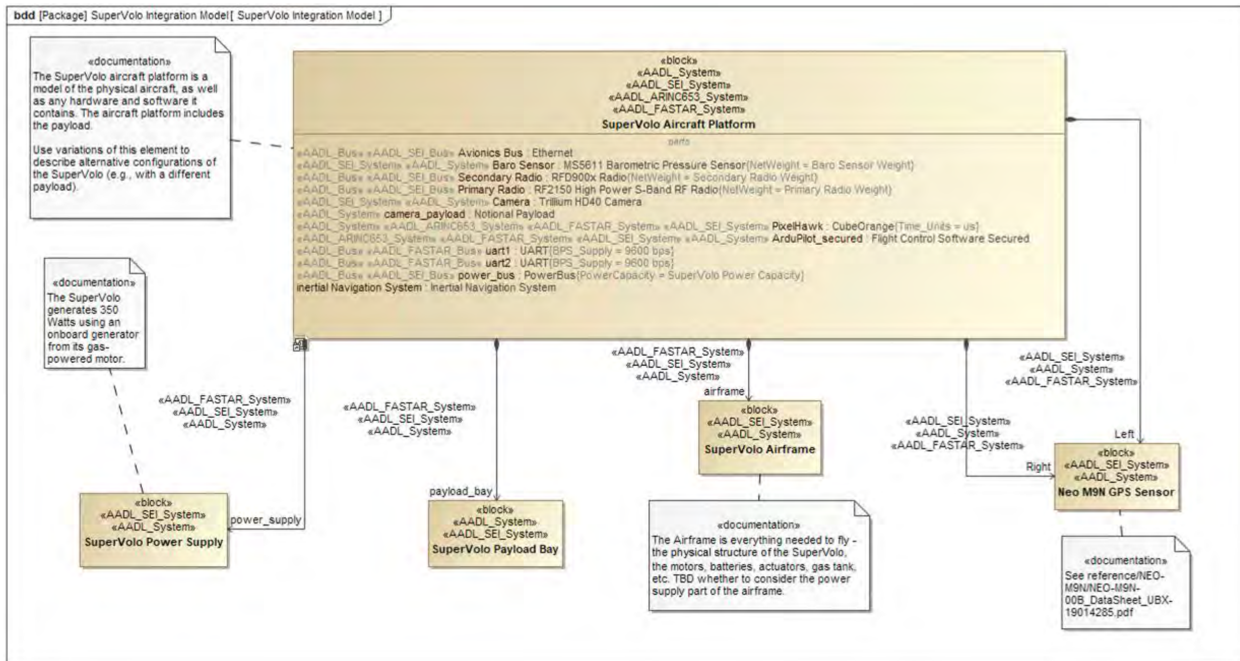


Figure 5.2 Block Definition Diagram (BDD) showing elements of the SuperVolo architecture and their documentation.

The SuperVolo is based on the ArduPilot architecture and many of the components are Commercial Off The Shelf (COTS), so we used online manuals and specification sheets to populate model properties (**Model → Artifacts**). We also had access to the source code for the software for this step and used Taphos to generate AADL models (**Model → Domain Specific Models**) of the Ardupilot software from the source code.

Taphos (who you met earlier) is a tool for *lifting* a software model (**Model → Software Models**) from compiled bytecode. Taphos works on LLVM bitcode and other intermediate compilation artifacts. Because Taphos operates on the compiled code, many of the language nuances that can make change impact analysis difficult are simplified. For example, macros are a common language feature in C or C++ source code, making automated source code analysis difficult. Macros are resolved in LLVM bitcode, reducing the risk of macro-induced ambiguities. Listing

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

5.1 shows taphos commands for lifting AADL models from LLVM, and Listing 5.2 shows an excerpt of such a model.

Listing 5.1 Taphos commands for loading the LLVM bitcode for Arduplane and generating AADL models.

```
load bitcode-sitl-plane.bc
analyze bitcode-sitl-plane.bc
write bitcode-sitl-plane.bc > AADL ./bitcode-sitl-plane.aadl
```

Listing 5.2 Snippet of AADL generated by Taphos

```
-- synthesized declaration of abstract class_AC_AttitudeControl_Multi
  abstract class_AC_AttitudeControl_Multi
  end class_AC_AttitudeControl_Multi;

  -- synthesized declaration of abstract class_AP_MotorsMulticopter
  abstract class_AP_MotorsMulticopter
  end class_AP_MotorsMulticopter;

  subprogram AC_AttitudeControl_Multi
    features
      arg_0 : in out parameter class_AC_AttitudeControl_Multi;
      arg_1 : in out parameter class_AP_AHRS_View;
      arg_2 : in out parameter AP_Vehicle_MultiCopter;
      arg_3 : in out parameter class_AP_MotorsMulticopter;
      arg_4 : in parameter Float_32;
    properties
      Source_Name => "AC_AttitudeControl_Multi::AC_AttitudeControl_Multi(AP_AHRS_View&,
AP_Vehicle::MultiCopter const&, AP_MotorsMulticopter&, float)";
      Source_Text => ("libraries/AC_AttitudeControl/AC_AttitudeControl_Multi.cpp");
  end AC_AttitudeControl_Multi;

  subprogram AC_AttitudeControl_Multi_AC_AttitudeControl_Multi
    features
      arg_0 : in out parameter class_AC_AttitudeControl_Multi;
    properties
      Source_Name => "AC_AttitudeControl_Multi::~AC_AttitudeControl_Multi()";
      Source_Text => ("libraries/AC_AttitudeControl/AC_AttitudeControl_Multi.h");
  end AC_AttitudeControl_Multi_AC_AttitudeControl_Multi;
```

During the modeling steps we discovered that, although the aircraft is based on the open source ArduPilot architecture, it includes custom boards and cables. With no documentation for these components, we modeled them as black boxes (elements where the content is undefined/unknown) and the interfaces were inferred from those of connecting components. For example, if a custom board was connected to a COTS GPS, we used the documentation for the GPS interface to infer the interface of the custom board. We used these models to communicate about the architecture of the SuperVolo amongst our team and with external stakeholders.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Creating AADL Models

As discussed above, we generated some AADL models using Taphos to extract models from software. AADL is a domain specific language that supports analysis of the tradeoffs of component level changes. To use AADL for analysis of cyberphysical system change impacts, we also needed to generate AADL describing the physical components of the SuperVolo. We used the Galois CAMET SysML to AADL Bridge to translate the architecture of the main hardware components and their interconnections from the SysML model to AADL (**Model → Domain Specific Models**). The SysML BDD above shows several components annotated with AADL *stereotypes*. In SysML, a model element can be annotated with a stereotype to provide additional semantically precise information. The AADL stereotypes allow the SysML to AADL bridge to generate AADL (see Figure 5.3).

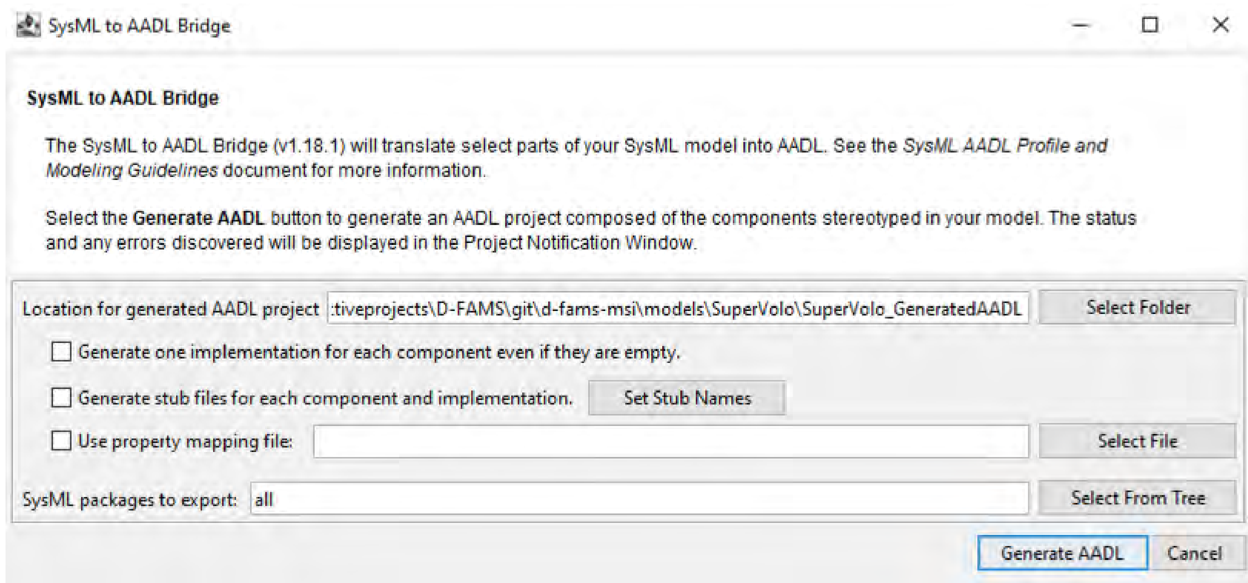


Figure 5.3 AADL Generation Prompt from MagicDraw/Cameo SysML

With both software and hardware AADL models generated, we were able to connect them together (using AADL relationships like *bindings*) to do analysis on things like power usage, signals passing between components, and cable connections between components.

Tutorial: Running Taphos

This tutorial describes how to launch Taphos and use it to generate AADL models.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Build Environment

We wrote this tutorial using Ubuntu Linux version 22.04 LTS and Taphos 0.7.1.0. You must have `sudo` access and `docker` installed.

Get and Compile Ardupilot Source

We learned from the manufacturer that the SuperVolo uses a public *fork* of the Ardupilot project. We modified the build process for Ardupilot to generate LLVM bitcode (a `.bc` file). The workflow for modifying a build process to generate LLVM varies significantly based on the toolchain and is out of scope for this book.

Install Taphos

We distribute Taphos as a docker image. Load it and confirm it is available to docker, as shown in Listing 5.3.

Listing 5.3 Steps for Loading Taphos via Docker

```
// Load the Taphos docker container
sudo docker load < taphos-docker-latest.tgz
// Confirm that it was added to the list of images
sudo docker image list
// Confirm that it runs
sudo docker run -it taphos:0.7.1.0 taphos-tui
```

The last line launches the Taphos Textual User Interface (TUI) which looks like this:

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

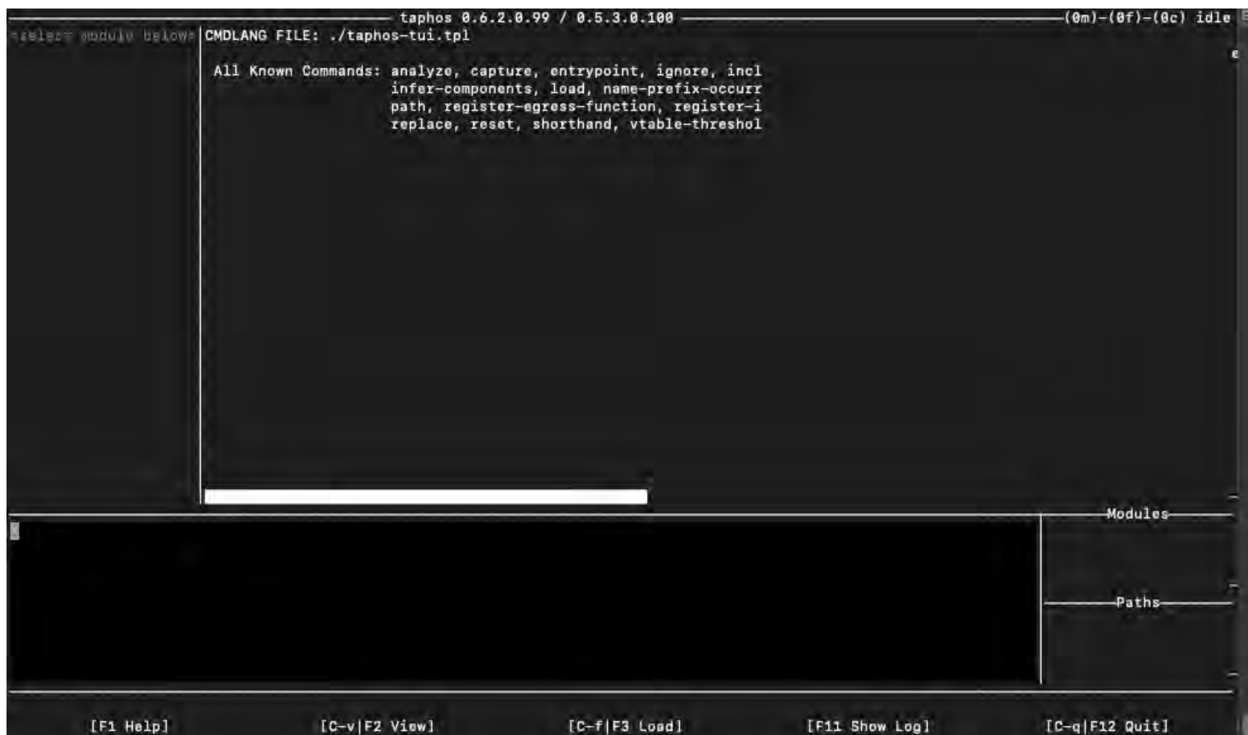


Figure 5.4 Screenshot of Taphos User Interface

The basic command to launch Taphos doesn't actually set up a working directory with a `.bc` file for Taphos to analyze, so hit `F12` to close Taphos, then launch it again, specifying a working directory with a `.bc` file (see Figure 5.4).

Listing 5.4 Launching Taphos via Docker

```
docker run -it -v $(pwd):/taphos-demo -w /taphos-demo taphos:0.7.1.0 taphos-tui
```

Next, hit `F3` to launch the load file dialog (see Figure 5.5). Select the `.bc` file to analyze. In this case our file is called `bitcode-sitl-plane.bc`, the naming is an artifact of the Ardupilot build configuration options.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

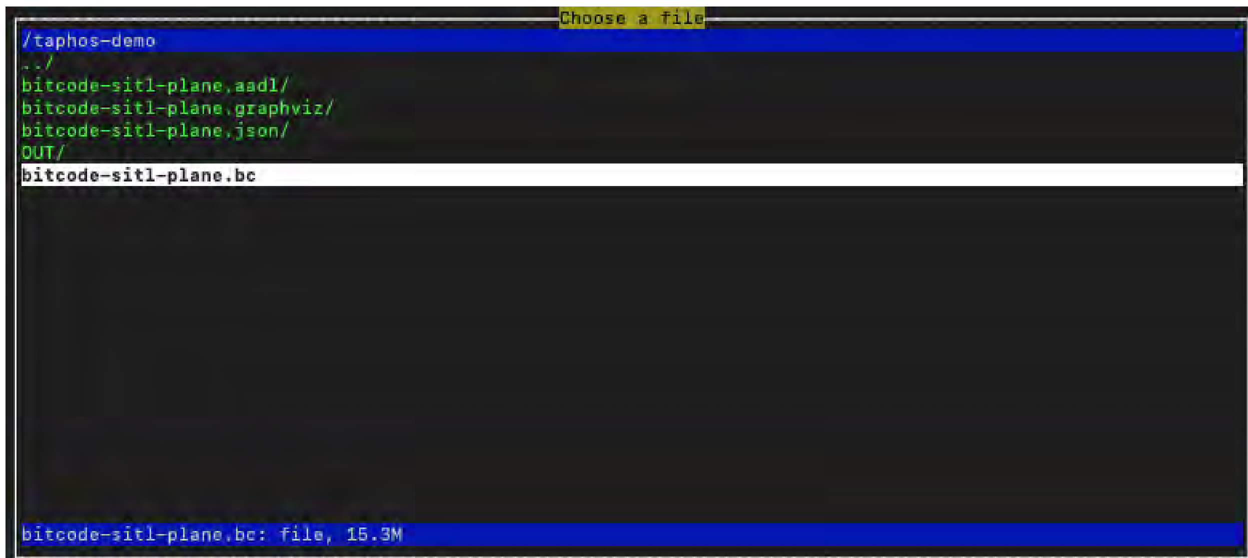


Figure 5.5 Taphos Input File Selection

Hit `enter`. This will create a line of Taphos command language in the command area:

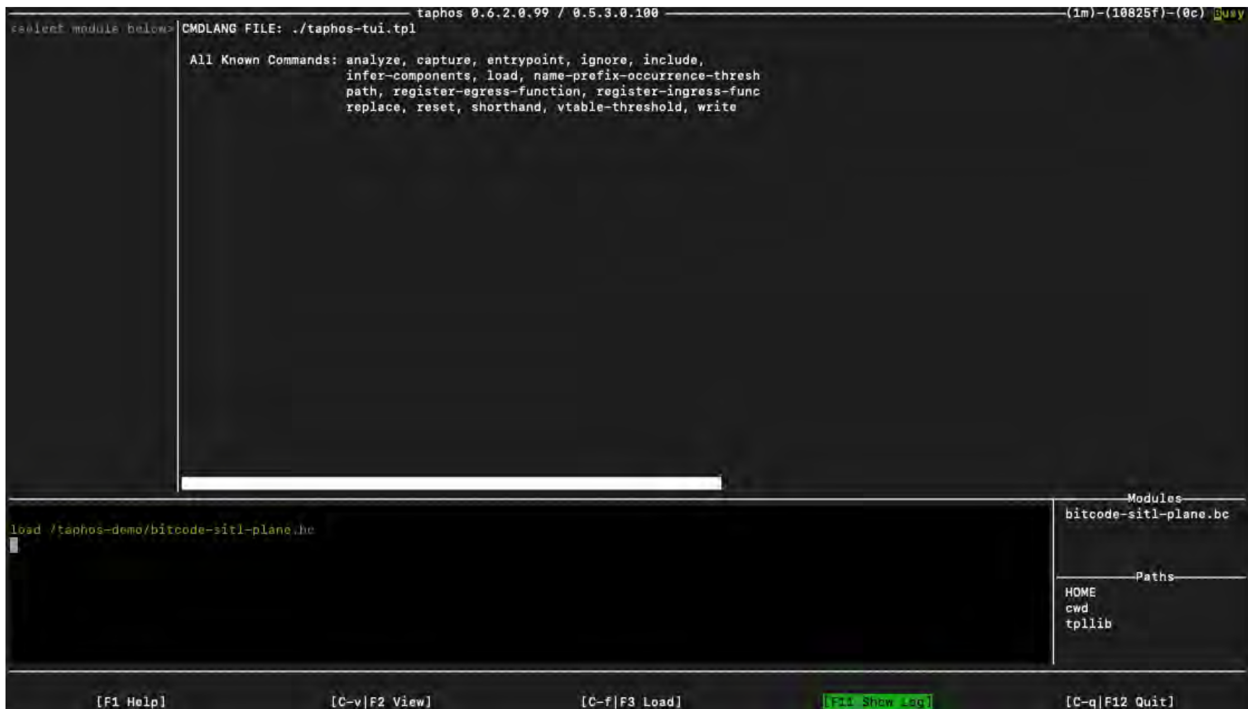


Figure 5.6 Taphos with a Module Loaded

Hit `F5` to run the TPL. Notice the `busy` indicator in the top right corner as Taphos works (see Figure 5.6). If nothing happens, try hitting `F11` to view the log.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Once Taphos has finished working, you'll see a new module in the bottom right corner. Hit Tab to select the Modules area, then down to select the module. Taphos will show you the available information about the module (see Figure 5.7).

Figure 5.7 Taphos showing information about the Ardupilot `bitcode-sitl-plane.bc` bitcode file.

There are a lot of ways we can navigate this information. For example, you can Tab to the function list and browse the information Taphos has captured about each function.

If you want to generate output like AADL models, you can Tab to the command area and add more commands, such as `write bitcode-sitl-plane.bc > AADL ./bitcode-sitl-plane.aadl` to generate AADL models from the `bitcode-sitl-plane.bc` module.

Portable Systems Integration Lab

Although not a "model" in the sense of an abstract mathematical model, we decided to create a portable systems integration lab (PSIL) duplicating the main hardware and software components of the aircraft. We purchased the following components from the SuperVolo architecture and assembled them into a portable case. Like a digital model, the PSIL allows us to conduct low-risk experiments to evaluate the impact of change.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

We purchased the following components from the SuperVolo architecture and assembled them into a portable case. We created an AADL model of the PSIL to document its contents and their relationships (a view of this AADL model is shown in Figure 5.8).

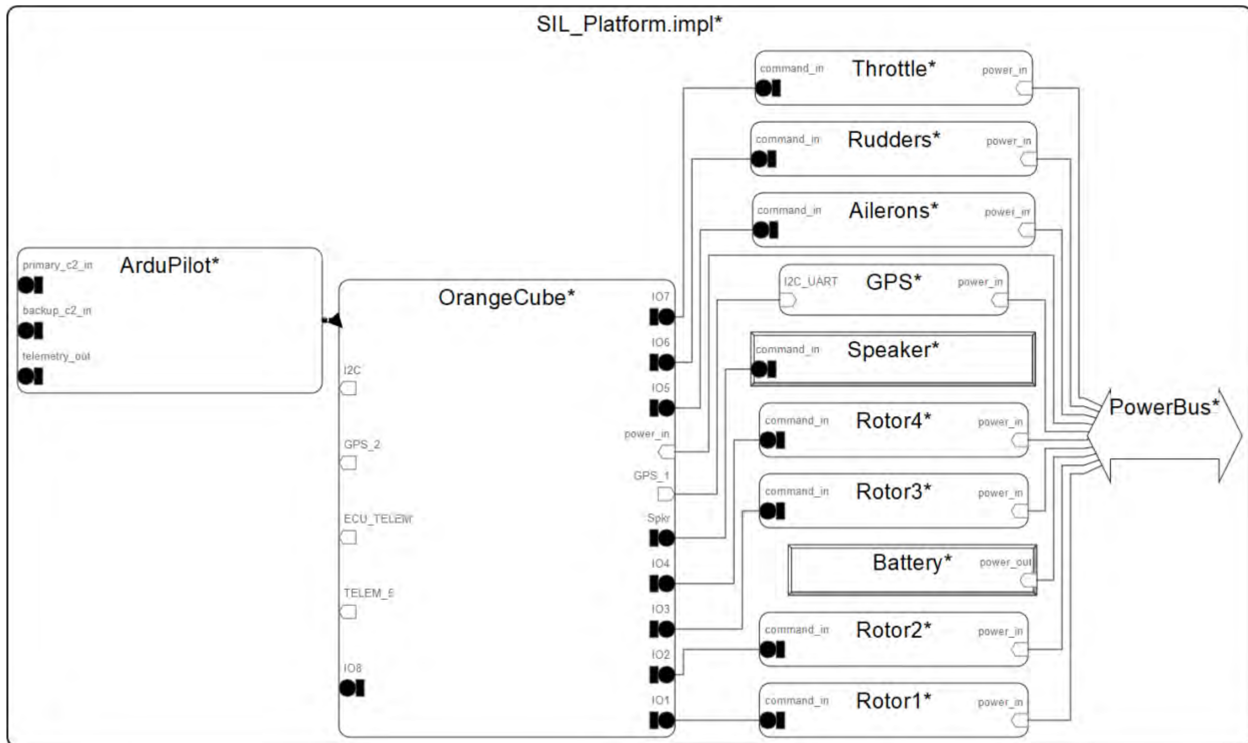


Figure 5.8 AADL Model of PSIL

Processor: Orange Cube
This is the main processor hosting the flight control software. It includes a USB port and a SD card for data logging. The firmware can be updated through the USB port.
Note: The SuperVolo contains one Orange Cube (not plus).

Carrier board: ADSB Orange Cube carrier board.
Includes the support hardware to interface with the Orange Cube. Includes connections for all of the components, power, and buses.
Note: The SuperVolo uses a custom carrier board to interface the Orange Cube with other components.

GPS1: u-Blox Neo-M9N - IST8308 from MRO
This is the same GPS hardware used in the SuperVolo. It includes an I2C port as well as a GPS port and has a cable which connects both ports to the CPU carrier board.
Note: The SuperVolo includes two of this GPS hardware.

GPS2: u-Blox Neo-M9N - IST8308 from 3DR

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

This is the same GPS hardware used in the SuperVolo but from a different vendor. It includes an I2C port as well as a GPS port and has a cable which connects both ports to the carrier board.
Note: The SuperVolo includes two of this GPS hardware.

Power distribution: Cube Orange Power Brick Mini
Provides power from the batteries (or other power source) to the Cube Orange and other components. Also provide some degree of power monitoring and control.
Note: The SuperVolo has a custom power distribution board(s).

Power: 2200 mAh LiPo RC battery
Provides power to the SIL components. Using a battery rather than power supply supports mobile operation for taking the SIL to areas with better GPS reception. Also acquired a compatible charger for the battery.
Note: The SuperVolo includes two 4400 mAh LiPo batteries.

Buzzer: Buzzer provided with the Orange Cube carrier board
The buzzer provides key system status information throughout power up and test as well as during flight. It provides good feedback on the operation of the flight control system. It connects to the spkr port on the carrier board.
Note: The SuperVolo includes an internal speaker for signals. It can be heard outside the aircraft.

Air Surface Servos:
Servos control the flaps on the fixed wings of the SuperVolo. The PSIL includes servos with pointer arms, but not the control surfaces. Their movement shows the operation of the flight control software for fixed wing flight.
Note: The SuperVolo contains 4 flight surface servos.

Rotor Servos:
These full rotation servos show when the SuperVolo electric motors are engaged. Their movement shows the operation of the flight control software for rotor flight.
Note: The SuperVolo includes 4 electric motors and electronic speed controllers.

Software:
The software running on the PSIL is Ardupilot (ArduPlane) version v3.10.0. The code is generated from a custom GitHub branch of ArduPilot. It talks to the Mission Planner software running on a laptop ground station.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Assembly and Integration of the PSIL

Before assembly and integration, we built a model of the PSIL to inform key design decisions. We reused components from the SuperVolo model where appropriate and added additional components as needed. As we designed the PSIL it was clear that the power requirements would be substantially lower than the needs of the SuperVolo. We used the model to select the appropriate battery and power management hardware. We also used the model to determine specific cabling needs based on component interconnections.

The initial PSIL included an upgraded Orange Cube Plus CPU as the specs showed that it was backwards compatible with the Orange Cube (not plus) in the SuperVolo. As we were configuring the software for the PSIL we learned that the SuperVolo uses an older version of the ArduPilot software that was not compatible with the Orange Cube Plus. This necessitated an upgrade (downgrade) of the PSIL CPU to an Orange Cube.

We installed the same version of ArduPilot as is used on the SuperVolo and initially encountered numerous warning messages that prevented the system from being armed for flight. Some of these checks looked for hardware that was not included in the PSIL or tested the backup radio connection, also not included in the PSIL. Using the Mission Planner software, we changed the configuration of ArduPilot on the PSIL to do a minimum of checks before arming and this eliminated the error messages. After this we were able to arm and “fly” the PSIL by moving it around. The Mission Planner display tracked the movements in both the heads up display as well on the GPS derived map location. The control surface servos in the PSIL responded appropriately to the movements of the PSIL.

Analyze: Implementation Options and Change Impacts

One of the first topics we explored when considering a GPS swap was whether to pursue new GPS solutions that would require software changes. Using Taphos, we can evaluate the code base to find code that might be affected by a change to the GPS software (see the tutorial earlier in Part 5). The size of a function’s call graph, both its callers and callees, is a measurement of the risk of change impact for any change to that function.

We investigated four alternative options for GPS hardware using the PSIL. These include the following: an older M8N Ublox GPS, an M9N Ublox GPS identical to the hardware in the SuperVolo, an M9N UBlox GPS from an alternative vendor source, and a DroneCan M9N GPS. All of the GPS options used the same telemetry connections to the CPU other than the DroneCan GPS, which used a CanBus connection. The older M8N required a custom cable to make the connection.

Software Change Impact Analysis

Analyze → Change Impact Analysis

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

We can use several approaches to understand how the GPS is used in the current SuperVolo architecture. We started our evaluation by assuming that a change to the GPS sensor *may* involve changes to some software system elements. We used two methods to evaluate software change impact for a notional GPS change. Note that we did this before writing code, to help us decide how to approach this change.

We focused our analysis on the `AP_GPS` package and used Lattix and Taphos to evaluate the possible impact.

Lattix Analysis

We generated a Design Structure Matrix (DSM) of software dependencies in Ardupilot using Lattix. With Lattix, we can see that `APMover2`, `AntennaTracker`, `Arducopter`, `Arduplane`, `Ardusub`, and `Tools` all use `AP_GPS`.

Figure 5.9 Lattix Analysis of GPS Software

There are also several library packages, (e.g., AHRS) that use the AP_GPS package. Lattix provides a detailed view that lets us see what specific source files (e.g., `GCS_Mavlink.cpp`) use the `AP_GPS` package and how.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

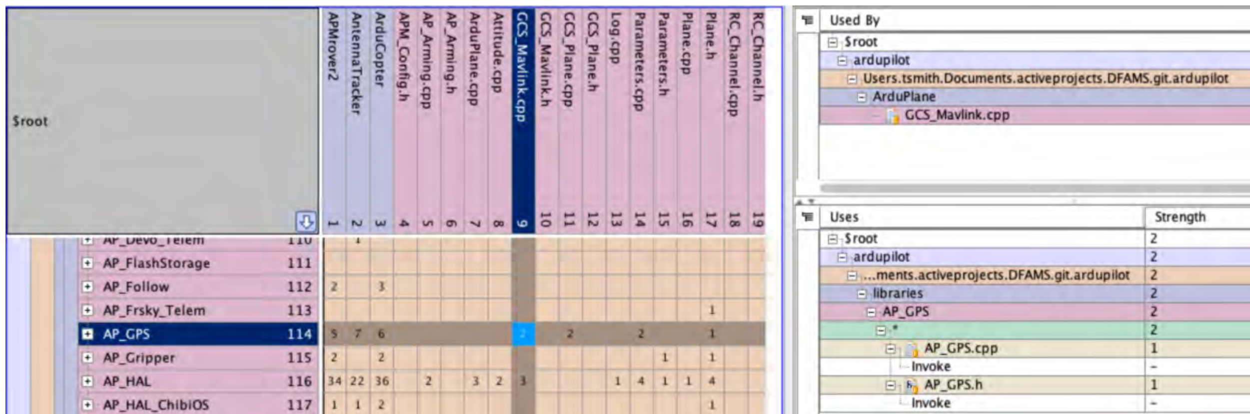


Figure 5.10 Detailed View of AP_GPS Dependencies in Lattix

This view generates a picture of the potential impact of changing the GPS software. In particular, we now have a list of the first tier dependencies of `AP_GPS` as reported by Lattix. If we apply this approach recursively, we can build an exhaustive list, shown in Table 5.2.

Table 5.2 First Tier Dependencies - Lattix Detection

Name
APMover2
Antenna Tracker
Arducopter
Arduplane
ArduSub
Tools
libraries/APM_Control
libraries/AP_ADSB
libraries/AP_AHRS
libraries/AP_AdvancedFailsafe
libraries/AP_Airspeed
libraries/AP_Arming
libraries/AP_Camera

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Name
libraries/AP_Devo_Telem
libraries/AP_Frsky_Telem
libraries/AP_GPS
libraries/AP_Landing
libraries/AP_Logger
libraries/AP_Mission
libraries/AP_Module
libraries/AP_Motors
libraries/AP_Mount
libraries/AP_NavEKF2
libraries/AP_NavEKF3
libraries/AP_Notify
libraries/AP_OSD
libraries/AP_OpticalFlow
libraries/AP_Scripting
libraries/AP_SpartRTL
libraries/GCS_Mavlink
libraries/RC_Channel

As we noted earlier, software usage is only one type of dependency, and this is only the first tier of such dependencies. It is possible (common, in fact!) for an error introduced in one software component to introduce an observed defect several tiers of dependency away!

Let's follow one of these to see what depends on it! `GCS_Mavlink` depends on `AP_GPS`. The items in Table 5.3 depend on GCS_Mavlink.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Table 5.3 Components Depending on GCS_Mavlink

Name
APMover2
AntennaTracker
ArduCopter
Arduplane
ArduSub
Tools
libraries/AC_AutoTune
libraries/AC_Fence
libraries/AC_PreLand
libraries/AP_ADSB
libraries/AP_AHRS
libraries/AP_AccelCal
libraries/AP_AsdvancedFailsafe
libraries/AP_Airspeed
libraries/AP_Arming
libraries/AP_Avoidance
libraries/AP_Baro
libraries/AP_BattMonitor
libraries/AP_BoardConfig
libraries/AP_Button
libraries/AP_Camera
libraries/AP_Common
libraries/AP_Compass
And more!

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

If we follow this thread, we get a sequence of recursive dependency lists, which will eventually complete either with the *entire* code base of Ardupilot, or a subset of the code base.

Taphos Analysis

Taphos is another tool we used to analyze the GPS package. Using Taphos we can search for functions and learn about their degree of connectivity to the overall architecture in a more automated fashion than we can with Lattix. For example, we can search for functions that use the term `GPS`, as shown in Listing 5.5.

Listing 5.5 Searching functions with Taphos

```
tsmith@tyler-ubuntu:~/Documents/activeprojects/example$ docker run -it -v $(pwd):/example -w
/example taphos:0.6.1.1 taphos_json_info.py arduplane.json functions | grep GPS
    groundspeed               gps                   AP_GPS::get_singleton
       Vector2<float>::Vector2()
    AP_GPS::status() const    GCS_MAVLINK::capabilities     GCS_MAVLINK::in_hil_mode
       vtol_state
    get_prot_for_mission_type system_status         AP_GPS::status(unsigned char) const
       handle_command_accelcal_vehicle_pos
    Write_Format_Units        AP_Logger_Backend::Write_Parameter(char const*, float)
       AP_GPS::time_week_ms(unsigned char) const    AP_GPS::time_week(unsigned char) const
    Write_CameraInfo          AP_GPS::time_week_ms() const  AP_GPS::time_week() const
       Write_Camera
    AP_Logger_Backend::Write_Mode     airspeed_estimate_true AP_GPS::ground_course_cd() const
       radians(float) [clone .366]
    AP_GPS::ground_speed() const      AP_GPS::ground_speed(unsigned char) const
       AP_GPS::ground_course_cd(unsigned char) const       AP_GPS::ground_course(unsigned
char) const
    reserve                   commit                AP_GPS::location() const
       get_soft_armed
    get_angle_pitch_p         AP_GPS::location(unsigned char) const Compass::get_primary
       Compass::get_field(unsigned char) const
```

We use the Taphos radius tool to evaluate the degree of connectivity. For example, the function `AP_GPS::get_singleton` has 10 layer callers (that is, functions that call a function that calls `AP_GPS::get_singleton`) as shown in Listing 5.6.

Listing 5.6 A search with radius 2 for callers and callees of the `AP_GPS::get_singleton` function.

```
tsmith@tyler-ubuntu:~/Documents/activeprojects/example$ docker run -it -v $(pwd):/example -w
/example taphos:0.6.1.1 taphos_json_info.py arduplane.json radius AP_GPS::get_singleton 2
Call Radius (2) for AP_GPS::get_singleton
    Calling layer 2 :
```

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

```
        update_sensor_status_flags    GCS_MAVLINK::try_send_message
       GCS_MAVLINK::vfr_hud_airspeed handle_common_message
        set_home_to_current_location  GCS_MAVLINK_Plane::handle_command_long_packet
       AP_AHRS::groundspeed_vector    handle_serial_control
        Write_CameraInfo      active_EKF_type
        ^--- 10 layer2 callers
    Calling layer 1 :
        gps
        ^--- 1 layer1 callers
```

Tutorial: Inspecting Dependencies with Taphos

Earlier we walked through the process of inspecting a bitcode module using Taphos. Now we extend that tutorial to describe how we can use Taphos to inspect software dependencies. Add a couple of additional options to the command used to launch Taphos: `TAPHOS_OUTPUT_SOLO_FUNCTIONS` and `TAPHOS_OUTPUT_SINGLE_PACKAGE`. These options tell Taphos to include functions defined outside of recognized components and tell Taphos to merge its output into a single package.

Listing 5.7 Launching Taphos with Environment Variables set for JSON Generation

```
docker run -it -v $(pwd):/taphos-demo -w /taphos-demo \
 -e TAPHOS_OUTPUT_SOLO_FUNCTIONS=1 -e TAPHOS_OUTPUT_SINGLE_PACKAGE=1 \
 taphos:0.7.1.0 taphos-tui
```

Our commands to Taphos will load the module, analyze it, and generate `JSON`

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Figure 5.11 Taphos Showing Commands for JSON Generation

Type this in the command area (as shown in Figure 5.11 and Listing 5.8), then press F5

Listing 5.8 Commands to Generation JSON with Taphos

```
load bitcode-sitl-plane.bc
analyze bitcode-sitl-plane.bc
write bitcode-sitl-plane.bc > json ./bitcode-sitl-plane.json
```

This creates a new file in your working directory called `bitcode-sitl-plane.json/Llvm_Link.json`. Hit `F12` to close Taphos. The generated `.json` file includes function and global variable relationship data parsed by Taphos.

We use the `taphos_json_info.py` tool to inspect this `.json` file. Launch the `taphos_json_info.py` file with docker, as shown in Listing 5.9.

Listing 5.9 Using Taphos json_info.py to List Functions found by Taphos

```
cd bitcode-sitl-plane.json
docker run -it -v $(pwd):/taphos-example -w /taphos-example taphos:0.7.1.0
taphos_json_info.py  Llvm_Link.json functions
```

The result is a listing of all functions Taphos found in the bitcode:

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Listing 5.9 Excerpt of Function List from Taphos

```
Functions :
    use_leaky_i             set_inverted_flight    get_roll_trim_rad      radians(float)
    constrain_float         ang_vel_limit          AC_P::kP()             euler_accel_limit
    input_shaping_angle     get_slew_yaw_rads      asin                   AC_PID::reset_filter
    input_quaternion        shift_ef_yaw_target    inertial_frame_reset   accel_limiting
    degrees(float)          stopping_point         AC_PID::kD()           AC_PID::kP()
    max_rate_step_bf_yaw    get_angle_pitch_p      get_angle_roll_p       get_angle_yaw_p
    AC_PID::ff()            AC_PID::kI()           get_rate_roll_pid      get_rate_pitch_pid
    get_rate_yaw_pid        rate_controller_run    set_throttle_out       get_throttle_boosted
    set_throttle            get_throttle_avg_max   set_throttle_avg_max   set_roll
    set_roll_ff             set_pitch              set_pitch_ff           set_yaw
    set_yaw_ff              AC_AttitudeControl     AP_Param::AP_Param()   AC_P
    AC_AutoTune::run        AC_AutoTune::start     position_ok            get_interlock
…
```

To find out what functions use, and are used by, each of these functions, we can use the `call-radius` capability of `taphos_json_info.py`. For example, if we want to see what functions uses `set_pitch`, we can use the command shown in Listing 5.10.

Listing 5.10 Taphos Command to Search for Calls to and From set_pitch at Radius 1

```
docker run -it -v $(pwd):/taphos-example -w /taphos-example taphos:0.7.1.0
taphos_json_info.py  Llvm_Link.json radius set_pitch 1
```

Searching for callers or callees is a common feature of IDEs. However, Taphos can do *recursive* search, which is much more useful for evaluating change impact. We can set the depth of the recursive search in the final parameter. Here we've increased it to two, as shown in Listing 5.11 and Listing 5.12.

Listing 5.11 Taphos Command to Search for Calls to and From set_pitch at Radius 2

```
docker run -it -v $(pwd):/taphos-example -w /taphos-example taphos:0.7.1.0
taphos_json_info.py  Llvm_Link.json radius set_pitch 2
```

Listing 5.12 Output from Taphos Command to Search for Calls to and From set_pitch at Radius 2

```
Call Radius (2) for set_pitch
    Called from layer 2 :

        ^--- 0 layer2 callers
  ↓
    Called from layer 1 :
        rate_controller_run
```

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

```
      ^--- 1 layer1 caller
↓
  ====> set_pitch ====>
↓
  Calls to layer 1 :

      ^--- 0 layer1 callees
↓
  Calls to layer 2 :

      ^--- 0 layer2 callees
```

In this case we can see that `set_pitch` is a leaf function and is only called by one other function, which in turn is not called from anything Taphos could find (indirect function calls, such as functions invoked as part of scheduled dispatches, may not be found by Taphos). Other functions, such as `AP_GPS_speed_accuracy` are more highly connected, as shown in Listing 5.13 and Listing 5.14:

Listing 5.13 Taphos Command to Search for Calls to and From AP_GPS_speed_accuracy at radius 3

```
docker run -it -v $(pwd):/taphos-example -w /taphos-example taphos:0.7.1.0
taphos_json_info.py  Llvm_Link.json radius AP_GPS_speed_accuracy 3
```

In this case we see that nothing in Taphos's purview calls `AP_GPS_speed_accuracy` but that it uses *many* other functions.

Listing 5.14 Output from Taphos Command to Search for Calls to and From AP_GPS_speed_accuracy at radius 3

```
Call Radius (3) for AP_GPS_speed_accuracy
    Called from layer 3 :

        ^--- 0 layer3 callers
  ↓
    Called from layer 2 :

        ^--- 0 layer2 callers
  ↓
    Called from layer 1 :

        ^--- 0 layer1 callers
  ↓
    ====> AP_GPS_speed_accuracy ====>
  ↓
    Calls to layer 1 :
        AP_GPS::num_sensors   lua_pushnil           lua_pushnumber        luaL_argerror
        luaL_checkinteger     binding_argcheck
```

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

```
        AP_GPS::get_singleton
        AP_GPS::speed_accuracy(unsigned char, float&) const
        decltype ((({parm#1}>{parm#2})?{parm#1} : {parm#2}) MAX<int, int>(int const&, int
const&) [clone .9120]
        decltype ((({parm#1}<{parm#2})?{parm#1} : {parm#2}) MIN<unsigned char, int>(unsigned
char const&, int const&) [clone .9121]
        ^--- 10 layer1 callees
  ↓
    Calls to layer 2 :
        lua_tolstring          pushglobalfuncname     luaL_error             lua_getstack
        lua_getinfo            lua_tointegerx         interror               lua_gettop
        luaL_argerror


        ^--- 9 layer2 callees
  ↓
    Calls to layer 3 :
        index2addr             luaC_step              luaO_tostring          lua_gettop
        lua_settop             lua_rotate             lua_copy               lua_tolstring
        lua_pushstring         lua_getfield           findfield              lua_getinfo
        lua_pushvfstring       lua_error              lua_concat             luaL_where
        swapextra              auxgetinfo             collectvalidlines      index2addr
        luaV_tointeger         lua_isnumber           luaL_argerror          tag_error
        lua_tolstring          pushglobalfuncname     luaL_error             lua_getstack
        lua_getinfo


        ^--- 29 layer3 callees
```

Other functions, such as `AP_GPS::ground_speed() const,` are called by many functions, but call few, as shown in Listing 5.15 and Listing 5.16.

Listing 5.15 Taphos Command to Search for Calls to and From ground_speed at radius 3

```
docker run -it -v $(pwd):/taphos-example -w /taphos-example taphos:0.7.1.0
taphos_json_info.py  Llvm_Link.json radius "AP_GPS::ground_speed() const" 3
```

Listing 5.16 Output from Taphos Command to Search for Calls to and From ground_speed at radius 3

```
Call Radius (3) for AP_GPS::ground_speed() const
    Called from layer 3 :
        AP_AHRS_DCM::update    drift_correction       update_DCM             AP_Airspeed::update
        AP_Frsky_Telem::loop   AP_LTM_Telem::tick     update_alt             update_GPS_50Hz
        get_speed_scaler       stabilize_training     stabilize_acro         Plane::stabilize
        send_wind              calcMagHeadingInnov    update_home            geofence_check
        update_is_flying_5Hz   ModeRTL::update        ModeRTL::_enter        navigate
        calc_airspeed_errors   stopping_distance      forward_throttle_pct   read_airspeed
        set_servos
        AP_AHRS_NavEKF::update                        AP_InertialNav_NavEKF::update
        Log_Write_Control_Tuning                      rangefinder_height_update
```

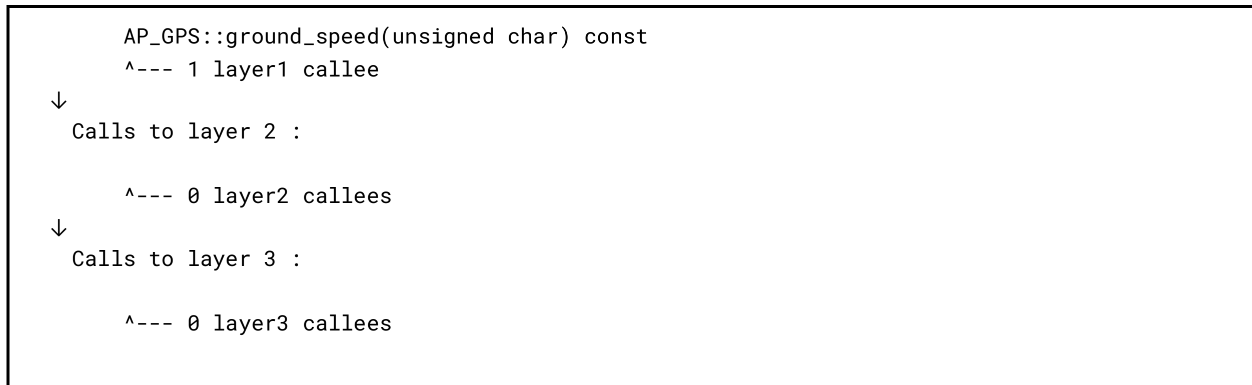
This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

```
    Plane::verify_command                       verify_landing_vtol_spiral_approach
    verify_command_callback                     geofence_prearm_check
    calc_gndspeed_undershoot                    vtol_position_controller
    update_throttle_thr_mix                     set_servos_controlled

    ^--- 37 layer3 callers
↓
  Called from layer 2 :
    AP_AHRS_DCM::update    drift_correction      drift_correction_yaw   get_accel_ef_blended
    airspeed_estimate      get_velocity_NED      update_SITL            update_EKF2
    update_EKF3            get_mag_field_NED     get_vert_pos_rate      AP_DEVO_Telem::tick
    send_D                 send_SPort            AP_Hott_Telem::loop    generate_LTM
    update_navigation      stabilize_yaw         update_is_flying_5Hz   ModeAuto::update
    ModeTakeoff::update
    AP_AHRS_NavEKF::check_lane_switch           AP_AHRS_NavEKF::get_gyro
    AP_AHRS_NavEKF::get_gyro_drift              get_rotation_body_to_ned
    AP_AHRS_NavEKF::get_position                AP_AHRS_NavEKF::get_hagl
    AP_AHRS_NavEKF::wind_estimate               AP_AHRS_NavEKF::groundspeed_vector
    AP_AHRS_NavEKF::get_mag_field_correction    AP_AHRS_NavEKF::use_compass
    AP_AHRS_NavEKF::get_secondary_attitude      AP_AHRS_NavEKF::get_secondary_quaternion
    AP_AHRS_NavEKF::get_secondary_position      AP_AHRS_NavEKF::set_origin
    AP_AHRS_NavEKF::have_inertial_nav           AP_AHRS_NavEKF::healthy
    is_ext_nav_used_for_yaw                     check_sensor_failures
    AP_Landing::verify_land                     Plane::verify_command
    Plane::suppress_throttle                    set_servos_controlled
    AP_AHRS_NavEKF::get_accel_ef(unsigned char) const
    AP_AHRS_NavEKF::get_relative_position_NED_origin
    AP_AHRS_NavEKF::get_relative_position_NE_origin
    AP_AHRS_NavEKF::get_relative_position_D_origin
    AP_AHRS_NavEKF::getCorrectedDeltaVelocityNED
    ^--- 48 layer2 callers
↓
  Called from layer 1 :
    drift_correction       drift_correction_yaw  active_EKF_type        send_frames
    calc_gps_position      send_GPS              send_Gframe            calc_nav_yaw_ground
    Plane::stabilize       verify_takeoff        ground_speed_cm        update_cruise
    auto_takeoff_check     takeoff_calc_pitch
    AP_AHRS::airspeed_estimate                  AP_AHRS::groundspeed_vector
    AP_AHRS_DCM::get_position                   AP_AHRS_DCM::airspeed_estimate
    AP_AHRS_DCM::use_compass                    check_sensor_ahrs_wind_max_failures
    type_slope_verify_land                      airspeed_ratio_update
    GCS_MAVLINK::vfr_hud_airspeed               Plane::suppress_throttle

    ^--- 24 layer1 callers
↓
  ====> AP_GPS::ground_speed() const ====>
↓
  Calls to layer 1 :
```

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025



```
      AP_GPS::ground_speed(unsigned char) const
      ^--- 1 layer1 callee
↓
  Calls to layer 2 :

      ^--- 0 layer2 callees
↓
  Calls to layer 3 :

      ^--- 0 layer3 callees
```

Commentary on Caller vs Callee Functions

In the tutorial above we considered two functions related to GPS: `AP_GPS_speed_accuracy` , which *calls* many other functions and `AP_GPS::ground_speed() const`, which is *called by* many other functions. Both are examples of highly-interconnected functions that present a significant risk of change impact.

You might find it odd that we consider both the caller and callees of a function when assessing change impact risk. In a purely functional language, we could consider the change impacts to functions called by a function in question. However, in a language like `C` or `C++`, calling a function can have *side effects*. This means that a change to the behavior of a function like `AP_GPS_speed_accuracy` could propagate to any function it calls. Languages like `Ada` that are more restrictive with regard to side-effects significantly reduce this risk, and purely functional languages, though less common in avionics, can eliminate it altogether.

Conclusion

Analysis both with Lattix and Taphos shows that the GPS software is *highly* interconnected with other elements of the Ardupilot software. With further effort we could quantify this connectivity. For example, we could use the Taphos `radius` to count all of the callers and callees of functions as part of a GPS software change to a given radius (e.g., radius 3).

Contain: Refactor, Software Changes, and Results

The potential change impact of any alternation of the GPS software was very high - as shown in the tutorial above, a single function (`AP_GPS::ground_speed()`) had 37 callers at a radius of 3. In this case, the GPS devices we considered all used a common standard specification (**Contain → Break Dependencies with Interoperability Standards**), so we opted to leverage the standard and pursue GPS alternatives that did not include software change.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

We used the PSIL (**Contain → Hardware in the Loop Testing**) to evaluate a couple of specific options. The older M8N integrated into the PSIL fine but did not provide a location update. The Mission Planner reported “No GPS Fix”. This may be due to the very small antenna on the device. The two M9N Ublox GPS devices performed well, providing good GPS location information. When the DroneCan GPS was installed, Mission Planner reported a “No GPS” alert message. Changing the ArduPilot GPS configuration parameter to “DroneCan” did not change the results. This may be due to the older version of ArduPilot not supporting DroneCan. Later versions of this software do support DroneCan but it is not an option for the version of the software currently in the SuperVolo.

We also tested configurations using multiple GPS Devices. With the older M8N GPS in port 1 and the newer M8N in port 2, Mission Planner reported “No Fix” for the first device and then used the location information from device 2. This was the same behavior when the DroneCan GPS was added. When the two M9N devices were installed in ports 1 and ports 2. Mission Planned reported good data from both devices.

Conclusion

In the GPS case study, we hypothesized that a stakeholder required a new GPS unit in the SuperVolo. We generated SysML and AADL models of the SuperVolo (we’ll use these more in Part 6). We used Taphos and Lattix to evaluate the change impact of changes to GPS-related software. We determined that software changes to GPS presented a high risk of change impact propagation to other parts of the architecture, and opted to leverage existing standards to make a low-risk change.

Part 6: Case Study - Implementing a Partitioned Architecture

Frame: Mission Requires new Rapid Sensor Integration

Our first case study (Part 5) focused on a single component and a relatively simple stakeholder requirement (swap out the GPS). For the second case study, we evaluate a much more challenging scenario: stakeholders need a domain separation in the Ardupilot software to improve its cyber resiliency.[50]

> *“Cyber resiliency engineering intends to architect, design, develop, maintain, and sustain the trustworthiness of systems with the capability to anticipate, withstand, recover from, and adapt*

[50] https://csrc.nist.gov/pubs/sp/800/160/v2/r1/final

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

to adverse conditions, stresses, attacks, or compromises that use or are enabled by cyber resources."

NIST Special Publication 800-160, Volume 2 Revision 1 Developing Cyber-Resilient Systems: A Systems Security Engineering Approach

When we first introduced the notion of change impact, we focused on the impact of intentional change made to an architecture. We now expand that notion to include the impact of intentional changes to an architecture by designers, the impact of adverse or unexpected operating conditions, and the impact of malicious actions by adversaries. Domain separation is an architectural approach that can help address all of these types of change impact.

Domain separation is a design approach in which components of an architecture are segregated into *domains*. A domain is a grouping of resources. Components might be grouped in domains based on their criticality, what kind of information they process, or other criteria. For example, domains may be applied based on security policy.[51] Domain separation is accomplished through the isolation of domains from one another by eliminating or managing dependencies between them. For example, a system responsible for handling both classified and unclassified data might isolate these two domains by barring software that handles unclassified data from *depending on* the same hardware as software that handles classified data.

Galois's CAMET Library provides several tools for analyzing system architectures in terms of various domains. We used several of these tools in this case study, including the Multiple Analyses for Domain Separation (MADS), Risk Management Framework (RMF), and FASTAR scheduling tools.

The ArduPilot avionics software on the SuperVolo was developed without domain separation controls or well-defined domain separation requirements (for example, ArduPilot has no notion of separate levels of security control).[52] This means that there are no mechanisms in place to prevent a single task from, for example, over-utilizing the CPU and starving other tasks.

With this in mind, our exercise of measuring the impact of a rapid sensor integration first focuses on updating the SuperVolo system model to support a potential refactoring of the software architecture. With model updates in place that address the underlying security flaws, we can then conduct a number of design domain analyses that target a particular facet of security controls of the newly integrated sensor.

The design domains of interest are criticality and security sensitivity. The change impact propagation avenues of concern are time partitioning, space partitioning, and component

---

[51] https://csrc.nist.gov/glossary/term/security_domain

[52] The SuperVolo runs a customized version of Ardupilot. Ardupilot is available on github: https://github.com/ArduPilot/ardupilot

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

separation and isolation. As is the case for any complex embedded system, every significant design decision must be balanced against all design domains, since modifications often cut across domains in both predictable and unforeseen ways.

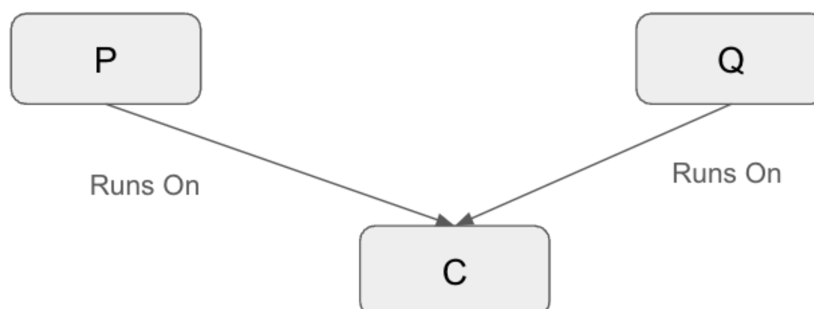


Figure 6.1 P and Q are software threads running on processor core C.

For example, suppose we have two software threads P and Q, both running on a processing core C *(See Figure 6.1)*. Although P and Q do not have *software dependencies* from one to the other on which change impact could propagate, they both are *bound to* the same processing core, and thus impact from change to one might indirectly propagate from P to Q or vice versa. For example, a change that increases the required compute capacity for P might affect Q by reducing the overall available compute capacity.

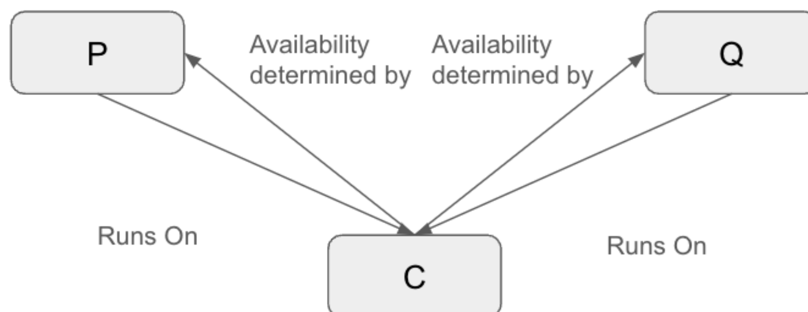


Figure 6.2 If C does not impose schedule constraints on P and Q, they can affect one another by changing the availability of C.

A *shared execution environment* means two software threads that appear unrelated can actually propagate change impact from one to the other. In the same fashion, a software vulnerability in P could expose data from Q if they use the same memory space. Partitioning is a method of splitting the shared resource to prevent change impact from propagating from one domain to another.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

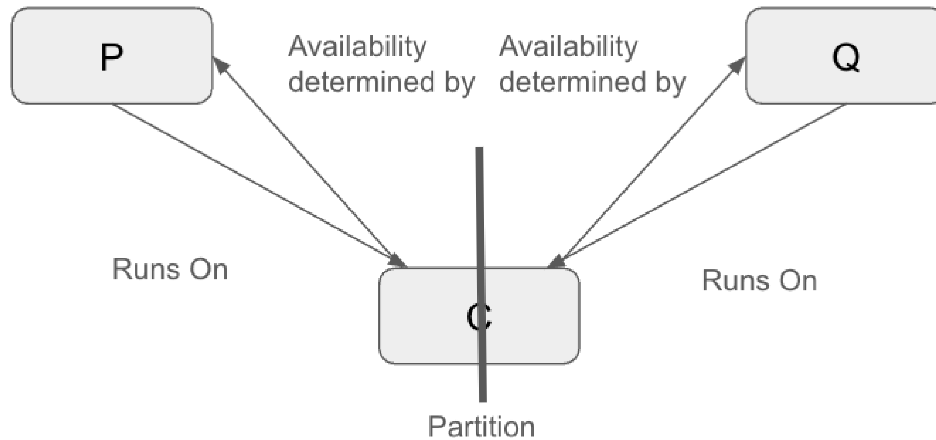


Figure 6.3 A partitioned approach divides the resource in such a way that system elements cannot affect one another, preventing change impact propagation.

We decided to focus on a *time- and space-partitioned* domain separation approach, in which individual domains are separated in both computational time and computational space (recall, we introduced such separation approaches in Part 4). Such separation breaks potential avenues of change impact propagation by eliminating the possibility of interference (intentional or accidental) between domains. For example, software components from one domain are barred from executing on the same CPU as software components from another domain at the same time. ARINC653 is a standard for operating systems that provide such domain separation.[53] Operating systems like the formally-verified Sel4 can also provide such separation.[54] For this case study, we focus on ARINC653, as the use of such a standard further reduces change impact risk.

Model: SuperVolo Design Artifacts

In Part 5 we used models exclusively for *analyzing* the change impact risk associated with a potential software change. For the more ambitious goal of applying a separation architecture, we used (and expanded) models *both* to plan how we might change the architecture and to analyze the impacts of such changes.

To create necessary models to evaluate the impact of applying a separation architecture, we started with Taphos-generated AADL from the SuperVolo/ArduPilot source (as discussed in Part

[53] https://www.sae.org/standards/content/arinc653p0-3/
[54] https://sel4.systems/

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

5).[55] We extended the SysML model to support ARINC 653 scheduling analysis by including additional software process definitions, reassignments of threads to (new) processes, extended hardware specifications to include new scheduling partitions, updated software-hardware bindings, and added scheduling attributes on threads and hardware components. We extended the model further to support data flow (Risk Management Framework) analysis. Using GPS sensor data for the example, analysis will highlight data flow conflicts between software processes that access the same GPS sensor data. We added data flow specifications on software components. Using this model and analysis tools, we found potential data flow violations and fixed them by re-assigning threads to different software processes. However, making a change to fix a data flow violation caused a timing problem (also detected by model analysis!). We re-applied the scheduling analysis to generate a new ARINC 653 schedule.

We extended the SuperVolo SysML suite of models in four steps:

1. Separate Tasks into Scheduleable Processes.
2. Add Protected Shared Memory Components.
3. Assign MILS Domains to Applicable Hardware Components.
4. Create End-to-End Flows and Assign Confidentiality, Integrity, Availability (CIA) Values.

Together these steps align the system model into a design that reduces the change impact, both from a GPS software replacement perspective and from a general security perspective.

Separate Tasks into Scheduleable Processes.

The original open-source ArduPilot software design, which drives the avionics controls on the SuperVolo, hosts all 50+ avionics tasks within a single software process that executes via a round-robin scheduling paradigm on the ChibiOS real-time operating system. To satisfy security requirements, we assume that ChibiOS is replaced by a time-space partitioning operating system, and the SuperVolo design model is updated so that the avionics tasks are separated into multiple schedulable processes. Each process then is bound to its own scheduling partition, which in total enforces a time-partitioned scheduling paradigm (see Figure 6.4 and Figure 6.5). The data read/write access on the threads are also explicitly propagated up to software ports on the host processes in the model, in preparation for data flow representations captured in step 4 (see Figure 6.6 and Figure 6.9).

[55] The SysML from Taphos for now was limited to use as the starting point, because subsequent SysML translated from AADL overwrites a prior SysML model and does not maintain internal Cameo/MagicDraw references.

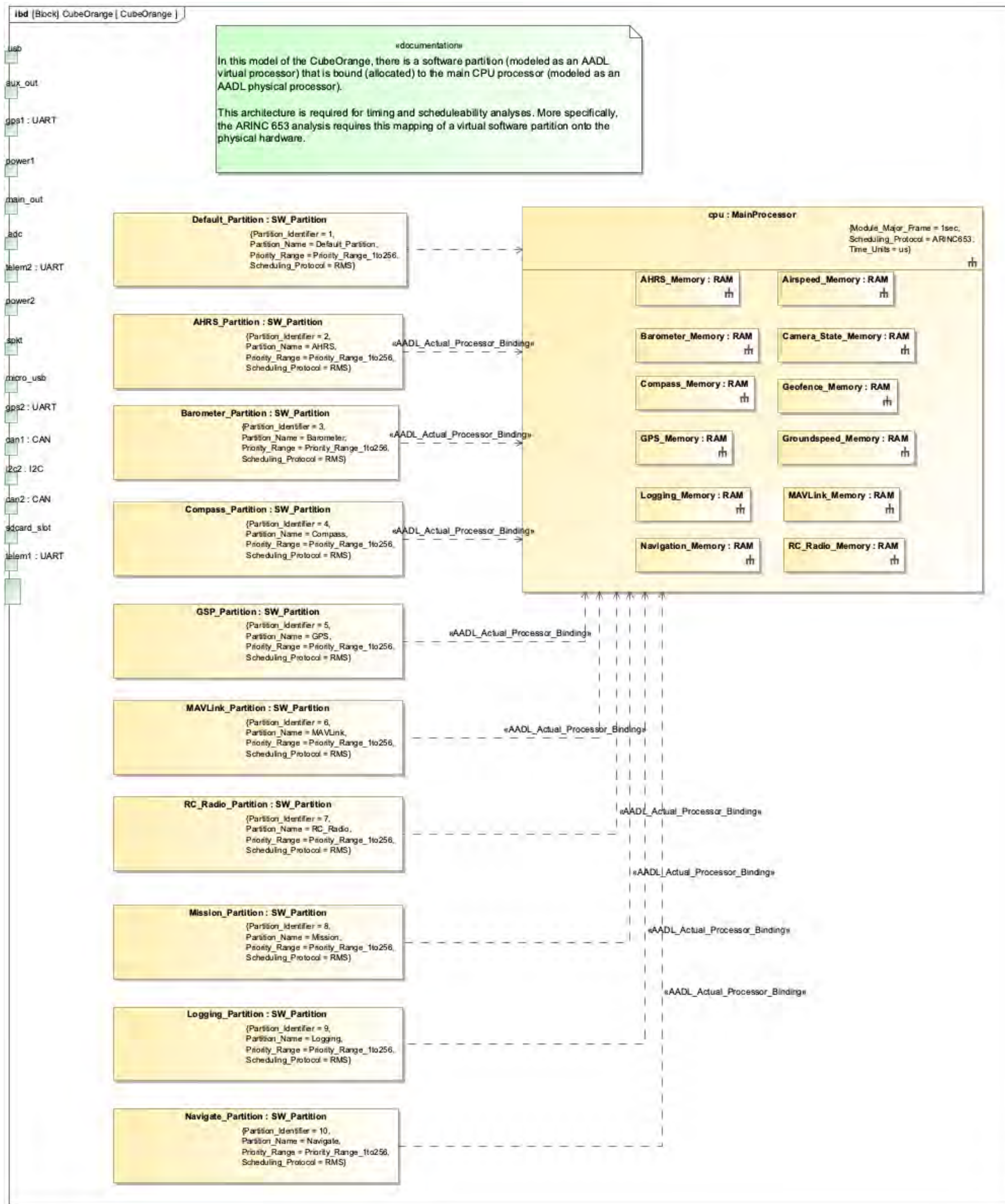
This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Figure 6.4 SysML model illustrating a SuperVolo system design configuration with software processes bound to scheduling partitions

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025



This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Figure 6.5 A SysML internal block diagram showing scheduling partitions of the SuperVolo design bound to the primary processor

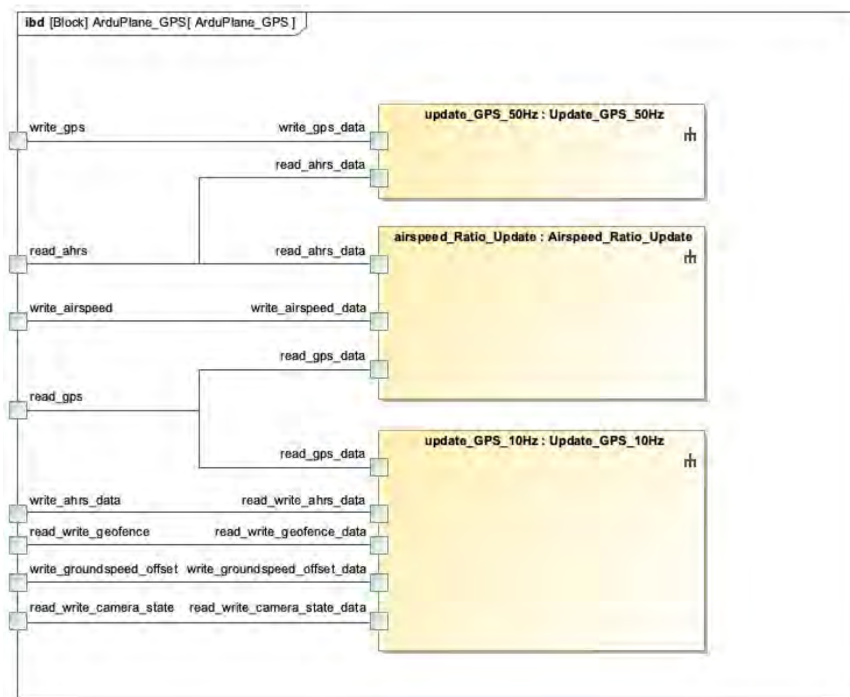


Figure 6.6 A SysML internal block diagram showing the decomposition of an example process with threads and port connections.

Add Protected Shared Memory Components.

Avionics tasks within the ArduPilot software interact through shared memory constructs (i.e., global variables). For our security analysis, we assume that these shared memory constructs are upgraded into protected memory components that strictly enforce read or write access to designated process APIs. In the resulting SysML, we capture each ArduPilot global variable as a typed data component hosted in a "protected space" software construct with separate read and write ports (see Figure 6.7). We modeled each interaction between processes and shared data as port read or write connections in the SysML, an example of which is shown in Figure 6.8. A bi-directional port that provides both read and write access is generally not permitted in systems with strict security controls. The explicit modeling of every read/write access is necessary to satisfy strict security controls and enables detailed representation of end-to-end data flows between processes and shared memory.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

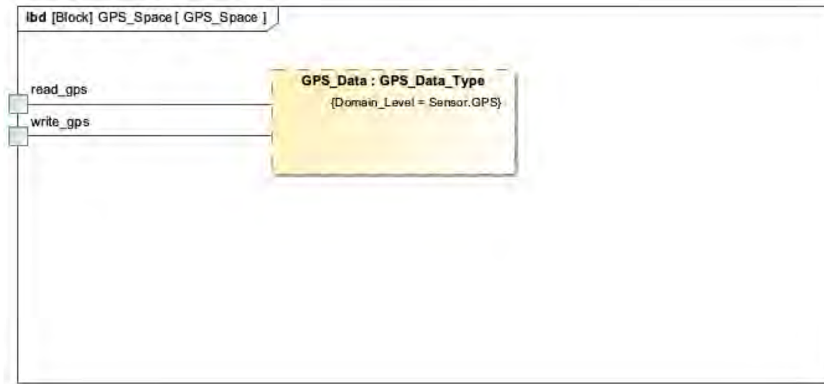
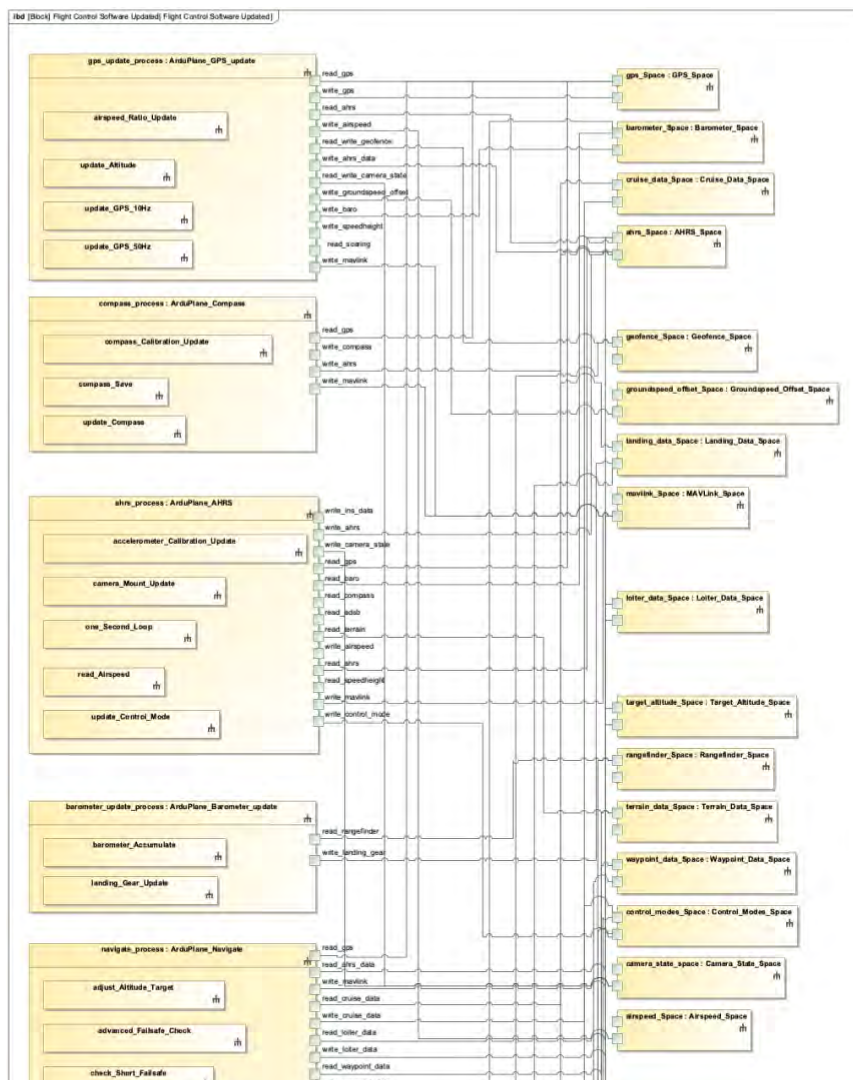


Figure 6.7 A SysML internal block diagram depicting a shared data component within protected space with read/write access



This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Figure 6.8 Example SysML internal block diagram showing the defined message connections between software processes and protected data components.

Assign MILS Domains to Applicable Hardware Components

As stated earlier, the open-source ArduPilot system out-of-the-box does not attempt to isolate software or hardware components according to a domain separation paradigm. As part of our SuperVolo upgrade, we tag specific hardware components in the SysML models, according to their assigned sensor domain. For example, hardware components with direct access to GPS related sensor data are tagged with a "GPS" domain. Details of the domain separation modeling are described in Part 4.

Create End-to-End Flows and Assign CIA Values

For our SuperVolo modeling effort, we capture a collection of end-to-end data flows in SysML to validate that our updated system design does not include data flows that violate security requirements. In steps 1 and 2 above, we modeled each read/write access between threads, processes, and shared data components in the system. The final step in representing end-to-end flows is creating a hierarchy of SysML sequence diagrams that decompose an end-to-end flow into its component source, sink, and intermediate path elements. See Figure 6.9 as an example flow source. The representation of data flows as a hierarchy of sequence diagrams may seem unnecessarily complicated at first. However, this is the format required by the AADL-based data flow analysis tool (described further below), where each intermediate component along the data flow path is explicitly defined.

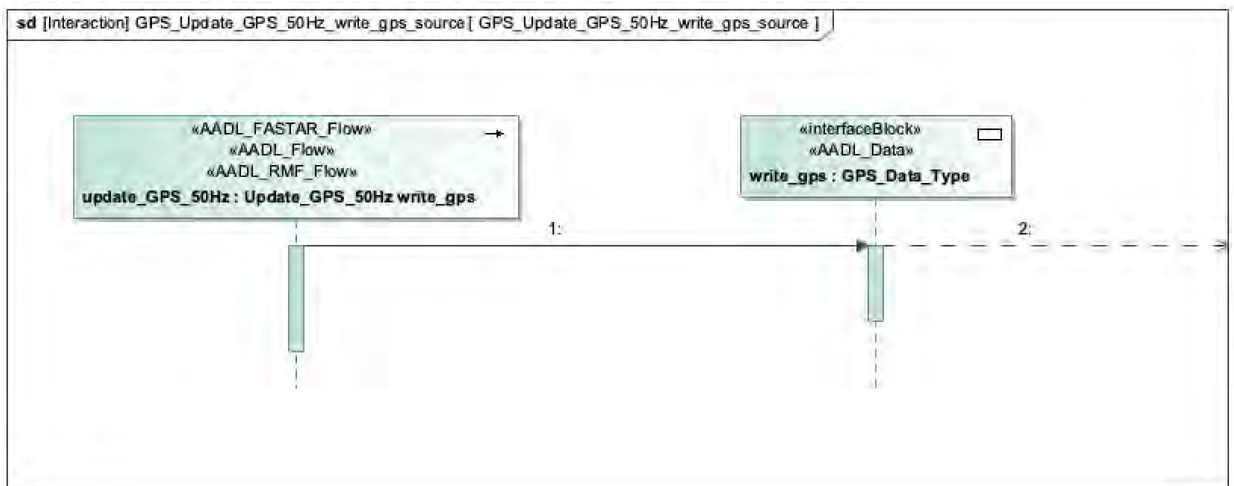


Figure 6.9 Example SysML sequence diagram showing a write operation data flow originating at a GPS related thread and passing through an outgoing process port

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Once the end-to-end flow is specified, it is assigned a priority level of high, medium, or low, for each of the three security flow attributes Confidentiality, Integrity, and Availability (CIA).[56] For this exercise, we use our best engineering judgment to determine the CIA attributes values to assign each end-to-end flow, based on our hypothetical GPS replacement scenario. In practice, these values would be defined directly in the requirements or identified by CONOPS established at the beginning of the design process.

Analyze: Measuring the Impact of the Design Updates

We approach the analysis phase of this design scenario broadly, with the understanding that fundamental architectural modifications are needed to achieve the expected levels of security. We break down this analysis phase into two parts that each focus on a major category of security controls: time partitioning and space partitioning.

Time partitioning

Time partitioning addresses both safety and security requirements. Without time partitioning, a malicious or errant task could disrupt or halt vehicle operations by preventing other tasks from executing (task starvation). The original SuperVolo system implementation deploys the ChibiOS real-time operating system with all 50+ avionics tasks scheduled within a single execution space. The RTOS provides for fine-grained scheduling of execution threads, but by itself does not enforce time partitioning. A malicious or poorly implemented logging task, for example, could prevent the aircraft from receiving timely GPS reading and jeopardize aircraft security. To satisfy strict DoD level time partitioning, the SuperVolo needs additional OS support to halt tasks when they exceed their allotted scheduling window.

ARINC 653 is the runtime standard most often applied to commercial and military avionics to enforce time partitioning. For the SuperVolo modeling and analysis, we assume that the refactoring of the SuperVolo legacy system includes switching from ChibiOS runtime environment to an ARINC 653 compliant operating system, like Green Hill Integrity or DEOS. The ARINC 653 standard requires a deterministic thread execution scheduling approach, which we achieve on SuperVolo via the CAMET FASTAR analysis tool.

FASTAR first analyzes a system design for scheduleability. If a valid schedule is feasible, FASTAR then attempts to create a deterministic ARINC 653 schedule from the system timing specifications captured in the SuperVolo models, in particular thread execution frequencies and worst-case execution times. (Note that the FASTAR tool is not officially certified for use on any existing DoD programs. In our SuperVolo demonstrations, the resulting schedule instead is envisioned as an early design point in the system development that is handed off to later stages for formal certification.)

[56] CIA attributes adopted here are defined by NIST Special Publication 1800-26.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

In our first invocation of FASTAR on the SuperVolo design models, we analyzed the initial configuration with all execution threads hosted in one process. As expected, a valid deterministic schedule was not feasible. The wide range of invocation frequencies and execution times distributed throughout the different threads make it mathematically infeasible to build a deterministic schedule that satisfies ARINC 653 constraints.

Next, we split the single process into six processes, each assigned to its own scheduling partition, and reassigned the 50+ threads to the new processes. The threads were distributed relatively equal among the six processes. Tasks were grouped in processes based on certain design attributes:

- Execution frequencies,
- Worst-case execution times,
- Common access to shared global data.

The decision on thread reassignment was a manual process using best engineering judgment. In practice, a larger engineering team that includes subject matter experts across multiple design domains would employ a more comprehensive process to determine how execution threads are assigned to processes. In this context, design change impact can be measured by the number of partitions that are needed to achieve a valid ARINC 653 schedule, and the number of execution threads that require modification to accommodate the refactored partitioned software architecture.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

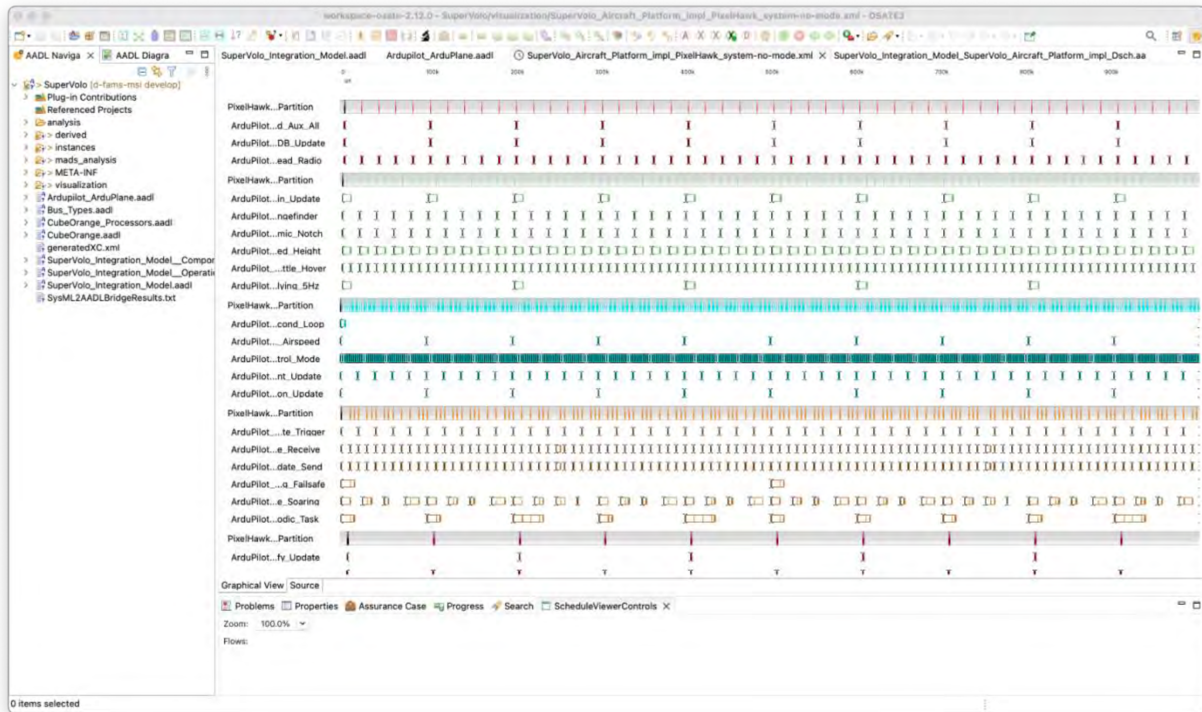


Figure 6.10 Example timing graph of a thread execution schedule from generated from the SuperVolo system design model by the FASTAR analysis tool

The initial division of threads into six processes, however, did not result in a feasible ARINC 653 schedule. Consequently, we built a third configuration of the ArduPilot software, where the threads are distributed between ten scheduleable processes. This resulted in a valid thread execution schedule. Figure 6.10 shows a selection of the schedule displayed as a timing graph, and Figure 6.11 shows a portion of the schedule displayed in AADL modeling format.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

```
SuperVolo_Integration_Model_SuperVolo_Aircraft_Platform_impl_Dsch.aadl
package SuperVolo_Integration_Model_SuperVolo_Aircraft_Platform_impl_Dsch
public
    with ARINC653;
    with SuperVolo_Integration_Model;

    SuperVolo_Aircraft_Platform renames system SuperVolo_Integration_Model::SuperVolo_Aircraft_Platform;

    system implementation SuperVolo_Aircraft_Platform.impl extends SuperVolo_Integration_Model::SuperVolo_Aircraft_Platform.impl
    properties
        ARINC653::Module_Major_Frame => 1000000 us applies to PixelHawk.cpu;
        ARINC653::Module_Schedule => (
            [Partition => reference(PixelHawk.Default_Partition); Duration => 700 us; Periodic_Processing_Start => true; ], -- 0..700
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 700 us; Periodic_Processing_Start => true; ], -- 700..1400
            [Partition => reference(PixelHawk.Navigate_Partition); Duration => 250 us; Periodic_Processing_Start => true; ], -- 1400..1650
            [Partition => reference(PixelHawk.Logging_Partition); Duration => 850 us; Periodic_Processing_Start => true; ], -- 1650..2500
            [Partition => reference(PixelHawk.Default_Partition); Duration => 200 us; Periodic_Processing_Start => false; ], -- 2500..2700
            [Partition => reference(PixelHawk.Compass_Partition); Duration => 500 us; Periodic_Processing_Start => true; ], -- 2700..3200
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 100 us; Periodic_Processing_Start => false; ], -- 3200..3300
            [Partition => reference(PixelHawk.MAVLink_Partition); Duration => 1700 us; Periodic_Processing_Start => true; ], -- 3300..5000
            [Partition => reference(PixelHawk.Default_Partition); Duration => 200 us; Periodic_Processing_Start => false; ], -- 5000..5200
            [Partition => reference(PixelHawk.Barometer_Partition); Duration => 500 us; Periodic_Processing_Start => true; ], -- 5200..5700
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 100 us; Periodic_Processing_Start => false; ], -- 5700..5800
            [Partition => reference(PixelHawk.RC_Radio_Partition); Duration => 600 us; Periodic_Processing_Start => true; ], -- 5800..6400
            [Partition => reference(PixelHawk.Navigate_Partition); Duration => 300 us; Periodic_Processing_Start => false; ], -- 6400..6700
            [Partition => reference(PixelHawk.GSP_Partition); Duration => 800 us; Periodic_Processing_Start => true; ], -- 6700..7500
            [Partition => reference(PixelHawk.Default_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 7500..8200
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 100 us; Periodic_Processing_Start => false; ], -- 8200..8300
            [Partition => reference(PixelHawk.Mission_Partition); Duration => 1700 us; Periodic_Processing_Start => true; ], -- 8300..10000
            [Partition => reference(PixelHawk.Default_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 10000..10700
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 10700..11400
            [Partition => reference(PixelHawk.Navigate_Partition); Duration => 250 us; Periodic_Processing_Start => false; ], -- 11400..11650
            [Partition => reference(PixelHawk.Logging_Partition); Duration => 850 us; Periodic_Processing_Start => false; ], -- 11650..12500
            [Partition => reference(PixelHawk.Default_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 12500..13200
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 100 us; Periodic_Processing_Start => false; ], -- 13200..13300
            [Partition => reference(PixelHawk.MAVLink_Partition); Duration => 1700 us; Periodic_Processing_Start => false; ], -- 13300..15000
            [Partition => reference(PixelHawk.Default_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 15000..15700
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 15700..16400
            [Partition => reference(PixelHawk.Navigate_Partition); Duration => 1100 us; Periodic_Processing_Start => false; ], -- 16400..17500
            [Partition => reference(PixelHawk.Default_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 17500..18200
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 1800 us; Periodic_Processing_Start => false; ], -- 18200..20000
            [Partition => reference(PixelHawk.Default_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 20000..20700
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 20700..21400
            [Partition => reference(PixelHawk.Navigate_Partition); Duration => 250 us; Periodic_Processing_Start => false; ], -- 21400..21650
            [Partition => reference(PixelHawk.Logging_Partition); Duration => 850 us; Periodic_Processing_Start => false; ], -- 21650..22500
            [Partition => reference(PixelHawk.Default_Partition); Duration => 200 us; Periodic_Processing_Start => false; ], -- 22500..22700
            [Partition => reference(PixelHawk.Compass_Partition); Duration => 500 us; Periodic_Processing_Start => false; ], -- 22700..23200
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 100 us; Periodic_Processing_Start => false; ], -- 23200..23300
            [Partition => reference(PixelHawk.MAVLink_Partition); Duration => 1700 us; Periodic_Processing_Start => false; ], -- 23300..25000
            [Partition => reference(PixelHawk.Default_Partition); Duration => 200 us; Periodic_Processing_Start => false; ], -- 25000..25200
            [Partition => reference(PixelHawk.Barometer_Partition); Duration => 500 us; Periodic_Processing_Start => false; ], -- 25200..25700
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 100 us; Periodic_Processing_Start => false; ], -- 25700..25800
            [Partition => reference(PixelHawk.RC_Radio_Partition); Duration => 600 us; Periodic_Processing_Start => false; ], -- 25800..26400
            [Partition => reference(PixelHawk.Navigate_Partition); Duration => 300 us; Periodic_Processing_Start => false; ], -- 26400..26700
            [Partition => reference(PixelHawk.GSP_Partition); Duration => 800 us; Periodic_Processing_Start => false; ], -- 26700..27500
            [Partition => reference(PixelHawk.Default_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 27500..28200
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 1800 us; Periodic_Processing_Start => false; ], -- 28200..30000
            [Partition => reference(PixelHawk.Default_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 30000..30700
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 30700..31400
            [Partition => reference(PixelHawk.Navigate_Partition); Duration => 250 us; Periodic_Processing_Start => false; ], -- 31400..31650
            [Partition => reference(PixelHawk.Logging_Partition); Duration => 850 us; Periodic_Processing_Start => false; ], -- 31650..32500
            [Partition => reference(PixelHawk.Default_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 32500..33200
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 100 us; Periodic_Processing_Start => false; ], -- 33200..33300
            [Partition => reference(PixelHawk.MAVLink_Partition); Duration => 1700 us; Periodic_Processing_Start => false; ], -- 33300..35000
            [Partition => reference(PixelHawk.Default_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 35000..35700
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 35700..36400
            [Partition => reference(PixelHawk.Navigate_Partition); Duration => 1100 us; Periodic_Processing_Start => false; ], -- 36400..37500
            [Partition => reference(PixelHawk.Default_Partition); Duration => 700 us; Periodic_Processing_Start => false; ], -- 37500..38200
            [Partition => reference(PixelHawk.AHRS_Partition); Duration => 1800 us; Periodic_Processing_Start => false; ], -- 38200..40000
```

Figure 6.11 Textual representation of the SuperVolo thread execution schedule in AADL format

Space Partitioning

Space partitioning in the context of SuperVolo SuperVolo development is addressed by strictly enforcing access to specified data at its storage point (shared global variable) and via the read/write access to the data (process APIs and messaging infrastructure). The general rule for secure DoD systems is that a given software component should only have access to certain data, especially safety-critical or mission critical data, that satisfies the requirement for that component (nothing more, nothing less). Framework standards such as OMS help to enforce these aspects of space partitioning. For the SuperVolo legacy system refactoring, we assume an underlying software infrastructure is deployed that enforces the necessary data control

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

mechanisms, such as unidirectional inter-task messaging and strict semaphore access to shared data.

The mixed-criticality stage of the Risk Management Framework (RMF) standard identifies an approach of validating the data flow of a system design to address space partitioning requirements.[57] For the SuperVolo effort, we applied the CAMET RMF tool to analyze a selected collection of data flows defined in the SuperVolo design models.

As described previously, the SuperVolo SysML model has been extended to define data flow access between the GPS system data and all the processes that potentially access the GPS data. A number of other data flows are also defined that access other sensor data, such as barometer sensor data and Altitude/Heading Reference System (AHRS) data. Each data flow is assigned a priority level for confidentiality, integrity, and availability. For demonstration purposes, we defined nine end-to-end flows in the SuperVolo design for analysis. In practice, the extended SuperVolo design models have on the order of 200 potential end-to-end flows for analysis.

The RMF tool validates that access to specified data in the design model adheres to its CIA flow specifications. If data flows intersect in the system design, then CIA values must be identical, otherwise a space-partitioning violation occurs. The tool generates a report indicating if the analysis passed or failed, listing each of the evaluated data flows, their CIA attributes, and any detected violations.

We started with the design configuration at the end of the FASTAR analysis, with the execution threads divided among 10 processes. This configuration resulted in a space-partitioning conflict (see Figure 6.12), which we remedied by reassigning the violating `Update_Altitude` thread from the barometer centric process to the GPS centric process. We extended the SysML model with another system configuration reflecting the thread reassignment and updating the associated end-to-end flows. The new system configuration passed the RMF data flow analysis (see Figure 6.13). We also ran the FASTAR analysis over the new configuration, which resulted in a new, valid thread execution schedule.

[57] DoDI 8510.01 Risk Management Framework for DoD Information Technology

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

supervolo_integration_model_supervolo_aircraft_platform_impl_RMF_1_report.txt

```
=========== Conflicts ===========
ArduPilot_secured.gps_space.GPS_Data.GPS_Data   (INTERSECT: ArduPilot_secured.FC_update_altitude_read_gps_flow, ArduPilot_secured.FC_update_gps_50hz_write_gps_flow)
ArduPilot_secured.gps_process   (INTERSECT: ArduPilot_secured.FC_update_gps_50hz_read_ahrs_flow, ArduPilot_secured.FC_update_gps_50hz_write_gps_flow)
(INTERSECT: ArduPilot_secured.FC_update_altitude_read_gps_flow, ArduPilot_secured.FC_update_gps_50hz_write_gps_flow)
(INTERSECT: ArduPilot_secured.FC_update_gps_50hz_read_ahrs_flow, ArduPilot_secured.FC_update_gps_50hz_write_gps_flow)


=========== Flow CIA Info ==============

ArduPilot_secured.FC_compass_cal_update_read_gps_flow           [C: moderate; I: high; A: moderate]
ArduPilot_secured.FC_gps_airspeed_ratio_update_read_gps_flow            [C: moderate; I: high; A: moderate]
ArduPilot_secured.FC_navigate_advanced_fs_read_baro_flow                [C: moderate; I: high; A: low]
ArduPilot_secured.FC_navigate_advanced_fs_read_gps_flow         [C: moderate; I: high; A: moderate]
ArduPilot_secured.FC_navigate_read_gps_flow             [C: moderate; I: high; A: moderate]
ArduPilot_secured.FC_one_second_loop_read_gps_flow              [C: moderate; I: high; A: moderate]
ArduPilot_secured.FC_update_altitude_read_gps_flow              [C: moderate; I: high; A: high]
ArduPilot_secured.FC_update_gps_50hz_read_ahrs_flow             [C: moderate; I: moderate; A: moderate]
ArduPilot_secured.FC_update_gps_50hz_write_gps_flow             [C: moderate; I: high; A: moderate]

=========== Element Participating Flow Info ==============

ArduPilot_secured.ahrs_process          [ArduPilot_secured.FC_one_second_loop_read_gps_flow]
ArduPilot_secured.ahrs_process.one_Second_Loop          [ArduPilot_secured.FC_one_second_loop_read_gps_flow]
ArduPilot_secured.ahrs_space.AHRS_Data.AHRS_Data                [ArduPilot_secured.FC_update_gps_50hz_read_ahrs_flow]
ArduPilot_secured.barometer_process             [ArduPilot_secured.FC_update_altitude_read_gps_flow]
ArduPilot_secured.barometer_process.update_Altitude             [ArduPilot_secured.FC_update_altitude_read_gps_flow]
ArduPilot_secured.barometer_space.barometer_Data.barometer_Data         [ArduPilot_secured.FC_navigate_advanced_fs_read_baro_flow]
ArduPilot_secured.compass_process               [ArduPilot_secured.FC_compass_cal_update_read_gps_flow]
ArduPilot_secured.compass_process.compass_Calibration_Update            [ArduPilot_secured.FC_compass_cal_update_read_gps_flow]
ArduPilot_secured.gps_process           [ArduPilot_secured.FC_gps_airspeed_ratio_update_read_gps_flow, ArduPilot_secured.FC_update_gps_50hz_read_ahrs_flow,
ArduPilot_secured.FC_update_gps_50hz_write_gps_flow]
ArduPilot_secured.gps_process.airspeed_Ratio_Update             [ArduPilot_secured.FC_gps_airspeed_ratio_update_read_gps_flow]
ArduPilot_secured.gps_process.update_GPS_50Hz           [ArduPilot_secured.FC_update_gps_50hz_read_ahrs_flow, ArduPilot_secured.FC_update_gps_50hz_write_gps_flow]
ArduPilot_secured.gps_space.GPS_Data.GPS_Data           [ArduPilot_secured.FC_compass_cal_update_read_gps_flow, ArduPilot_secured.FC_gps_airspeed_ratio_update_read_gps_flow,
ArduPilot_secured.FC_navigate_advanced_fs_read_gps_flow, ArduPilot_secured.FC_navigate_read_gps_flow, ArduPilot_secured.FC_one_second_loop_read_gps_flow,
ArduPilot_secured.FC_update_altitude_read_gps_flow, ArduPilot_secured.FC_update_gps_50hz_write_gps_flow]
ArduPilot_secured.navigate_process              [ArduPilot_secured.FC_navigate_advanced_fs_read_baro_flow, ArduPilot_secured.FC_navigate_advanced_fs_read_gps_flow,
ArduPilot_secured.FC_navigate_read_gps_flow]
ArduPilot_secured.navigate_process.advanced_Failsafe_Check              [ArduPilot_secured.FC_navigate_advanced_fs_read_baro_flow,
ArduPilot_secured.FC_navigate_advanced_fs_read_gps_flow]
ArduPilot_secured.navigate_process.navigate             [ArduPilot_secured.FC_navigate_read_gps_flow]

=========== Element CIA Info ==============

ArduPilot_secured.ahrs_process          [C: moderate; I: high; A: moderate]
ArduPilot_secured.ahrs_process.one_Second_Loop          [C: moderate; I: high; A: moderate]
ArduPilot_secured.ahrs_space.AHRS_Data.AHRS_Data                [C: moderate; I: moderate; A: moderate]
ArduPilot_secured.barometer_process             [C: moderate; I: high; A: high]
ArduPilot_secured.barometer_process.update_Altitude             [C: moderate; I: high; A: high]
ArduPilot_secured.barometer_space.barometer_Data.barometer_Data         [C: moderate; I: high; A: low]
ArduPilot_secured.compass_process               [C: moderate; I: high; A: moderate]
ArduPilot_secured.compass_process.compass_Calibration_Update            [C: moderate; I: high; A: moderate]
ArduPilot_secured.gps_process           [C: moderate; I: high, moderate; A: moderate]
ArduPilot_secured.gps_process.airspeed_Ratio_Update             [C: moderate; I: high; A: moderate]
ArduPilot_secured.gps_process.update_GPS_50Hz           [C: moderate; I: high, moderate; A: moderate]
ArduPilot_secured.gps_space.GPS_Data.GPS_Data           [C: moderate; I: high; A: high, moderate]
ArduPilot_secured.navigate_process              [C: moderate; I: high; A: low, moderate]
ArduPilot_secured.navigate_process.advanced_Failsafe_Check              [C: moderate; I: high; A: low, moderate]
ArduPilot_secured.navigate_process.navigate             [C: moderate; I: high; A: moderate]
```

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Figure 6.12 RMF analysis report showing data flow violations in the SuperVolo software architecture design

```
supervolo_integration_model_supervolo_aircraft_platform_updated_impl_RMF_1_report.txt

=========== Conflicts ===========

=========== Flow CIA Info ==============

ArduPilot_updated.FC_compass_cal_update_read_gps_flow          [C: moderate; I: high; A: moderate]
ArduPilot_updated.FC_navigate_advanced_fs_read_baro_flow          [C: moderate; I: high; A: low]
ArduPilot_updated.FC_navigate_advanced_fs_read_gps_flow          [C: moderate; I: high; A: moderate]
ArduPilot_updated.FC_navigate_read_gps_flow          [C: moderate; I: high; A: moderate]
ArduPilot_updated.FC_one_second_loop_read_gps_flow          [C: moderate; I: high; A: moderate]
ArduPilot_updated.FC_update_altitude_read_gps_flow          [C: moderate; I: high; A: moderate]
ArduPilot_updated.FC_update_gps_50hz_read_ahrs_flow          [C: moderate; I: high; A: moderate]
ArduPilot_updated.FC_update_gps_50Hz_write_gps_flow          [C: moderate; I: high; A: moderate]
ArduPilot_updated.FC_update_gps_airspeed_ratio_upate_read_gps_flow          [C: moderate; I: high; A: moderate]

=========== Element Participating Flow Info ==============

ArduPilot_updated.ahrs_process          [ArduPilot_updated.FC_one_second_loop_read_gps_flow]
ArduPilot_updated.ahrs_process.one_Second_Loop          [ArduPilot_updated.FC_one_second_loop_read_gps_flow]
ArduPilot_updated.ahrs_Space.AHRS_Data.AHRS_Data          [ArduPilot_updated.FC_update_gps_50hz_read_ahrs_flow]
ArduPilot_updated.barometer_Space.barometer_Data.barometer_Data          [ArduPilot_updated.FC_navigate_advanced_fs_read_baro_flow]
ArduPilot_updated.compass_process          [ArduPilot_updated.FC_compass_cal_update_read_gps_flow]
ArduPilot_updated.compass_process.compass_Calibration_Update          [ArduPilot_updated.FC_compass_cal_update_read_gps_flow]
ArduPilot_updated.gps_Space.GPS_Data.GPS_Data          [ArduPilot_updated.FC_compass_cal_update_read_gps_flow, ArduPilot_updated.FC_navigate_advanced_fs_read_gps_flow,
ArduPilot_updated.FC_navigate_read_gps_flow, ArduPilot_updated.FC_one_second_loop_read_gps_flow, ArduPilot_updated.FC_update_altitude_read_gps_flow,
ArduPilot_updated.FC_update_gps_50Hz_write_gps_flow, ArduPilot_updated.FC_update_gps_airspeed_ratio_upate_read_gps_flow]
ArduPilot_updated.gps_update_process          [ArduPilot_updated.FC_update_altitude_read_gps_flow, ArduPilot_updated.FC_update_gps_50Hz_write_gps_flow,
ArduPilot_updated.FC_update_gps_50hz_read_ahrs_flow, ArduPilot_updated.FC_update_gps_airspeed_ratio_upate_read_gps_flow]
ArduPilot_updated.gps_update_process.airspeed_Ratio_Update          [ArduPilot_updated.FC_update_gps_airspeed_ratio_upate_read_gps_flow]
ArduPilot_updated.gps_update_process.update_Altitude          [ArduPilot_updated.FC_update_altitude_read_gps_flow]
ArduPilot_updated.gps_update_process.update_GPS_50Hz          [ArduPilot_updated.FC_update_gps_50Hz_write_gps_flow, ArduPilot_updated.FC_update_gps_50hz_read_ahrs_flow]
ArduPilot_updated.navigate_process          [ArduPilot_updated.FC_navigate_advanced_fs_read_baro_flow, ArduPilot_updated.FC_navigate_advanced_fs_read_gps_flow,
ArduPilot_updated.FC_navigate_read_gps_flow]
ArduPilot_updated.navigate_process.advanced_Failsafe_Check          [ArduPilot_updated.FC_navigate_advanced_fs_read_baro_flow, ArduPilot_updated.FC_navigate_advanced_fs_read_gps_flow]
ArduPilot_updated.navigate_process.navigate          [ArduPilot_updated.FC_navigate_read_gps_flow]

=========== Element CIA Info ==============

ArduPilot_updated.ahrs_process          [C: moderate; I: high; A: moderate]
ArduPilot_updated.ahrs_process.one_Second_Loop          [C: moderate; I: high; A: moderate]
ArduPilot_updated.ahrs_Space.AHRS_Data.AHRS_Data          [C: moderate; I: high; A: moderate]
ArduPilot_updated.barometer_Space.barometer_Data.barometer_Data          [C: moderate; I: high; A: low]
ArduPilot_updated.compass_process          [C: moderate; I: high; A: moderate]
ArduPilot_updated.compass_process.compass_Calibration_Update          [C: moderate; I: high; A: moderate]
ArduPilot_updated.gps_Space.GPS_Data.GPS_Data          [C: moderate; I: high; A: moderate]
ArduPilot_updated.gps_update_process          [C: moderate; I: high; A: moderate]
ArduPilot_updated.gps_update_process.airspeed_Ratio_Update          [C: moderate; I: high; A: moderate]
ArduPilot_updated.gps_update_process.update_Altitude          [C: moderate; I: high; A: moderate]
ArduPilot_updated.gps_update_process.update_GPS_50Hz          [C: moderate; I: high; A: moderate]
ArduPilot_updated.navigate_process          [C: moderate; I: high; A: low, moderate]
ArduPilot_updated.navigate_process.advanced_Failsafe_Check          [C: moderate; I: high; A: low, moderate]
ArduPilot_updated.navigate_process.navigate          [C: moderate; I: high; A: moderate]
```

Figure 6.13 RMF analysis report showing that the updated SuperVolo software architecture has removed the prior data flow violation

It is important to reemphasize that for larger-scale vehicle design projects an iterative modeling and analysis process is strongly recommended, for both legacy system upgrades and systems built from scratch. For example, system designers make updates to the model, perform a suite of analyses upon the updates, evaluate the results, evolve the design further with new updates to the model, and so on throughout the design and implementation process. This approach is made possible by capturing the entire system design in a single integrated modeling environment. An investment in modeling up-front saves time and budget downstream, which encourages a better, more secure overall system design.

Tutorial: Running Risk Management Framework Mixed Criticality Analysis

Galois's CAMET Library of MBSE tools includes the Risk Management Framework (RMF) Mixed Criticality Analysis tool. This tutorial explains how to run the RMF Mixed Criticality Analysis tool on a model, starting from the SuperVolo SysML model.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Open the SuperVolo model in Dassault's MagicDraw or Cameo 2024. For the former, you will need to install the SysML plugin (Cameo is MagicDraw with a variety of additional plugins, including SysML). For either MagicDraw or Cameo, you must install the SysML to AADL Bridge, available on CAMET (see Figure 6.14).

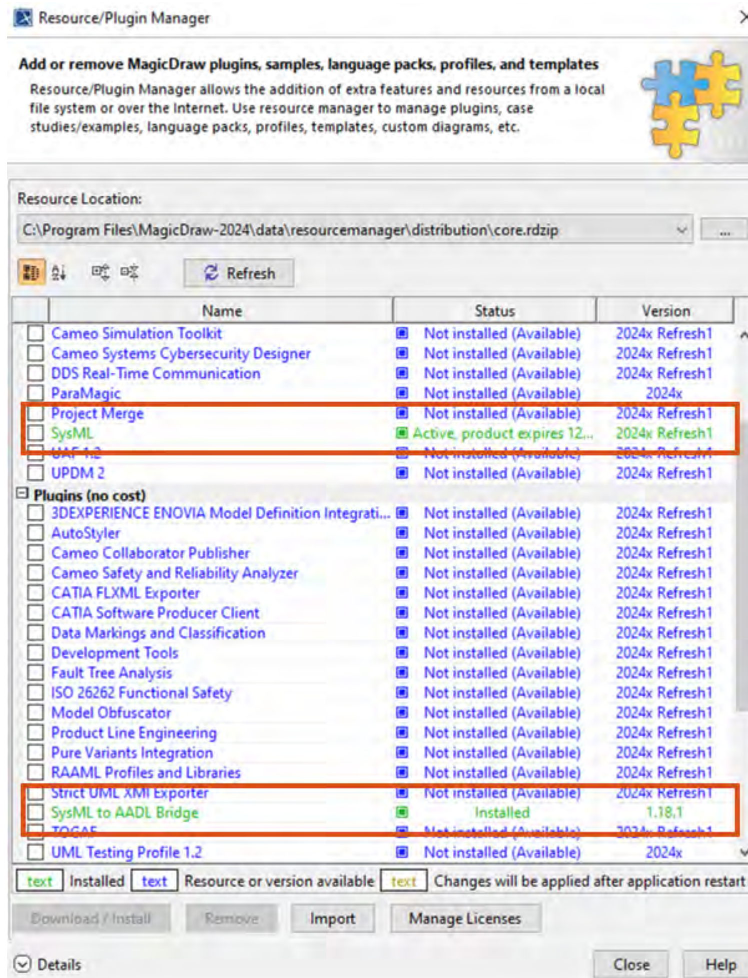


Figure 6.14 Screenshot of the Plugin Manager in MagicDraw with plugins needed for this tutorial.

There are two primary *projects* in the SuperVolo model. The SuperVolo itself, and the Ardupilot project, which is loaded as a *used project*. For this tutorial we do not need to modify the configuration of the Ardupilot; doing so would require opening its model for read/write access.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

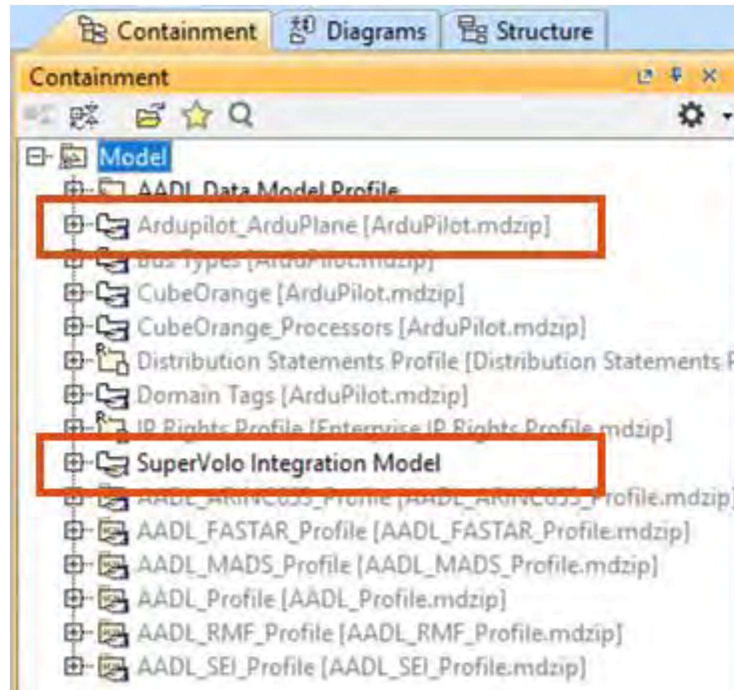


Figure 6.15 Containment tree after opening the SuperVolo model. Note that Ardupilot_Arduplane is loaded as read-only, hence the gray text.

The SuperVolo and Ardupilot models include a variety of stereotypes on its blocks that provide support for AADL Properties. Open `Ardupilot_Arduplane` to see the threads in Ardupilot (see Figure 6.16).

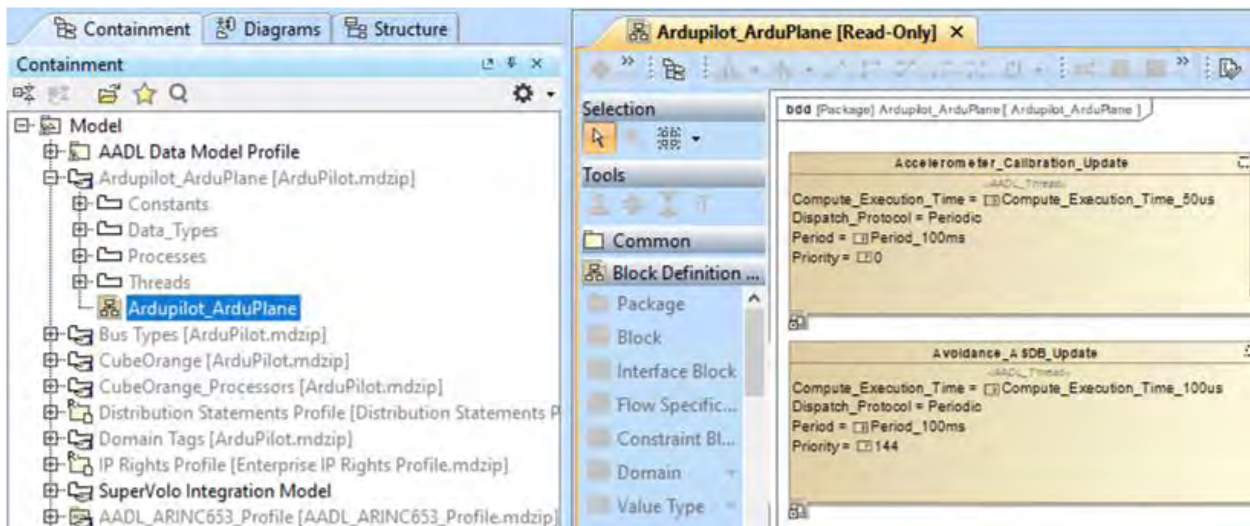


Figure 6.16 Screenshot of the Ardupilot_Arduplane model

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Software threads like Accelerometer_Calibration_Update are *stereotyped* as `AADL_Thread` and have associated properties that govern their scheduling, such as `Period` and `Priority`. (see Figure 6.16). See the FASTAR user manual for details on these properties and their interpretation.

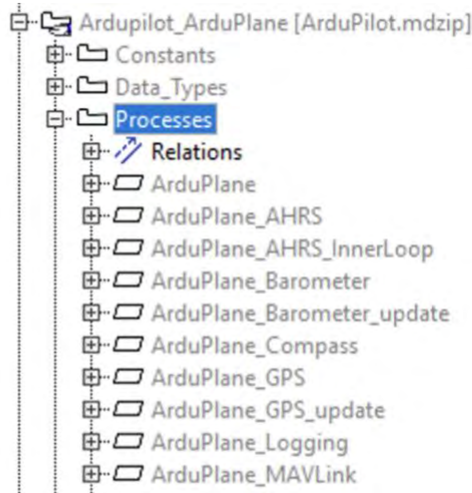


Figure 6.17 The Processes package under Ardupilot_Arduplane shows the memory spaces defined for the partitioning exercise in this case study. Each process is a memory space.

Threads are allocated to processes by usage as parts in a process. Open the `Arduplane_AHRS IBD` to see the threads allocated to it (see Figure 6.17 and Figure 6.18).

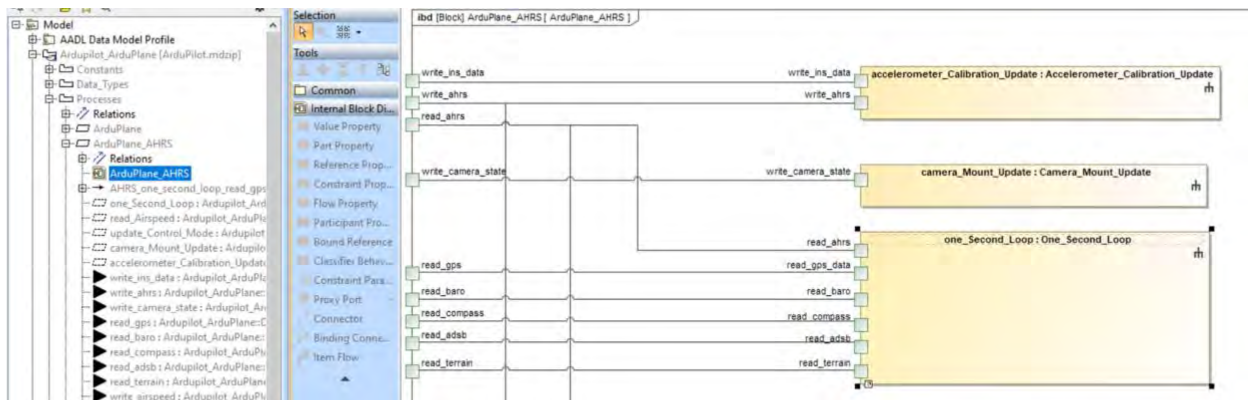


Figure 6.18 Arduplane_AHRS Process IBD

Right click on the SuperVolo Integration Model in the containment tree and select `Generate AADL` (see Figure 6.19). Choose a directory for the generated AADL (the default usually works fine).

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

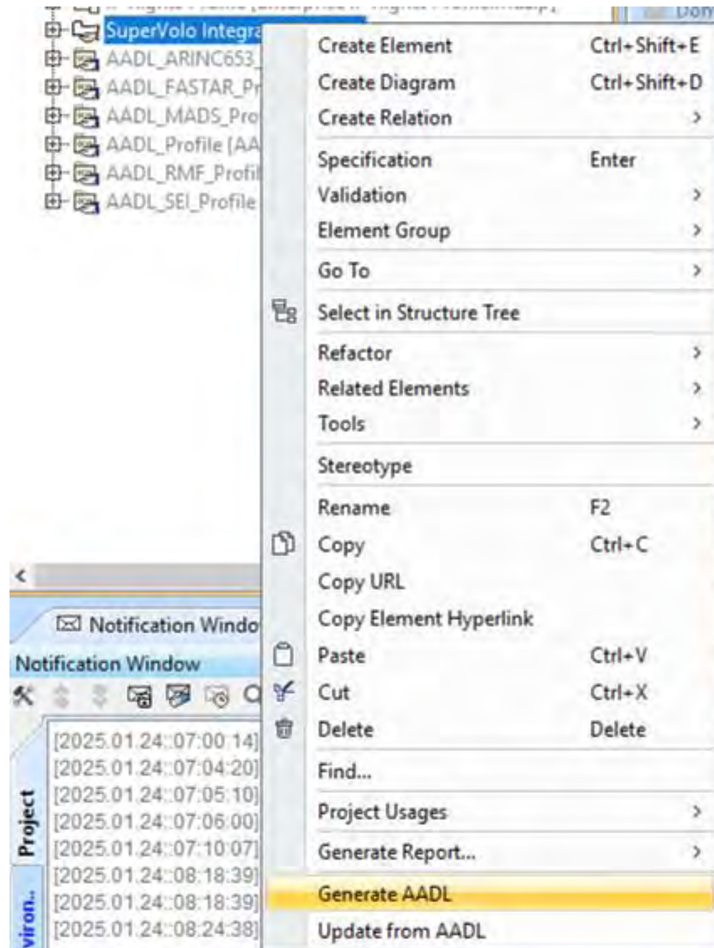


Figure 6.19 Generate AADL Option

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

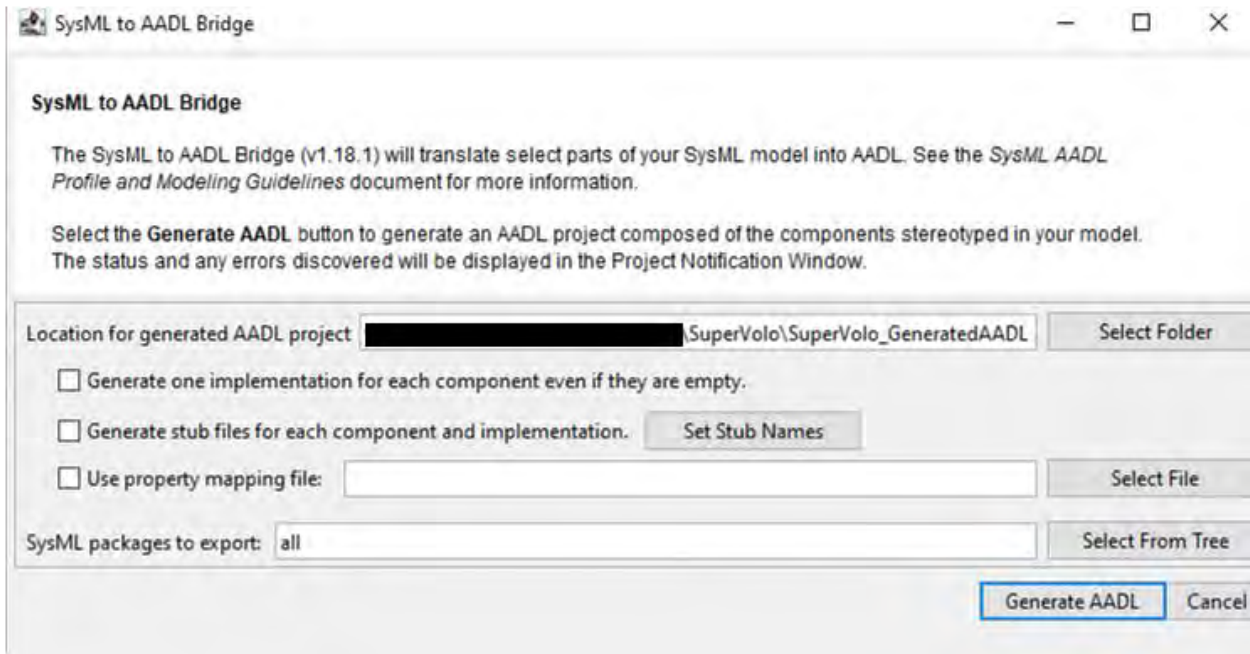


Figure 6.20 AADL Generation Options

This will generate a collection of AADL files and an OSATE project file in the specified directory (see Figure 6.20 and Figure 6.21).

Name	Date modified	Type	Size
META-INF	1/28/2025 11:30 AM	File folder	
.project	1/28/2025 11:30 AM	PROJECT File	1 KB
Ardupilot_ArduPlane_Data_Types.aadl	1/28/2025 11:30 AM	AADL File	4 KB
Ardupilot_ArduPlane__Processes.aadl	1/28/2025 11:30 AM	AADL File	56 KB
Ardupilot_ArduPlane__Threads.aadl	1/28/2025 11:30 AM	AADL File	56 KB
Bus_Types.aadl	1/28/2025 11:30 AM	AADL File	1 KB
CubeOrange.aadl	1/28/2025 11:30 AM	AADL File	18 KB
CubeOrange_Processors.aadl	1/28/2025 11:30 AM	AADL File	5 KB
generatedXC.xml	1/28/2025 11:30 AM	XML File	0 KB
SuperVolo_Integration_Model.aadl	1/28/2025 11:30 AM	AADL File	30 KB
SuperVolo_Integration_Model__Compon...	1/28/2025 11:30 AM	AADL File	50 KB
SuperVolo_Integration_Model__Operatio...	1/28/2025 11:30 AM	AADL File	5 KB
SuperVolo_Integration_Model__SW_Prote...	1/28/2025 11:30 AM	AADL File	24 KB
SysML2AADLBridgeResults.txt	1/28/2025 11:30 AM	Text Document	1 KB

Figure 6.21 Files generated by the SysML to AADL Bridge

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Launch OSATE. Install the CAMET Base Pack if you have not already done so (see the CAMET Quick Start Guide for instructions).

Click File → Import... then select `Existing Project Into Workspace` and select the folder created by the AADL Generation (see Figure 6.22 and Figure 6.23).

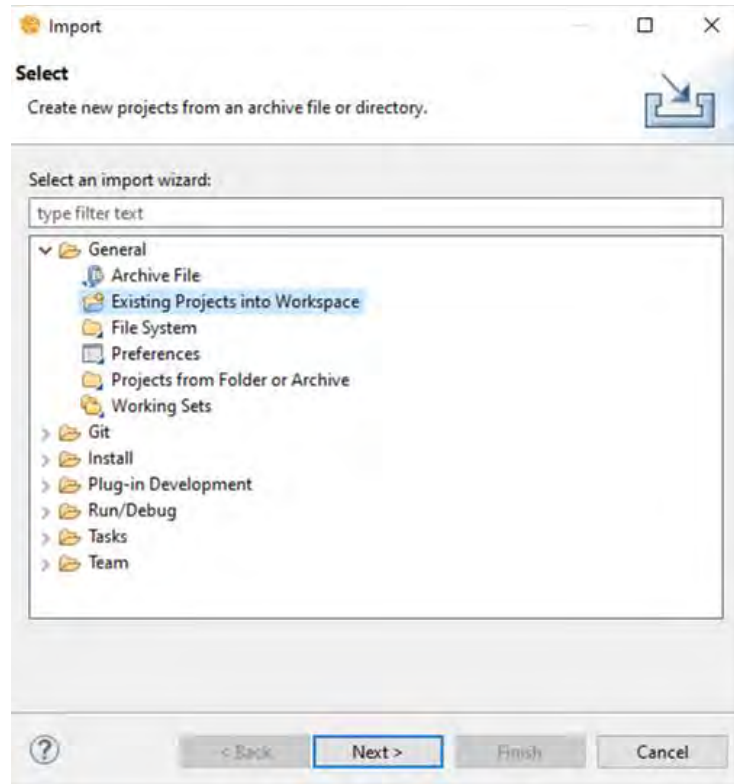


Figure 6.22 OSATE Import Dialog

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

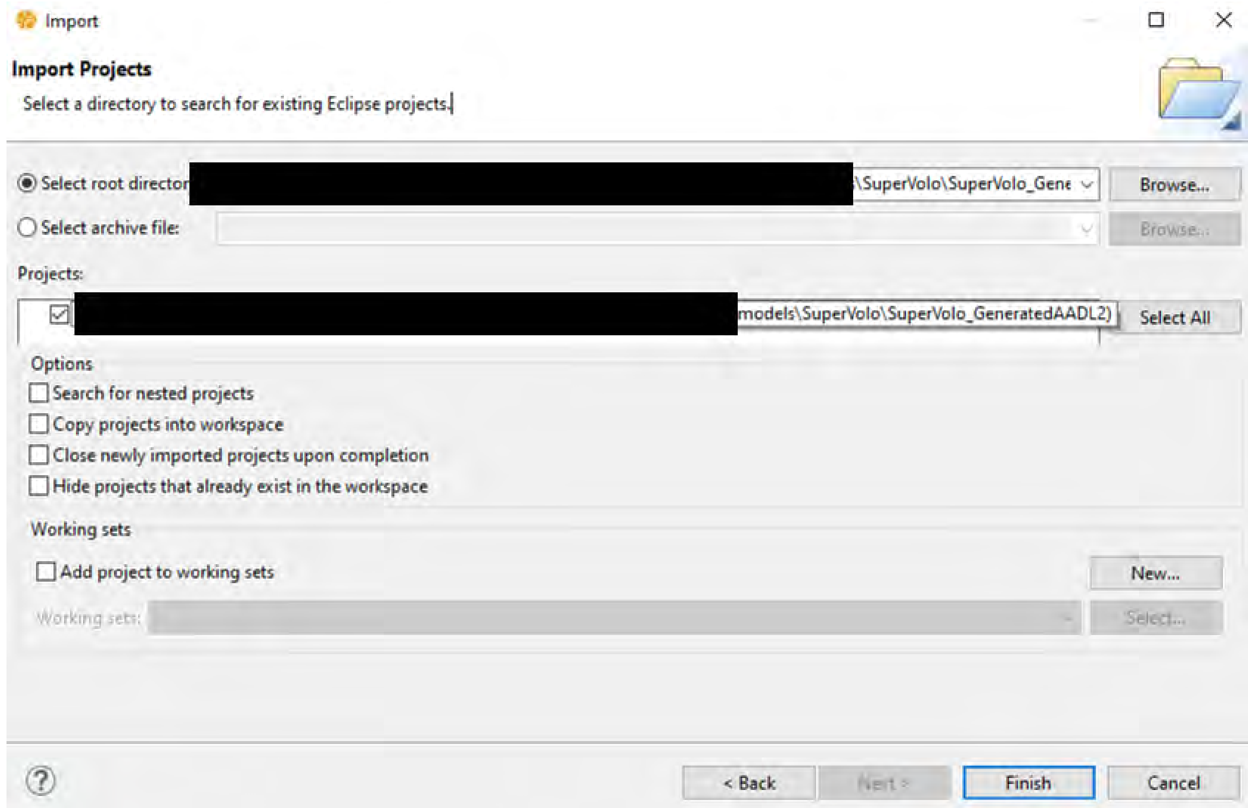


Figure 6.23 OSATE Import Details Dialog

Double click on `SuperVolo_Integration_Model.aadl` to open it in the AADL text editor.

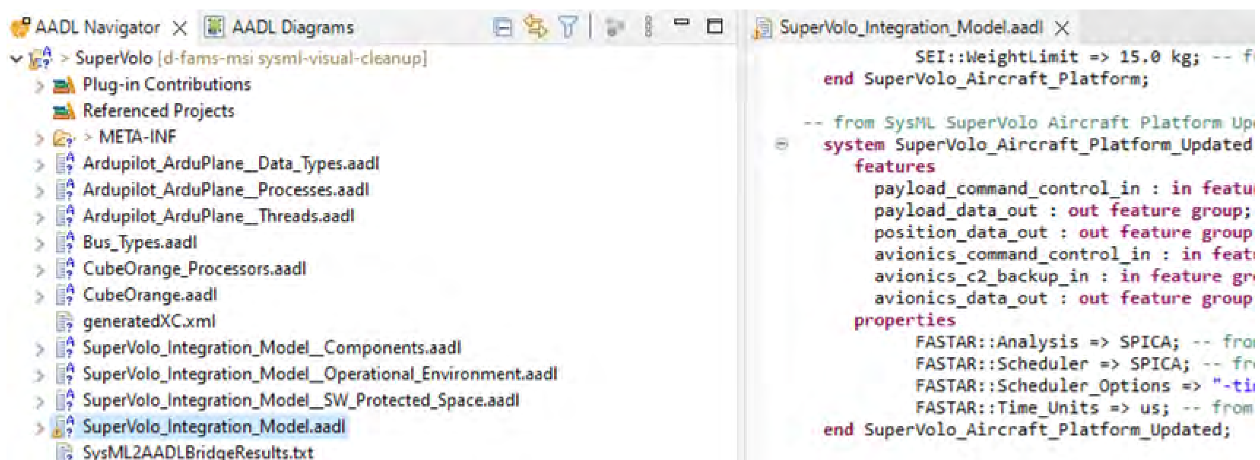


Figure 6.24 `SuperVolo_Integration_Model.aadl` selected in the AADL Navigator

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Select System SuperVolo_Aircraft_Platform_Updated.impl in the Outline view, right click, and select Instantiate (see Figure 6.25).

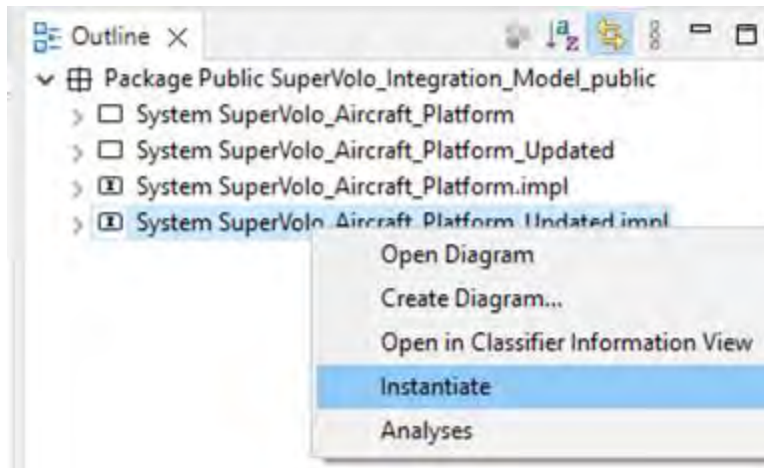
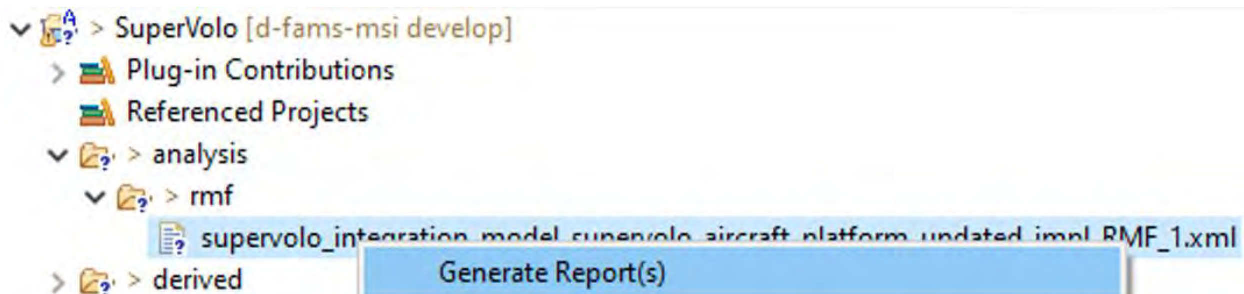


Figure 6.25 Instantiation Option in the Outline View

Figure 6.26 RMF Analysis Option

Select the resulting instance file (SyperVolo_Integration_Model_SuperVolo_Aircraft_Platform_Updated_impl_instance.aaxl2) in the AADL Navigator, then select CAMET → RMF → RMF Step 1 (see Figure 6.26). This will launch the Risk Management Framework Mixed Criticality analysis. The analysis tool creates an .xml file with its results. You can generate an easy-to-read version by right clicking on the resulting .xml file (under analysis/rmf) and selecting Generate Report(s) (see Figure 6.27).



This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Figure 6.27 Generate Report(s) Button

This creates a `.txt` file listing data flows modeled in the ArduPilot, their Confidentiality, Integrity, and Availability (CIA) levels, and which components of the SuperVolo architecture participate in each data flow (see Figure 6.28).

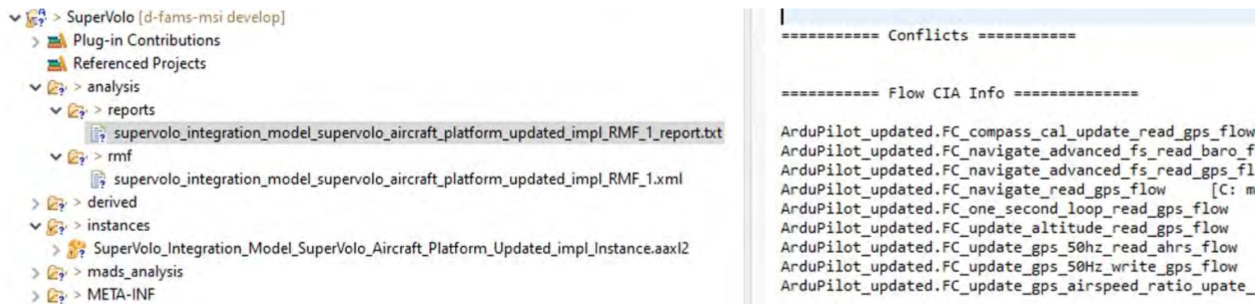


Figure 6.28 Textual report from RMF Step 1

Now try modifying one of the data flows in the SysML so that it has a *different* set of CIA criticality values. For example, change `FC_compass_cal_update_read_gps_flow` so that its criticality is Low-Low-Low instead of Moderate, High, Moderate.

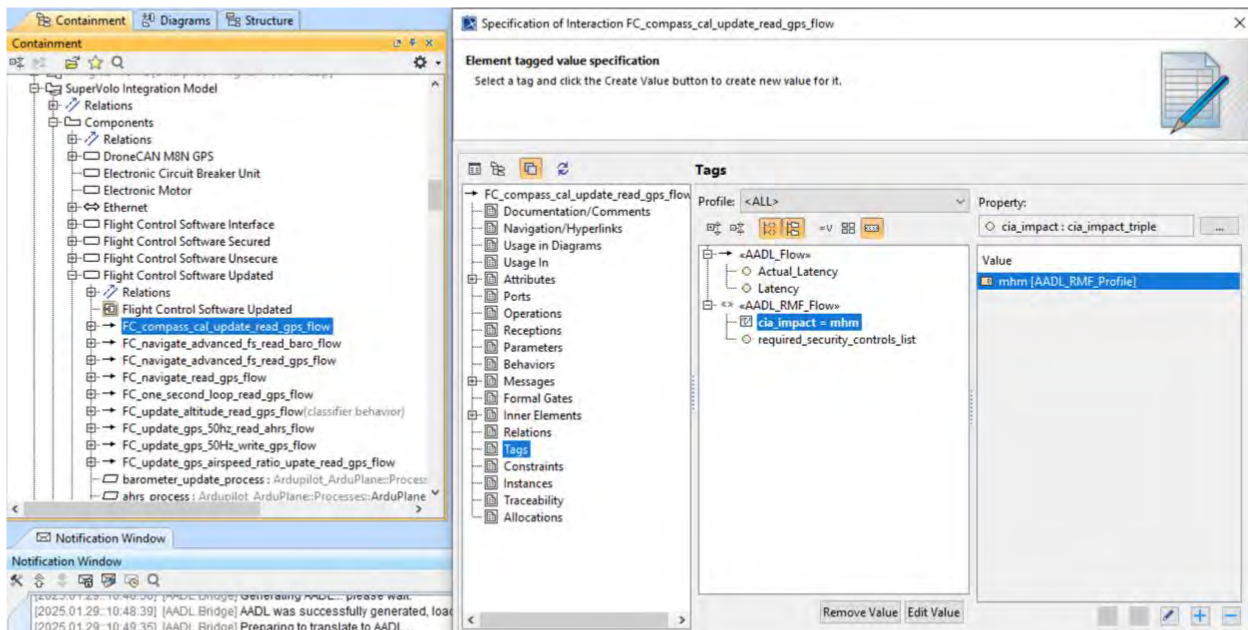


Figure 6.29 `cia_impact` tag in SysML

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Make this change by opening the specification of the `FC_compass_cal_update_read_gps_flow` interaction and changing the value of its `cia_impact` tag (see Figure 6.29 and Figure 6.30).

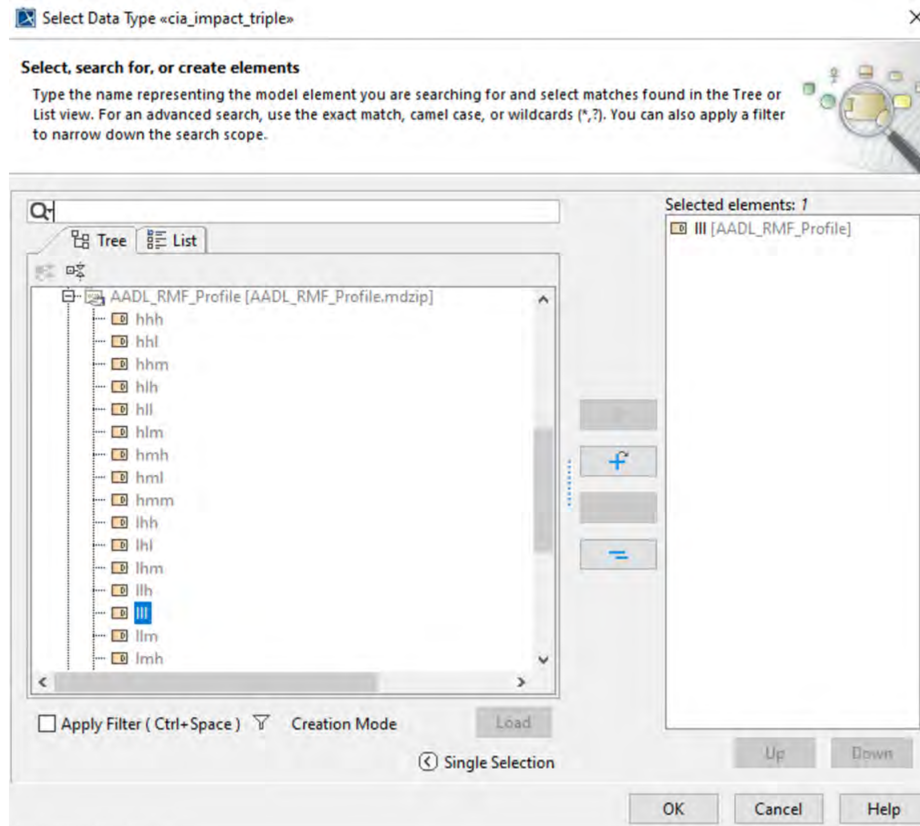


Figure 6.30 The different `cia_impact` options are provided by the `AADL_RMF_Profile`.

Re-generate the AADL, then re-run the RMF analysis. You'll get a message indicating that RMF analysis failed (see Figure 6.31).

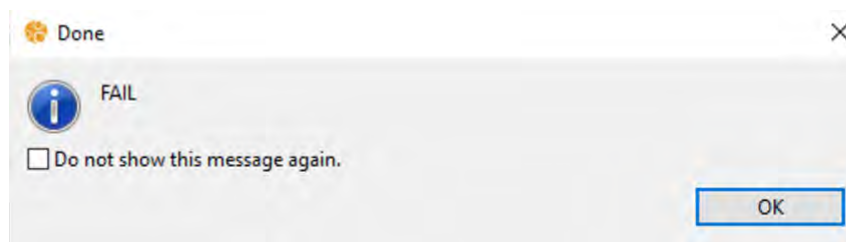


Figure 6.31 Failure dialog after changing flow criticality levels

Re-generate the RMF textual report to see why: The RMF report (see Listing 6.1) indicates that there is a conflict – flows with two different criticality levels share a common resource.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Listing 6.1 Excerpt from RMF analysis report showing conflicts between flows from different criticality domains.

```
=========== Conflicts ===========
ArduPilot_updated.gps_Space.GPS_Data.GPS_Data       (INTERSECT:
ArduPilot_updated.FC_compass_cal_update_read_gps_flow,
ArduPilot_updated.FC_update_gps_50Hz_write_gps_flow)
(INTERSECT: ArduPilot_updated.FC_compass_cal_update_read_gps_flow,
ArduPilot_updated.FC_update_gps_50Hz_write_gps_flow)
```

For more details on the RMF analysis tool, see the RMF Analysis Tool User Guide. For more information on ARINC 653 standard and the CAMET Schedule Generator, consult the *FASTAR Resource and Timing Analysis User Guide*.

Contain: Software Changes and Results

Unlike Part 5, in which we conducted analysis once and made a single decision, for this use case we conducted analysis many times to help make containment decisions.

Figure 6.32 Iterative Use of Analysis and Containment

The containment step requires repeating this process until your system meets functionality requirements and containment objectives (see Figure 6.32). Schedule and budget constraints prevented us from actually applying the time and space partitioning design we developed in the analysis phase. To enact the changes to the hardware and software architecture reflected in the analysis would require:

- The insertion of an underlying ARINC 653 compliant real-time operating system. As all of the software has dependency relationships to the operating system, this change impact propagates to *all* software, requiring rebuilding the system implementation re-targeted on the new operating system.
- The implementation of software timing partitions and the (non-overlapping) allocation of execution threads to partitions in an ARINC653 schedule. This would involve changing the existing thread execution schedules for most software tasks in the SuperVolo, again with significant change impact propagation.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

These are non-trivial changes and would impact, either directly or indirectly (measurable with Taphos, as discussed in Part 5) most of the software in the SuperVolo. Testing and certifying such changes would be expensive in both time and money. Such up-front effort may yield significant savings in the long run, as subsequent changes to system software would have reduced change impact (as discussed in Part 4).

For example, *after* implementing a time and space partitioned architecture, changes to a software thread in one partition are guaranteed isolation from software threads in other partitions. This means that although the initial change impact of implementing a partitioned architecture is high, later impacts will be lower because we have **entirely removed some avenues of change impact propagation**.

Conclusion

In this section we hypothesized that a stakeholder required domain separation between elements of the SuperVolo operational software. We discussed several approaches to domain separation (e.g., time and space partitioning) that can remove avenues of change impact propagation between components. We introduced analysis tools for labeling components with different domains, and for analyzing or generating configuration data for such components. We did not implement a domain separate architecture in the scope of this study, as the up-front costs (measurable via change impact) of changing the underlying software infrastructure are high. Were a real stakeholder to request such a change, we would collaborate with that stakeholder to determine whether the initial costs would be balanced by long term savings.

Part 7: Case Study - Add Software Defined Radio to SuperVolo

In Part 6 we explored what it would take to implement domain separation for the SuperVolo's Ardupilot flight control software. Using model-based analysis tools, we determined that implementing domain separation for Ardupilot is feasible (by analyzing things like a partitioned computing schedule) but that implementing such an architecture would be challenging due to extensive change impacts on Ardupilot software.

In this case study, we explore several alternative techniques to making software changes to the SuperVolo that do not have as extensive up-front costs as replacing its underlying computing architecture. Specifically, we apply software standards and code generation to rapidly generate standards-conformant code.

Frame: Mission Requires Secondary Navigation System

For this scenario we hypothesize that stakeholders want to make the Supervolo resistant to loss of GPS by adding another radio for ELORAN and updating software to include ELORAN sensor data in navigation processing. ELORAN is a GPS alternative that uses terrestrial towers to

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

provide navigation signals, rather than the satellites used by GPS.[58] Further, we assume that stakeholders wish for an ELORAN implementation that minimizes change impacts.

We implement this change by selecting additional radios to provide input as ELORAN sensors, and selecting software approaches for processing and communicating ELORAN location information with the rest of the SuperVolo architecture.

Figure 7.1 Software Defined Radios (ADALM-PLUTO and (Nooelec NESDR Smart XTR)

Model: New Components and Candidate Integration Sites

We started from the SysML models we developed and extended in Part 5 and Part 6. We *extended* these models by applying the Tangram Pro SysML profile. Tangram Pro supports a variety of Government Reference Architectures (GRAs) and associated software messaging standards.

For our SuperVolo development, we selected the Weapons Open Systems Architecture (WOSA) standard message format to pass information between a simulated ELORAN radio and a low-level avionics task responsible for aggregating sensor data. To meet the WOSA standard,

[58] https://en.wikipedia.org/wiki/ELoran

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

we need to insert WOSA message handling code into the open-source ArduPilot software to serialize and de-serialize message payloads. We chose to use the auto-generation tool TangramPro™ for this task, which supports many Government standards, including WOSA, for messaging. We used Tangram Pro to model the messaging connections between the producers and consumers of ELORAN data using WOSA (Figure 7.2).

The figure below is a snapshot of the simple model created in Tangram Pro. The boxes in the model represent generic tasks in the ArduPilot implementation that will read and write the ELORAN specific message. We modeled the tasks generically so that the resulting support code has the flexibility for insertion into the implementation where needed. The arrow between the blocks in the model represents the message flow, and in this case the messaging is set to a format that handles a specific ELORAN navigation data.

Together, these SysML and Tangram Pro models described our planned change to the SuperVolo architecture.

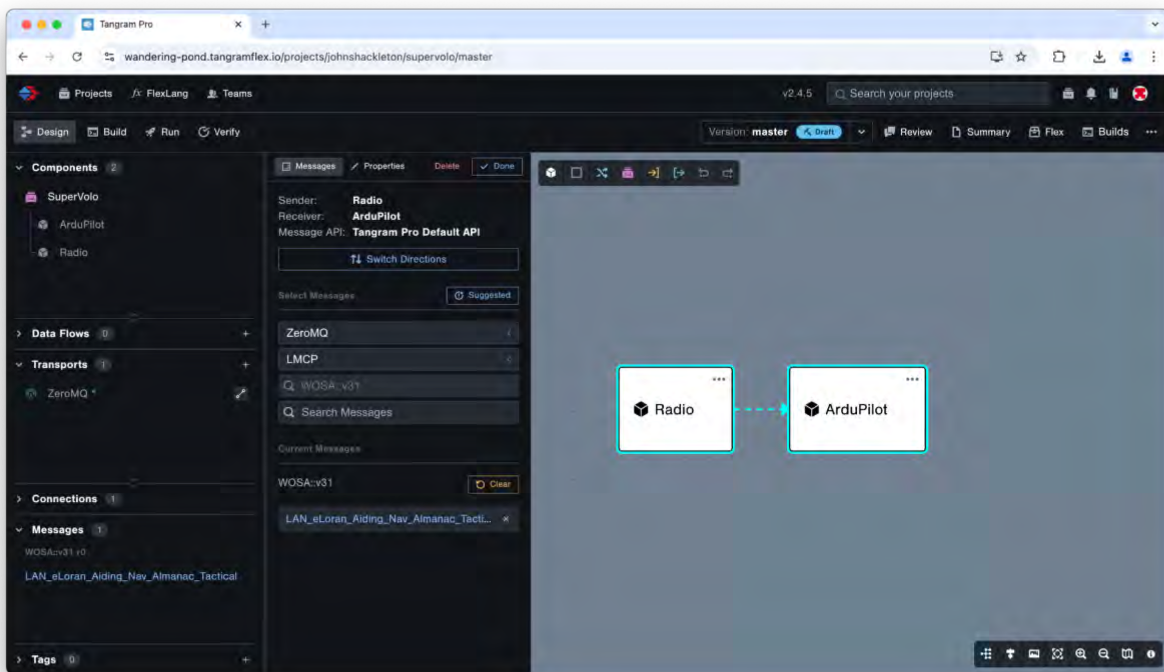


Figure 7.2 In this example model in TangramPro, we model a specific WOSA message type passed between two avionics tasks.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Analyze: Evaluate impact of integration options

Software Integration Options

We needed to decide *where* in the Supervolo Ardupilot software architecture to insert new software for handling ELORAN data (remember, there are lots of dimensions to the term “where” because there are lots of ways components relate to one another). We made several pragmatic decisions for this integration activity based on engineering experience and time constraints:

- We opted to modify the Ardupilot source directly, as there was no practical alternative method to integrate new location data.
- We opted to incorporate the ELORAN processing into an existing (i.e. already scheduled) task.

The Ardupilot software includes a variety of tasks, which we incorporated into our SysML model using a combination of Galois’s Taphos tool and manual inspection (see Figure 7.3).

Figure 7.3 Snapshot of the Arduplane SysML model, which includes dozens of periodic tasks.

There are several methods by which we could determine whether to add a new task or to add processing of ELORAN data to an existing task.[59]

Risk Insertion and Worst Case Error Impact

We evaluated several potential insertion points for new software by assessing their incoming and outgoing dependencies. The existing tasks, along with their execution schedule, are listed in ArduPlane.cpp (see Listing 7.1).

[59] Note: the terms “task” and “thread” are sometimes used interchangeably. The AADL uses “thread” to describe a software task independent of the memory space in which it executes.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Listing 7.1 Scheduler Excerpt from Ardupilot

```
/*
 scheduler table - all regular tasks are listed here, along with how
 often they should be called (in Hz) and the maximum time
 they are expected to take (in microseconds)
*/
const AP_Scheduler::Task Plane::scheduler_tasks[] = {
                           // Units:   Hz      us
   SCHED_TASK(ahrs_update,           400,    400),
   SCHED_TASK(read_radio,             50,    100),
   SCHED_TASK(check_short_failsafe,   50,    100),
   SCHED_TASK(update_speed_height,    50,    200),
   SCHED_TASK(update_control_mode,   400,    100),
   SCHED_TASK(stabilize,             400,    100),
   SCHED_TASK(set_servos,            400,    100),
   SCHED_TASK(update_throttle_hover, 100,     90),
   SCHED_TASK(read_control_switch,     7,    100),
   SCHED_TASK(update_GPS_50Hz,        50,    300),
   SCHED_TASK(update_GPS_10Hz,        10,    400),
   SCHED_TASK(navigate,               10,    150),
…
```

For the purposes of this experiment, we considered the existing tasks as integration points for new functionality. Using an existing task would let us minimize disruption to the processing schedule.

Loading the Arduplane bitcode into Taphos, we can inspect each of these functions to see how it interacts with the rest of the code base (see Listing 7.2). For information on how to launch Taphos, see the Taphos Tutorial from Part 5.

Listing 7.2 Taphos Commands to Analyze Arduplane Bitcode

```
load bitcode-sitl-plane.bc
analyze bitcode-sitl-plane.bc
infer-components message-connectivity
```

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

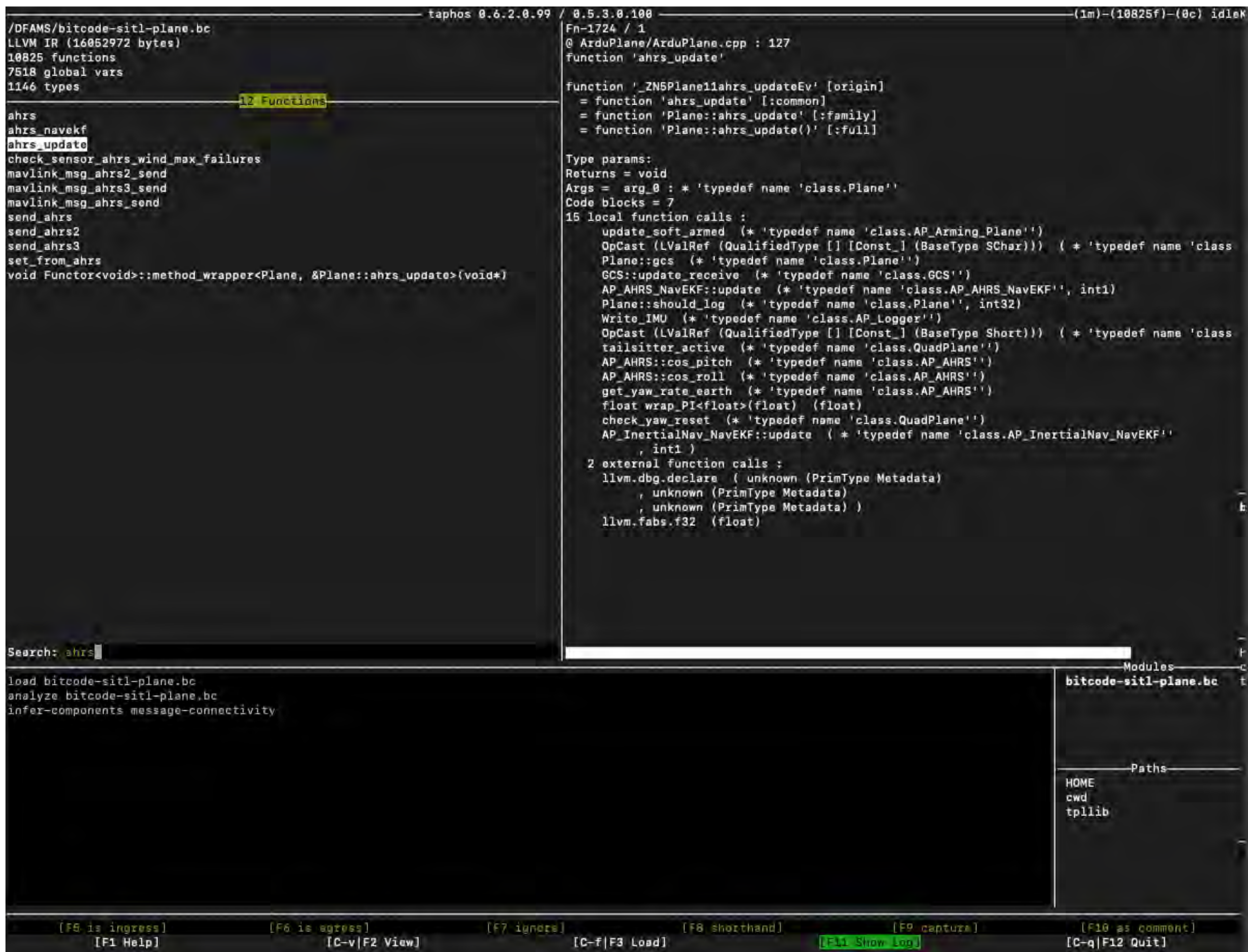


Figure 7.4 Taphos User Interface with `ahrs_update` Function Selected

Listing 7.3 Command to Write Bitcode to JSON

```
write bitcode-sitl-plane.bc > JSON bitcode-sitl-plane.json
```

With the Taphos call radius tool, we can analyze incoming and outgoing dependencies on individual functions, such as the scheduled task functions we consider as integration points for ELORAN functionality.

Table 7.1 Scheduled Tasks in Arduplane and their Callees at Various Radaii

Scheduled Task Functions (from ArduPlane.cpp)	Radius 1 Callees	Radius 2 callees	Radius 3 callees
ahrs_update	15	34	143
read_radio	16	53	78
update_speed_height	3	12	14

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Scheduled Task Functions (from ArduPlane.cpp)	Radius 1 Callees	Radius 2 callees	Radius 3 callees
update_GPS_50Hz	3	9	46
update_GPS_10Hz	8	57	87
update_alt	17	74	127
navigate	5	22	49

From call radius information we can see that `ahrs_update` and `update_alt` may be high-risk places to add new functionality, as they already have relatively high connectivity to the rest of the code base (e.g., `ahrs_update` depends on 15 functions directly, 34 functions at degree 2, and 143 functions at degree 3). Any update to the behavior of `ahrs_update` could affect the context in which these 143 other functions are invoked, incurring risk and requiring testing.

Conversely, `update_speed_height`, `update_GPS_50Hz`, `update_GPS_10Hz`, and `navigate` all appear to be viable candidates due to their relatively low number of dependencies. However, the number of software dependencies is only one measure of possible change impact.

Runtime performance impact

We opted to use the schedulability information we developed for notional domain separation in Part 6 to evaluate schedulability change impact of adding ELORAN parsing code. The Ardupilot uses a round robin schedule, but analysis of ARINC653 schedulability is still an interesting measure (and, it turns out, applicable in detecting problematic changes!).

Suppose new code for processing ELORAN input data has an execution time of approximately 4 milliseconds. As described in Part 6, the SuperVolo model contains detailed information about all of the software threads running in the SuperVolo and their execution constraints.

As the navigation functionality of ELORAN is similar to that of GPS, we first considered incorporating ELORAN functionality into the GPS processing thread, specifically the 50hz thread. The GPS 50hz thread, as shown in the following figure, executes at 50hz (equivalent to a period of 20ms).

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

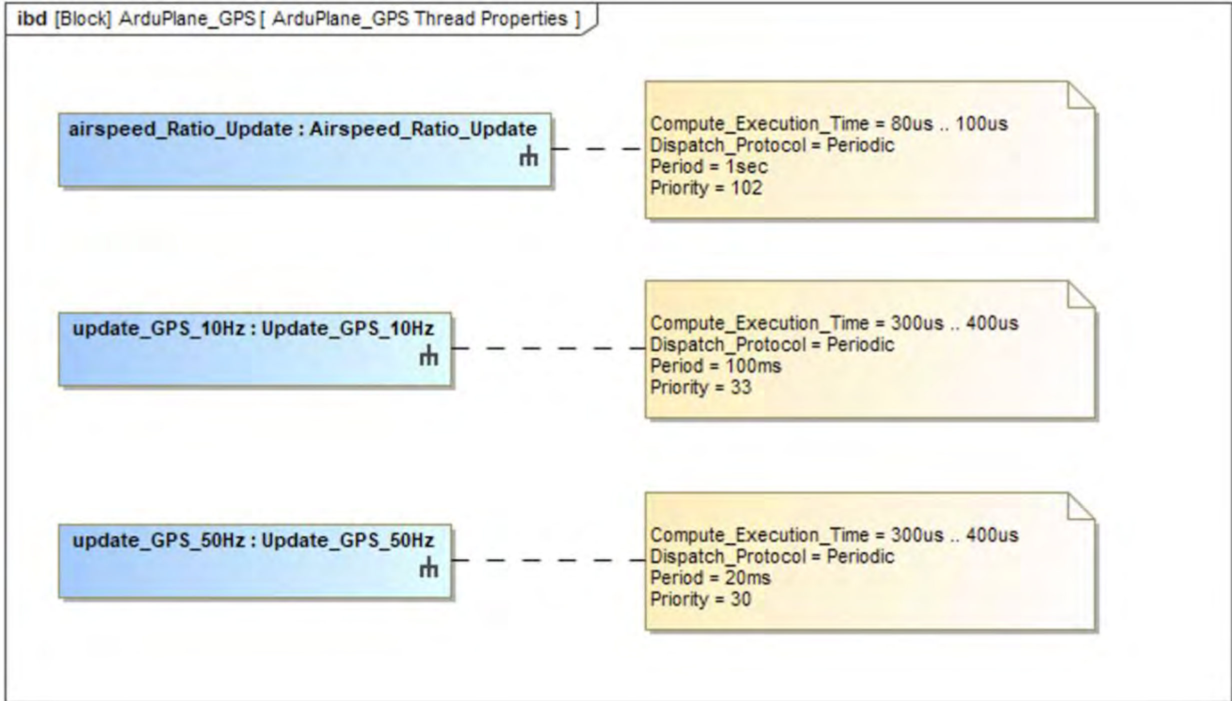
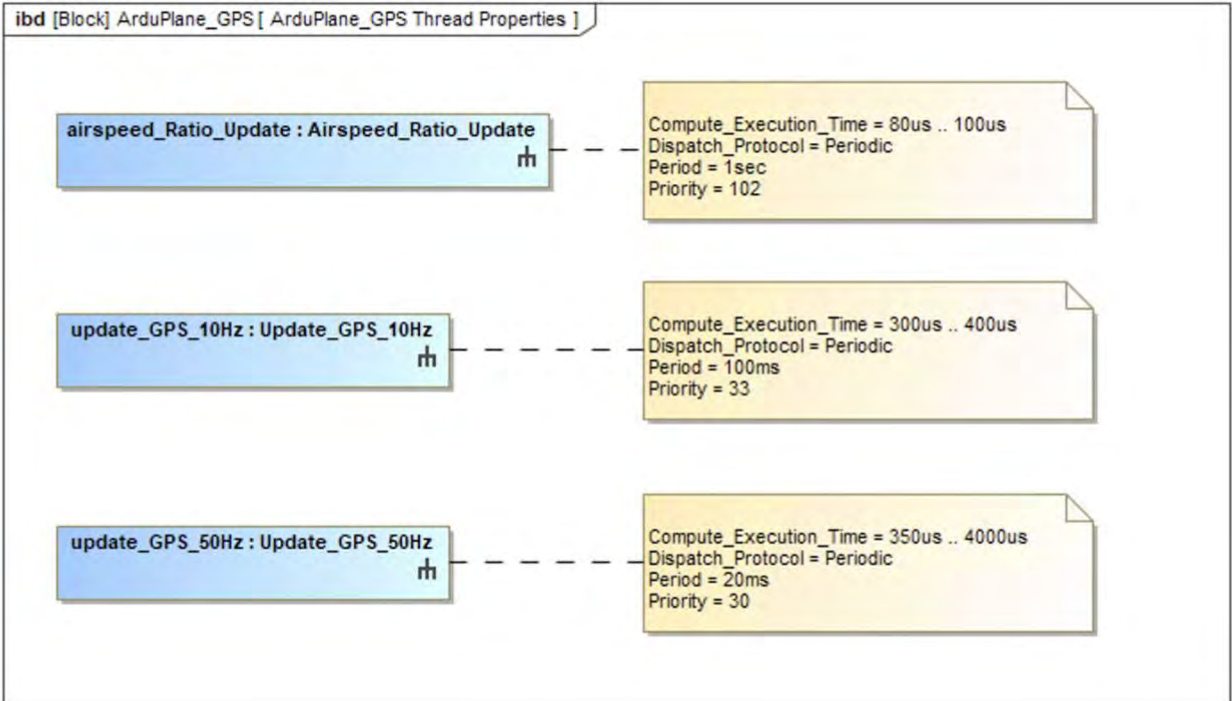


Figure 7.5 The GPS Thread Properties, before addition of ELORAN processing. Note that the actual properties are on the type definition, not its usage as a part.



This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Figure 7.6 The GPS Thread Properties shown in SysML, after addition of ELORAN processing. Note that the actual properties are on the type definition, not its usage as a part.

Considered in isolation, the GPS 50hz thread appears to be a reasonable place to add processing for additional navigation information. Naively, one might think that running a `4ms` task at `50hz` leaves plenty of processing slack - at worst that would take 20% of the available CPU time.

We updated the model (shown in the figure above) to reflect the increased processing time of 4ms. Using Galois's CAMET FASTAR tool, we tried schedulability analysis using a previously generated ARINC653 schedule (from Part 6). This schedulability analysis showed that we could not add this additional processing time to the 50hz GPS thread (or *any* thread in the GPS partition) without re-generating the overall processing schedule:

Listing 7.4 Output from FASTAR Schedule Analysis

```
PixelHawk.GSP_Partition: Partition load 20.51% on core 0 exceeds its
total window capacity 8.50%, cannot possibly be feasible.
```

Time partitioning is an important change containment strategy - if we can make a change and maintain an existing processing schedule, we significantly reduce change impact propagation due to computing resource competition, reducing certification risk. Conversely, if a change requires modifying a processing schedule, we significantly *increase* certification risk, since updates to a schedule almost always require full re-qualification.

We analyzed the updated configuration with Galois's CAMET FASTAR schedule generator (used in Part 6 to generate an execution schedule for the Ardupilot software). The schedule generator was *unable* to find a workable schedule for Ardupilot that accounted for this increased execution time of the GPS thread.

When we made this change in code and executed it on the PSIL, we observed the same result as our model-based analysis: the ELORAN processing code increased the execution time of the 50hz GPS thread too much and other threads were starved.

Thus we demonstrated with model-based analysis and validated in the PSIL that the 50hz GPS thread was not a viable place to insert new functionality. Instead, we selected an alternative thread that aggregates Altitude, Reference, and Heading System (ARHS) data. In this exercise, we only compared timing characteristics to decide where to place the ELORAN functionality. In practice, a larger engineering team would have evaluated and analyzed several design properties, such as security, timing, and qualification requirements to design the upgrade.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Contain: Generate and Integrate Code

Automated code generation is a containment mechanism because it limits the injection of defects through human error to the design phase (rather than allowing defects at the design phase and implementation phase). Code generation also limits the potential for drift between design and implementation.

Generating Source Code with TangramPro

The list of source code generated from the TangramPro file is shown in Figure 7.7, which includes:

- Functions for serializing/de-serializing WOSA navigation messages,
- Functions for transmitting messages using a specific logical bus protocol,
- Support to compile the generated code,
- Additional programs to run and test the generated code.

For our SuperVolo demonstration, we leveraged the existing shared memory infrastructure to transmit messages between scheduled tasks, so the second item in the list of above was not used. Similarly, we used the existing ArduPilot build environment, so the third item on the list was also not used. The WOSA source code itself is marked as CUI and is not included in this document.[60]

Integration of the generated code into the ArduPilot environment was relatively turn-key with a few minor manual adjustments. The integration steps include:

- Flatten the generated TrangramPro code into a single directory, to meet the ArduPilot build restrictions,
- Add TangramPro generated code to the ArduPilot build environment,
- Add the function calls to serialize/de-serialize the WOSA messages at the entry-points in the selected ArduPilot avionics tasks
- Add two specific C-compiler flags required by the TangramPro code to the appropriate ArduPilot Makefiles.

[60] For further details and information on TangramPro, see https://docs.tangramflex.io/user_manual/feature_overview.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

code-gen

Name	Date Modified	Size	Kind
beautify.sh	Aug 30, 2024 at 9:25 AM	229 bytes	Shell Script
doc	Aug 30, 2024 at 9:25 AM	—	Folder
Doxyfile	Aug 30, 2024 at 9:25 AM	10 KB	Document
Dockerfile	Sep 3, 2024 at 1:38 PM	1 KB	Document
ffi	Aug 30, 2024 at 9:25 AM	—	Folder
include	Aug 30, 2024 at 9:25 AM	—	Folder
src	Aug 30, 2024 at 9:25 AM	—	Folder
include	Today at 9:18 AM	—	Folder
GenericMessage.hpp	Aug 30, 2024 at 9:25 AM	511 bytes	C++ H...r Source
wosa_v3_1	Aug 30, 2024 at 9:25 AM	—	Folder
DMN_Status_Monitor.hpp	Aug 30, 2024 at 9:25 AM	18 KB	C++ H...r Source
Header.hpp	Aug 30, 2024 at 9:25 AM	12 KB	C++ H...r Source
LAN_Status_Monitor.hpp	Aug 30, 2024 at 9:25 AM	3 KB	C++ H...r Source
Trailer.hpp	Aug 30, 2024 at 9:25 AM	5 KB	C++ H...r Source
wosa_v3_1_DerivedEntityFactory.hpp	Aug 30, 2024 at 9:25 AM	5 KB	C++ H...r Source
local_install	Sep 3, 2024 at 1:25 PM	—	Folder
lock.package.toml	Aug 30, 2024 at 9:25 AM	358 bytes	Document
Makefile	Aug 30, 2024 at 9:25 AM	6 KB	Makefile
package.toml	Aug 30, 2024 at 9:25 AM	804 bytes	Document
pkg_deps	Today at 9:18 AM	—	Folder
genericapi	Sep 4, 2024 at 4:40 PM	—	Folder
beautify.sh	Aug 30, 2024 at 9:25 AM	2 KB	Shell Script
doc	Aug 30, 2024 at 9:25 AM	—	Folder
ffi	Aug 30, 2024 at 9:25 AM	—	Folder
gdb	Aug 30, 2024 at 9:25 AM	—	Folder
include	Aug 30, 2024 at 9:25 AM	—	Folder
io	Aug 30, 2024 at 9:25 AM	—	Folder
Makefile	Aug 30, 2024 at 9:25 AM	8 KB	Makefile
non-failing-cppcheck-errors.txt	Aug 30, 2024 at 9:25 AM	66 bytes	Plain Text
package.toml	Aug 30, 2024 at 9:25 AM	3 KB	Document
pkg_deps	Aug 30, 2024 at 9:25 AM	—	Folder
pkg_deps_report.json	Aug 30, 2024 at 9:25 AM	871 KB	JSON Document
pkg_deps.make	Aug 30, 2024 at 9:25 AM	1 KB	Makefile
populaters	Aug 30, 2024 at 9:25 AM	—	Folder
README.md	Aug 30, 2024 at 9:25 AM	50 KB	Markdo...ument
serializers	Sep 5, 2024 at 8:14 AM	—	Folder
src	Aug 30, 2024 at 9:25 AM	—	Folder
tests	Aug 30, 2024 at 9:25 AM	—	Folder
variable.mk	Aug 30, 2024 at 9:25 AM	10 KB	Makefile
xml	Aug 30, 2024 at 9:25 AM	—	Folder
pkg_deps_report.json	Aug 30, 2024 at 9:25 AM	893 KB	JSON Document
pkg_deps.make	Aug 30, 2024 at 9:25 AM	4 KB	Makefile
README.md	Sep 4, 2024 at 9:33 AM	28 KB	Markdo...ument
src	Today at 9:18 AM	—	Folder
GenericMessage.cpp	Aug 30, 2024 at 9:25 AM	413 bytes	C++ Source
Makefile	Aug 30, 2024 at 9:25 AM	331 bytes	Makefile
wosa_v3_1	Aug 30, 2024 at 9:25 AM	—	Folder
DMN_Status_Monitor.cpp	Aug 30, 2024 at 9:25 AM	17 KB	C++ Source
Header.cpp	Aug 30, 2024 at 9:25 AM	9 KB	C++ Source
LAN_Status_Monitor.cpp	Aug 30, 2024 at 9:25 AM	2 KB	C++ Source
Makefile	Aug 30, 2024 at 9:25 AM	233 bytes	Makefile
Trailer.cpp	Aug 30, 2024 at 9:25 AM	4 KB	C++ Source
wosa_v3_1_DerivedEntityFactory.cpp	Aug 30, 2024 at 9:25 AM	7 KB	C++ Source
swig	Aug 30, 2024 at 9:25 AM	—	Folder
Makefile	Aug 30, 2024 at 9:25 AM	521 bytes	Makefile
tangramgenericswig.i	Aug 30, 2024 at 9:25 AM	407 bytes	Plain Text
utility_swig.i	Aug 30, 2024 at 9:25 AM	1 KB	Plain Text
wosa_v3_1	Aug 30, 2024 at 9:25 AM	—	Folder
wosa_v3_1_DerivedEntityFactory_swig.i	Aug 30, 2024 at 9:25 AM	313 bytes	Plain Text
test	Sep 10, 2024 at 7:30 AM	—	Folder
deserialization	Aug 30, 2024 at 9:25 AM	—	Folder
end-to-end	Oct 21, 2024 at 12:36 PM	—	Folder
property	Aug 30, 2024 at 9:25 AM	—	Folder
runway	Aug 30, 2024 at 9:25 AM	—	Folder
serialization	Aug 30, 2024 at 9:25 AM	—	Folder
swig	Aug 30, 2024 at 9:25 AM	—	Folder

Figure 7.7 The list of source code files generated by TangramPro from the simple messaging model.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Using Tangram Pro to generate source code, we were able to eliminate potential change impact originating from the human authorship of source code. By using Tangram Pro in conjunction with the WOSA standard, we were further able to reduce potential change impact from non-standard message formats, preparing the software for inclusion of further WOSA standardized messages in the future with little change impact.

Conclusion

In this scenario we added a new physical device (a software defined radio) to the SuperVolo architecture. We used a combination of schedulability analysis and hardware in the loop testing with the PSIL to evaluate potential locations in the Ardupilot software for new content to select the location with acceptable change impact. We used Tangram Pro to automatically generate standards-based source code to reduce change impact from human source code writing and position the architecture for low impact changes in the future by leveraging interoperability standards.

Conclusion

In this book we introduced the process of Frame, Model, Analyze, and Contain (FMAC) for making impact-informed design decisions to cyber-physical systems. The process consists of framing the stakeholder needs, modeling the relevant artifacts, analyzing the impact of proposed changes using those artifacts, and making strategic design decisions that contain the impact of change. We walked through three scenarios in which we exercise the FMAC process for changes to the SuperVolo system.

Appendices

A. Acronyms

- AADL - Architecture Analysis and Design Language
- ABI - Application Binary Interface
- ACVIP - Architecture Centric Virtual Integration Process
- ARINC - Aeronautical Radio Incorporated
- CAD - Computer Aided Design
- CIA - Confidentiality, Integrity, Availability
- DSM - Design Structure Matrix
- FADEC - Full Authority Digital Engine Control
- FMAC - Frame, Model, Analyze, Contain
- FMET - Failure Modes and Effects Testing
- GPS - Global Positioning System

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

- GRA - Government Reference Architecture
- LORAN - Long Range Navigation
- MOSA - Modular Open Systems Approach
- PSIL - Portable System Integration Lab
- RDE - Rigorous Digital Engineering
- SDR - Software Defined Radio
- SIL - System Integration Lab
- SWaP - Size Weight and Power
- SysML - Systems Modeling Language
- TRL - Technology Readiness Level
- UML - Unified Modeling Language
- WOSA - Weapons Open Systems Architecture

B. Bibliography

Works Cited

Adventium Labs. *ACVIP Modeling and Analysis Handbook*. Adventium Labs, 2021, https://cdn.prod.website-files.com/673b407e535dbf3b547179dd/6786ed77930332b4ced3988c_ACVIP-Modeling-%26-Analysis-Handbook_Mar2021_DistA.pdf.

"Airworthiness & Airworthiness Certification AETM 002." *Airworthiness & Airworthiness Certification AETM 002*, Defense Acquisition University, https://www.dau.edu/acquipedia-article/airworthiness-airworthiness-certification#:~:text=Changes%20that%20affect%20the%20prescribed,authorized%20in%20the%20technical%20manual. Accessed 3 February 2025.

Amundson, Isaac, et al. "Loonwerks." *Loonwerks*, Collins Aerospace, https://loonwerks.com/. Accessed 31 January 2024.

Bjørner, Dines, and Yang ShaoFa. *Domain Modelling – A Primer*. Dines Bjørner & Yang ShaoFa, 2024. *https://www.imm.dtu.dk/~dibj/2024/primer/primer.pdf*.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Boydston, Alex, et al. “Architecture Centric Virtual Integration Process (ACVIP): A Key Component of the DoD Digital Engineering Strategy.” https://insights.sei.cmu.edu/documents/1510/2019_021_001_634975.pdf. Accessed 31 January 2025.

Delange, Julien. *AADL in Practice*. Reblochon Development Company, 2017. Accessed 31 January 2025.

Department of Defense. “DoD Reference Architecture Description.” *DoD Reference Architecture Description*, Department of Defense, June 2010, https://dodcio.defense.gov/Portals/0/Documents/Ref_Archi_Description_Final_v1_18Jun10.pdf. Accessed 31 January 2025.

DEVCOM Aviation and Missile Center. *Army Military Airworthiness Certification Criteria (AMACC) Documents and SysML Model Libraries*. U.S. Army, 2024, https://www.avmc.army.mil/Directorates/SRD/TechDataMgmt/. Accessed 31 January 2025.

Eppinger, Steven D., and Tyson R. Browning. *Design Structure Matrix Methods and Applications*. MIT Press, 2012. Accessed 31 January 2025.

Fowler, Martin. *UML distilled*. Addison-Wesley, 2004. Accessed 31 January 2025.

Hansson, Jörgen, et al. “ROI Analysis of the System Architecture Virtual Integration Initiative.” 2018. *CMU SEI Insights*, https://insights.sei.cmu.edu/library/roi-analysis-of-the-system-architecture-virtual-integration-initiative/. Accessed 31 January 2025.

Kiniry, Joe, et al. “The HARDENS Final Report.” *The HARDENS Final Report*, Office of Nuclear Regulatory Research, 2024, https://www.nrc.gov/docs/ML2232/ML22326A307.pdf. Accessed 31 January 2025.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Krebs, Brian. “Global Microsoft Meltdown Tied to Bad Crowdstrike Update – Krebs on Security.” *Krebs on Security*, 19 July 2024, https://krebsonsecurity.com/2024/07/global-microsoft-meltdown-tied-to-bad-crowstrike-update/. Accessed 31 January 2025.

NASA. “FRET : Formal Requirements Elicitation Tool (ARC-18066-1).” *NASA Technology Transfer Program*, NASA, https://software.nasa.gov/software/ARC-18066-1. Accessed 31 January 2025.

NIST. “NIST Special Publication (SP) 800-160 Vol. 2 Rev. 1, Developing Cyber-Resilient Systems: A Systems Security Engineering Approach.” *NIST Computer Security Resource Center*, 9 December 2021, https://csrc.nist.gov/pubs/sp/800/160/v2/r1/final. Accessed 31 January 2025.

“Remediation and Guidance Hub: Channel File 291 Incident.” *Crowdstrike Remediation*, Crowdstrike, 2024, https://www.crowdstrike.com/falcon-content-update-remediation-and-guidance-hub/. Accessed 31 January 2025.

SAE International. *Architecture Analysis and Design Language*. 2022. *Architecture Analysis & Design Language (AADL) AS5506D*, SAE International, https://www.sae.org/standards/content/as5506d/. Accessed 31 January 2025.

C. V-MESA GPS Analysis

Introduction

On V-MESA, we developed new automated capabilities for performing vulnerability assessments and deriving behavioral models of deployed embedded systems and systems of systems. We extended and matured technology that automatically analyzes the input/output behavior of networked systems using a model based on timed automata. This enables one to model the behavior and investigate potential vulnerabilities as depicted in Figure C.1.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

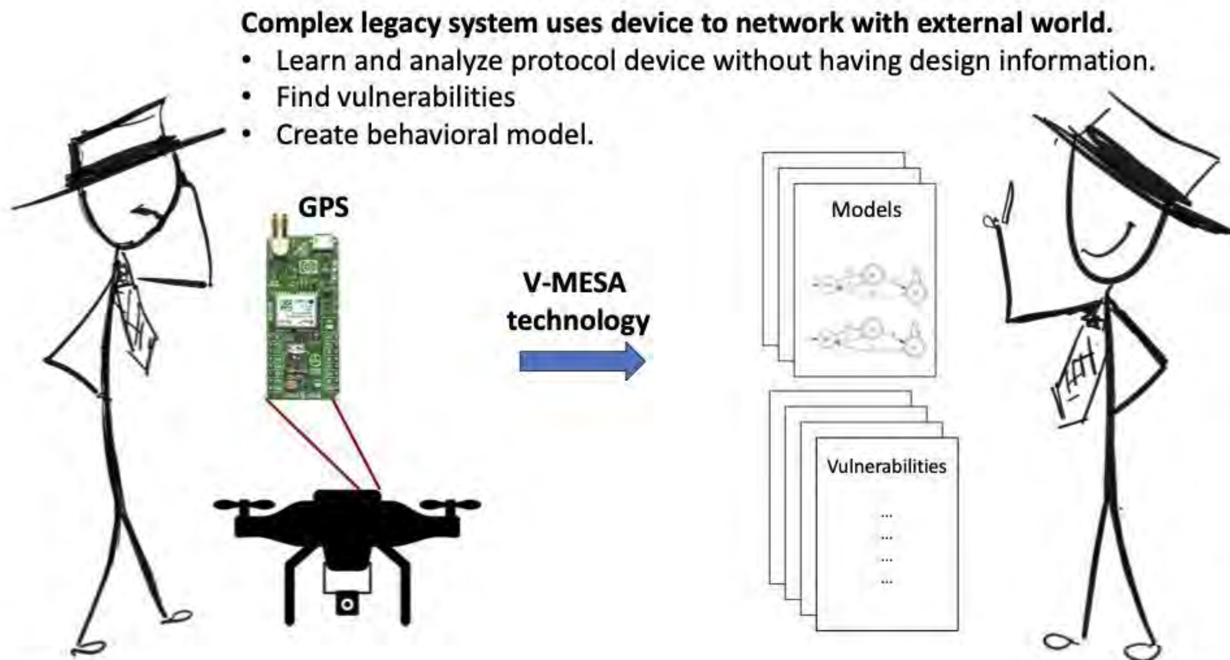


Figure C.1: V-MESA provides a way to automatically derive a behavioral model of a device and identify vulnerabilities within the networked system.

Security analysts presented with a device to investigate typically have some information regarding the device's context and function. However, they may have little, if any, reliable information regarding its architecture and inner workings. Embedded systems pose particular difficulties due to the presence of clocks and other continuous processes, many of them mediated through interaction with the physical world, as well as the widespread use of custom architectures and software platforms to implement those systems. V-MESA bridges this gap, using an explicit threat model and targeted device analysis to build a behavioral model of the embedded system and automatically identify vulnerabilities. In this way, V-MESA provides analysts with a powerful tool for improving the security and dependability of systems and the missions they support.

We originally developed a timed automata-based vulnerability analysis approach based on learning *timed automata* on the Vulnerabilities Out of Learned Time Automata (VOLTA) program. An automata is an abstract computing “device," which automatically follows a predetermined sequence of operations. It is an “active" model of states, transitions, and behavior. Timed automata introduces timing into the behavior taken by the automata. On VOLTA we evaluated this timed automata approach on four qualitatively different, commercially available embedded devices. At the end of the VOLTA program, the technology was a Technology Readiness Level (TRL) 4 prototype system capable of analyzing vulnerabilities, including timing vulnerabilities, in networked embedded systems.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

On this VOLTA Maturation for the Air Force (AF) / Department of Defense (DoD) Embedded Systems Analysis (V-MESA) project, we have adapted and extended VOLTA results to provide an effective tool that fills a critical analysis gap in vulnerability analysis of embedded physical systems. Over the course of the V-MESA project, we have extended and matured the VOLTA learning algorithm and associated tools, improved workflow and usability, extended learning capabilities to analyze commonly used protocols, and worked towards transition into legacy avionics systems.

SuperVolo GPS Results

Digital Future Acquisition, Maintenance, and Sustainment (DFAMS) was a government-funded effort to provide a proof-of-concept demonstration of digital engineering for legacy airframes. In partnership with Galois's ExistX spinoff, DFAMS applied Galois's Curated Access to Model-based Engineering Tools (CAMET) library of Model-Based Systems Engineering (MBSE) tools to reconfiguring and retrofitting avionics components. The Taphos tool, developed on DARPA Verified Security and Performance Enhancement of Large Legacy Software (V-SPELLS) program, is then used for further analysis. The existing tools largely focus on structural design. The system under analysis is the SuperVolo long-range hybrid gas-electric Unmanned Aerial Vehicle (UAV), as shown in
Figure C.2.

We have access to a SuperVolo at Galois's Minneapolis office. SuperVolo built a SIL (see Figure C.3) composed of a subset of the SuperVolo's components:

- Flight control processor - Orange Cube+
- GPS - u-blox NEO-M9N-00B
- Carrier board - ADSB Orange Cube carrier board
- Power distribution - Orange Cube power brick
- Battery - 1500 mAh LiPo battery
- Buzzer
- Air Surface Servo
- Rotor Servo

We investigated the Global Positioning System (GPS) component as an System Under Test (SUT) to demonstrate V-MESA's ability to characterize behavior, in particular, timed behavior of avionics components. The GPS unit supports multiple timing protocols, so we identified what subset of protocols are used in the SuperVolo UAV.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Figure C.2: The SuperVolo unmanned aerial vehicle.

The GPS is a satellite-based radio navigation system that provides geolocation and time information to a GPS receiver. The GPS module that we investigated is the GPS `u-blox NEO-M9N module1`. The NEO-M9N receiver provides all the necessary Radio Frequency (RF) and baseband processing to enable multi-constellation operation. The block diagram displayed in Figure C.4 shows the key functionality that was important for our analyses.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

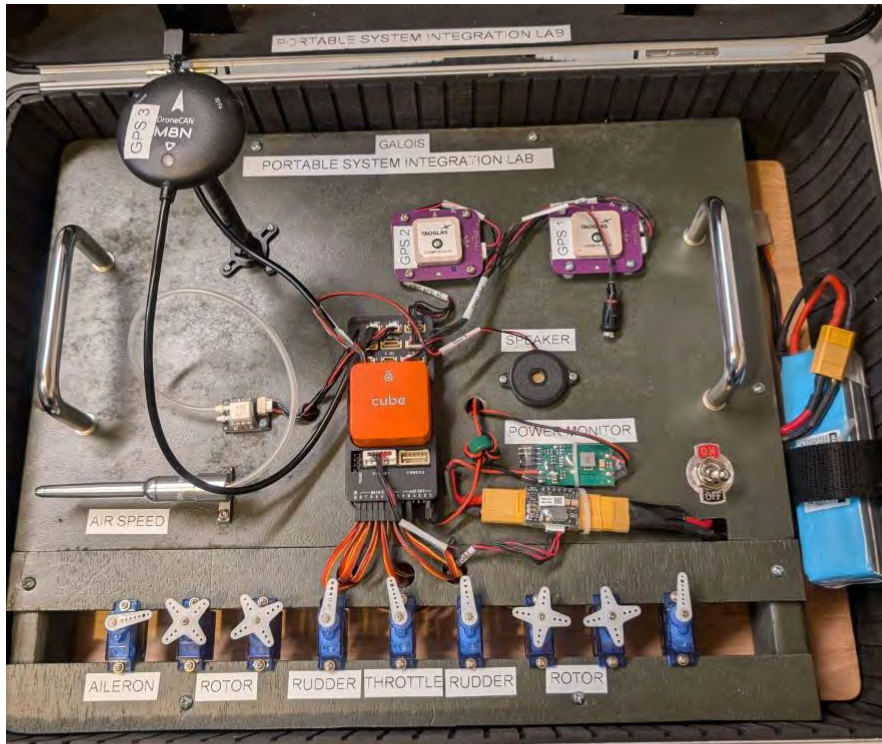


Figure C.3: SuperVolo SIL.

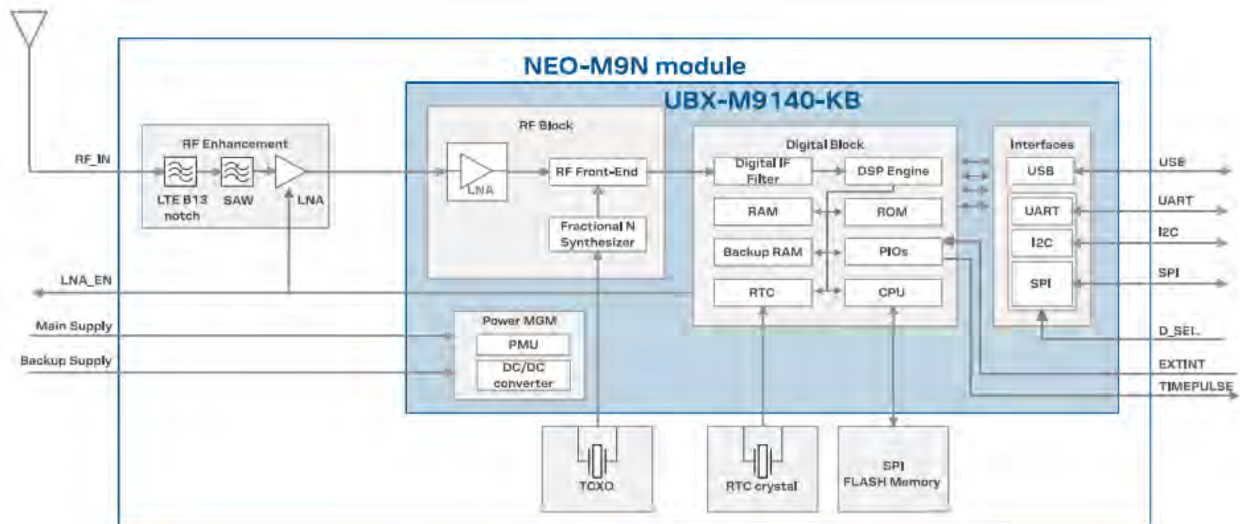


Figure C.4: NEO-M9N block diagram.

We applied the VOLTA learner to two versions of the NEO UBX GPS device to show how model learning can be used to identify differences in behavior. The physical setup of the learner is shown in Figure C.5.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

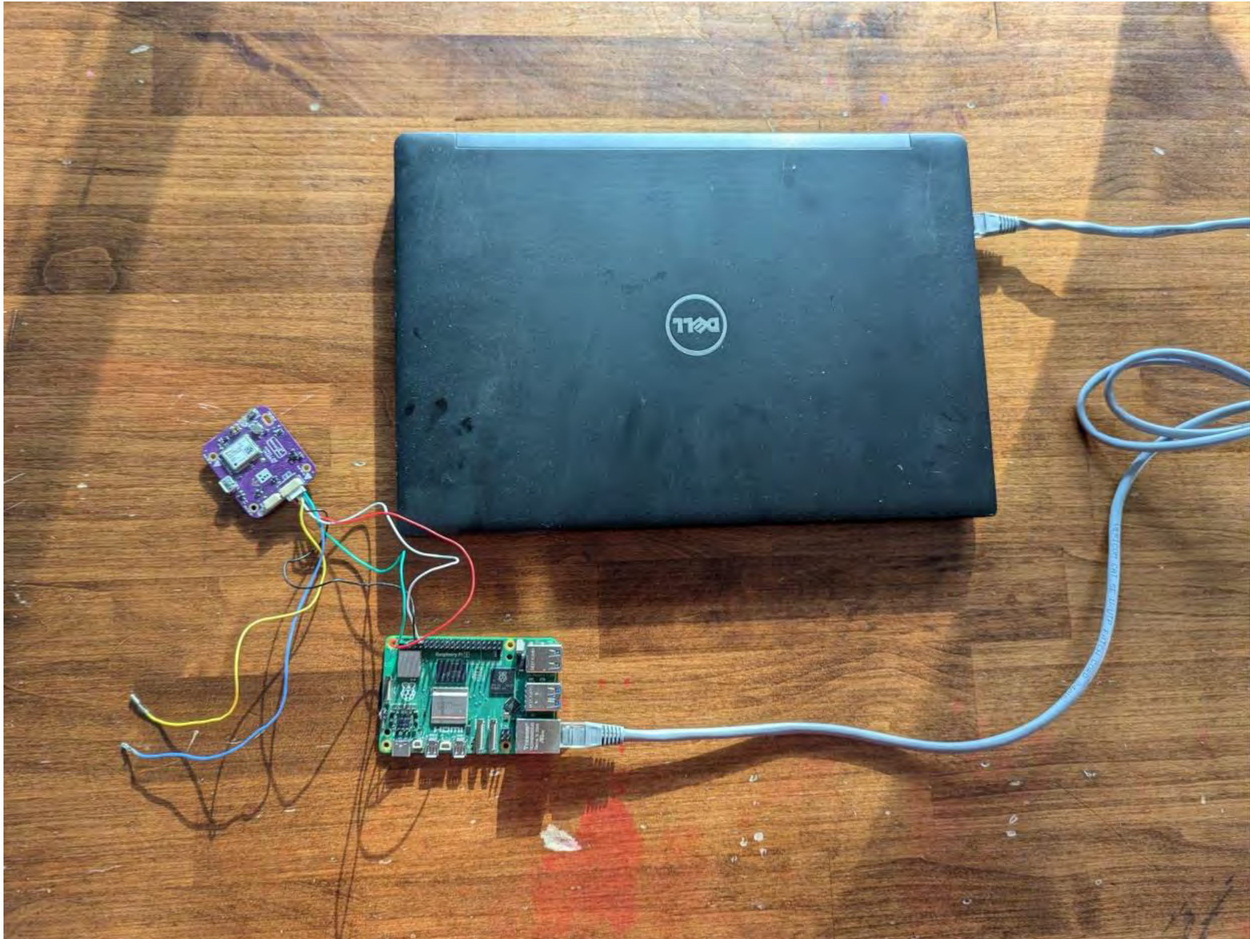


Figure C.5: The physical setup of the GPS learner.

We analysed the UBX protocol of the devices via their serial port interfaces. The learned state machines for the NEO M8 and M9 are shown in Figure C.6 and Figure C.7.

Volta learns a limited form of timed finite state machine that has a single countdown timer. In such a machine, output can only happen when this countdown timer expires. For the subset of the GPS protocol we analyzed, no interesting timer behaviour was observed. All outputs happened immediately after some input. These state machine diagrams can be interpreted as follows:

- Black single line circles are states with no timer running. Output cannot happen in
- such a state.
- Green double line circles are states where the timer is running. There will be exactly one possible output represented with a single outgoing output transition. The output occurs if the timer expires before leaving the state.
- The initial state has an incoming transition with no source.
- Black edges represent input transitions.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

- Green edges with a small circle at the source end represent output transitions.
- Transitions are labeled with an alphabet symbol, optionally followed by a colon and an integer indicating timer setting. A timer setting of zero indicates that the output in the target state will occur immediately.
- Output alphabet symbols are conventionally prefixed with “o" to distinguish them from input symbols.

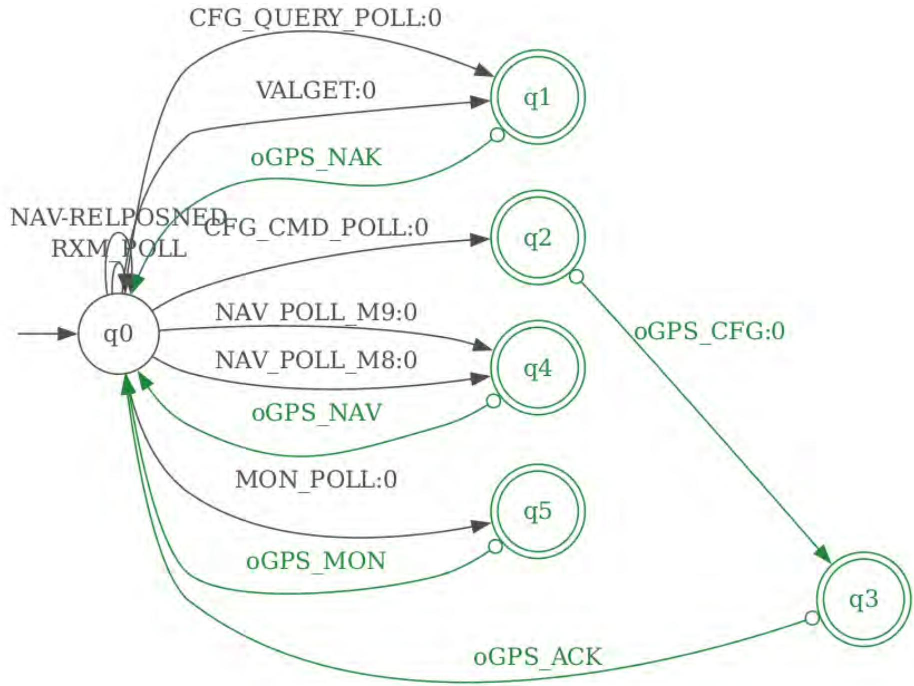
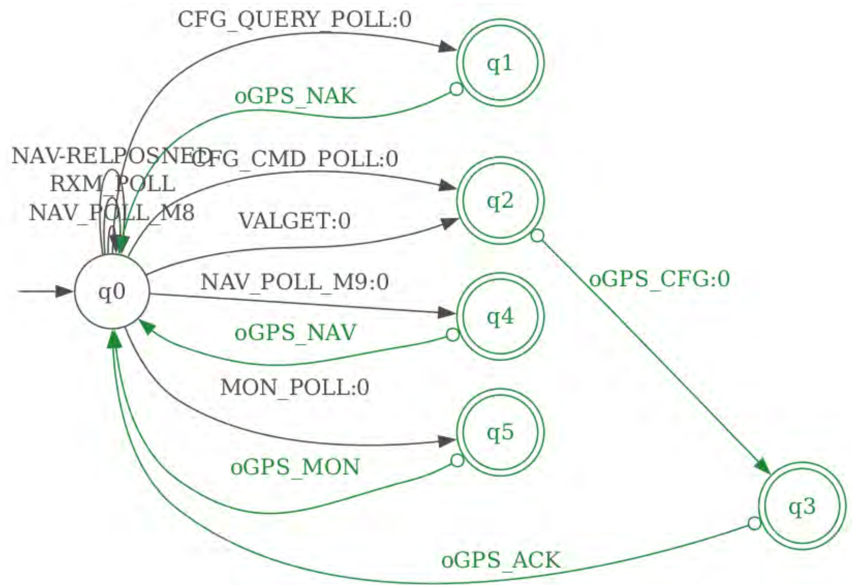


Figure C.6: The state diagram of the NEO M8 UBX device.



This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

Figure C.7: The state diagram of the NEO M9 UBX device.

The differences that can be seen in the M8 and M9 state diagrams are:

1. The M9 device supports VALGET messages while the M8 does not. We see this as different paths in the state diagrams:
   `M8: VALGET -> oGPS_NAK`
   `M9: VALGET -> oGPS_CFG -> oGPS_ACK`
2. The M8 device supports the NAV M8 messages while the M9 does not. We see this as different paths in the state diagrams:
   `M8: NAV_POL_M8 -> oGPS_NAV`
   `M9: NAV_POL_M8 -> no response`

While manual effort is required to interpret the resulting state machines, these artifacts offer valuable insights into the otherwise opaque GPS component. If the code assumes that the Ardupilot supports all message types, this may result in a security risk or unhandled software bug. If no safeguard is in place to handle such cases, Ardupilot may fail to recover when a different GPS version is used. However, we have not yet analyzed the code to determine if this assumption has been made.

VOLTA Learner Deployment

This section provides a general overview of how to deploy the VOLTA learner to learn a device's behaviour. The next section will discuss specifics of the deployment for the GPS device. The VOLTA learner is written as a JAVA library and deploying it requires writing Java code. The device we desire to learn is referred to as the System Under Learning (SUL). We need to specify the input symbols that can be used to stimulate the SUL and the output symbols that encode responses observed from the SUL. The union of these input and output symbols is referred to as the SUL alphabet. The alphabet must be defined as a Java type that can be used as a generic type parameter when instantiating the `com.adventiumlabs.volta.thl.HLearner<IO>` class. It can be a simple type such as String, or a Java Class which can be as complex as needed.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

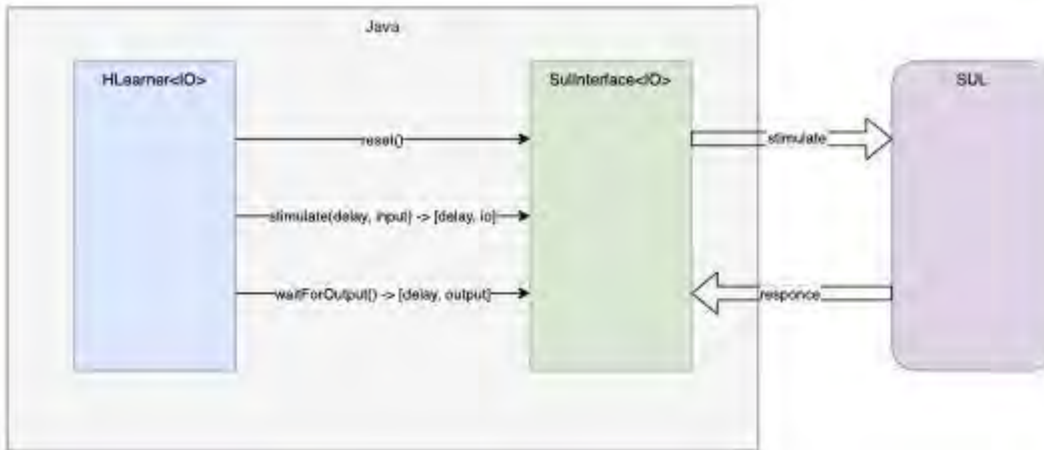


Figure C.8: Basic Learner Deployment.

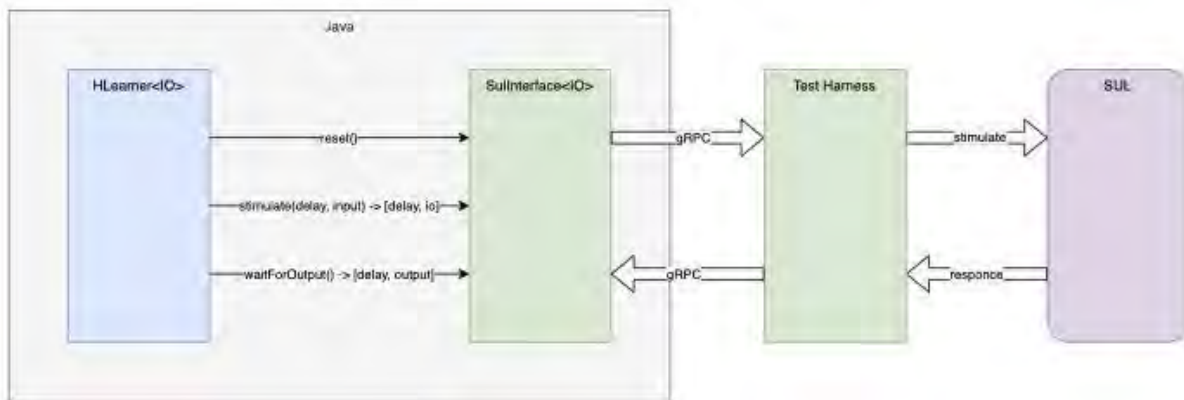


Figure C.9: gRPC Test Harness Learner Deployment.

We also need to implement the mechanisms that will stimulate and observe the SUL. This is done by implementing the `com.adventiumlabs.volta.thl.SulInterface<IO>` interface. See Figure C.8 with the green box indicating the user supplied code. The SUL interface is responsible for mapping input alphabet symbols into SUL stimuli and SUL responses into output alphabet symbols.

In many learner deployments it is convenient to have a test harness independent of Java that stimulates and observes the SUL. For example, communicating with the SUL may be easier from another language, perhaps due to available libraries. Or the SUL may need to be connected to a device that is not able to run Java. gRPC, an open-source remote procedure call (RPC) framework developed by Google, is a convenient mechanism to facilitate such deployments as shown in Figure 9.

The three methods that the `SulInterface<IO>` must implement are:

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

- `reset()` State machines must have a defined initial state. This method must force the SUL into this initial state.
- `stimulate(delay, input) -> [delay, io]` This method must stimulate the SUL as indicated by the input alphabet symbol after a delay. If successful the delay and input are returned. If however, an output is observed during the delay, the output and its observed delay are returned.
- `waitForOutput() -> [delay, output]` This method waits for a response from the SUL. If a response is observed, it is translated into an output alphabet symbol and returned along with the observed delay. If no output is observed after a configured amount of time, known as the quiescent delay, null is returned.

Delays are unitless. The `SulInterface<IO>` is responsible for mapping realtime measurements to delays uniformly. It is important to design this mapping so that delays are deterministic. The learner can only learn deterministic behaviour.

GPS Learner Details

This section describes how we deployed the VOLTA learner to learn the GPS device behavior. There are two main implementation tasks with any VOLTA learner deployment; the alphabet and the SUL interface.

We identified the GPS message classes and IDs supported by Ardupilot by inspection of the source code. The findings are shown in Figure C.10. The alphabet symbols are in the MSG column. VOLTA has an abstraction mechanism to group multiple similar alphabet symbols to make state diagrams more readable. The GROUP column shows the abstracted symbols that actually appear in the GPS state diagrams.

For communication with the GPS device we used a Python library. Thus we used the gRPC Test Harness pattern described in the previous section to deploy the VOLTA learner.

There are three basic message types that we need to support. This includes when the learner initiates the message to the SUL, when the learner initiates an SUL reset, and when a message is sent from the SUL to the learner. We outline these message types below and provide event diagrams as illustration.

We can use this function to send any kind of message we like including messages that violate one or more rules of the protocol.

Learner initiates message to SUL. Using Google Remote Procedure Call (GRPC), the Learner sends a message to the SUL via the server. We can use this function to send any kind of message we like, including messages that violate one or more rules of the protocol

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

(Figure C.11).

Learner initiates SUL reset. Like all state machines, the state diagram we are building must have a well defied initial state and we must be able to reach this state on demand. The SUL reset command is used to force the device into this initial state. This is illustrated in Figure C.12. On the final message sent from Learner to SUL, the “Empty" designator is a null message that is used to signify completion and acknowledgement of communication.

Message from SUL sent to Learner. The state of the SUL is determined by what output messages are generated in response to messages sent from the Learner. This means all messages output by the SUL must be communicated to the Learner with a timestamp indicating when they were sent by the SUL. When the Server sends a message to the Learner in response to an output from the SUL and the message is received successfully, the Learner acknowledges receipt via an empty message. This is illustrated in Figure C.13.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

MSG	CLASS	ID	GROUP
UBX-ACK-ACK	x05	x01	ACK
UBX-ACK-NAK	x05	x00	NAK
UBX-CFG-CFG	x06	0x09	CFG_QUERY
UBX-CFG-MSG	x06	0x01	CFG_QUERY
UBX-CFG-GNSS	x06	0x3e	CFG_CMD
UBX-CFG-NAV5	x06	0x24	CFG_CMD
UBX-CFG-PRT	x06	0x00	CFG_CMD
UBX-CFG-RATE	x06	0x08	CFG_CMD
UBX-CFG-SBAS	x06	0x16	CFG_CMD
UBX-CFG-TP5	x06	0x31	CFG_CMD
UBX-CFG-VALGET	x06	0x8b	VALGET
UBX-CFG-VALSET	x06	0x8a	VALSET
NAV_POSLLH	x01	x02	NAV_M9
NAV_STATUS	x01	x03	NAV_M9
NAV_DOP	x01	x04	NAV_M9
NAV_PVT	x01	x07	NAV_M9
NAV_TIMEGPS	x01	x20	NAV_M9
NAV_VELNED	x01	x12	NAV_M9
NAV_RELPOSNED	x01	x3c	NAV-RELPOSNED
NAV_SOL	x01	x06	NAV_M8
NAV_NAV_SVINFO	x01	x30	NAV_M8
MSG_MON_HW	x0a	x09	MON
MSG_MON_HW2	x0a	x0b	MON
MSG_MON_VER	x0a	x04	MON
MSG_RXM_RAW	x02	x10	RXM
MSG_RXM_RAWX	x03	x15	RXM

Figure C.10: The message classes and IDs supported by Ardupilot as determined by inspection of the source code.

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

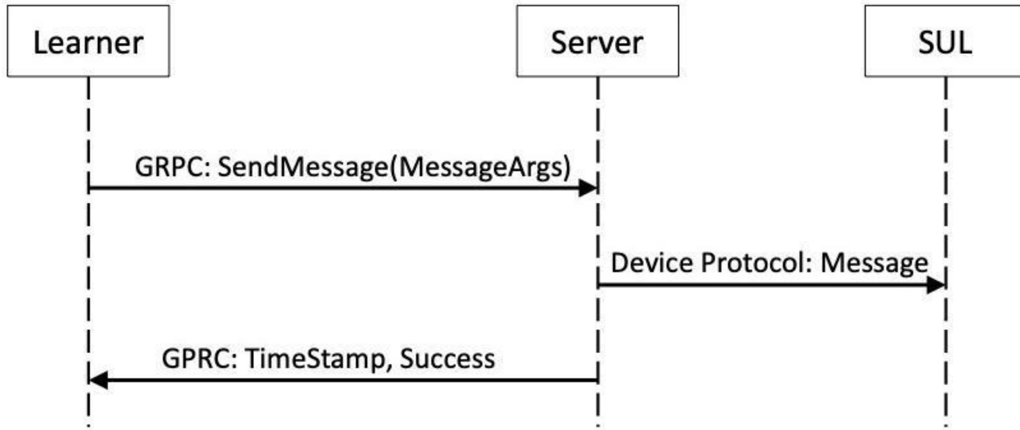


Figure C.11: Learner initiates message to the SUL.

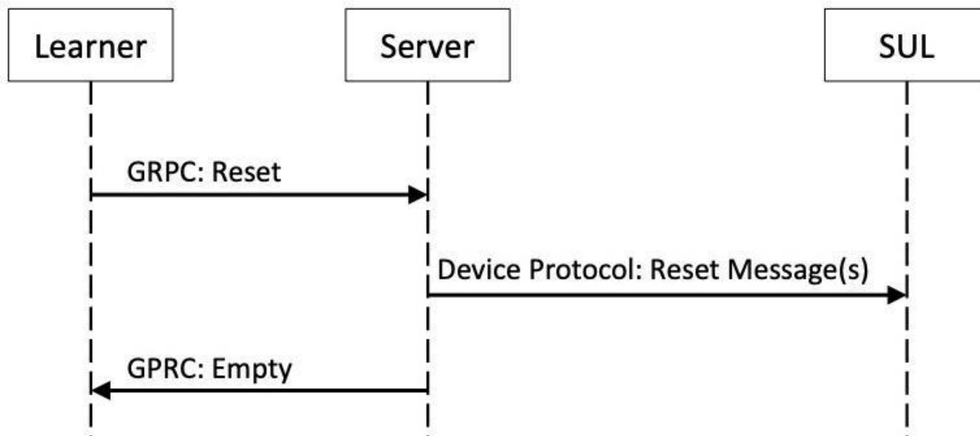


Figure C.12: Learner initiates SUL reset.

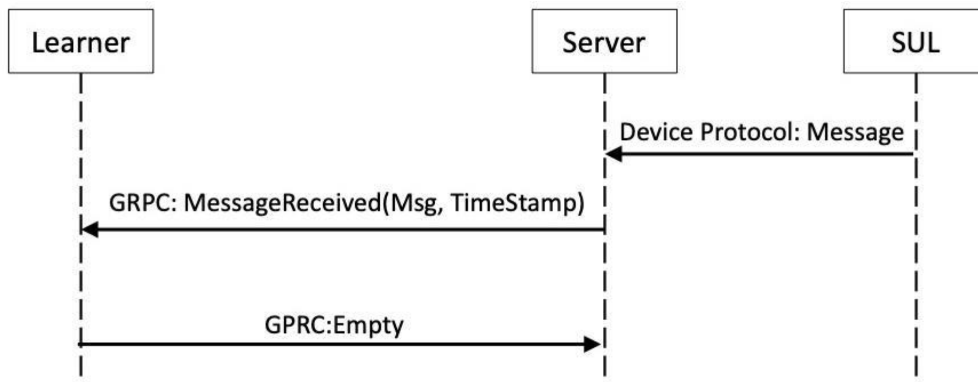


Figure C.13: Message from SUL sent to learner.

Each of these message types are defined in the following example `protobuf` file which is then

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

used to generate the Java and Python bindings used by the Learner and the Server. An example python server using the above bindings generated by the above proto would look like the code listed in Listing C.2. The class and methods were already declared in the bindings but the user must define their behavior.

Listing C.1: The `protobuf` file used to generate Java and Python bindings.

```
import "google/protobuf/empty.proto";
import "google/protobuf/timestamp.proto";

//Server: SUL (test harness)
//Client: Learner
service Sul {
  rpc SendMessage (MessageArgs) returns (SendResult) {}
  rpc Reset (google.protobuf.Empty) returns (google.protobuf.Empty) {}
}

//Server: Learner
//Client: SUL (test harness)
service Learner {
  rpc MessageReceived (MessageReceipt) returns (google.protobuf.Empty) {}
}

message MessageArgs {
  repeated string args = 1;
}

message MessageReceipt {
  google.protobuf.Timestamp receipt_time = 1;
  bytes full_message = 2;
}

message SendResult {
  google.protobuf.Timestamp send_time = 1;
  bool success = 2;
}
```

Listing C.2: An example server using the bindings described above.

```
import grpc
import example_pb2
import example_pb2_grpc
import google.protobuf.empty_pb2
from google.protobuf.timestamp_pb2 import Timestamp
import cool_device_protocol_library

class Sul(example_pb2_grpc.SulServicer):
      def SendMessage(self, request, context):
```

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025

```
    ...

    def Reset(self, request, context):
    ...
```

This material is based upon work supported by the GSA under Contract No. 47QFLA21D0017. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the GSA.

© Galois Inc. 2024, 2025