



Languages, Frameworks, and Tools for Developing Flight Software

Robert L. Bocchino Jr.

Jet Propulsion Laboratory
California Institute of Technology

© 2018 California Institute of Technology
Government sponsorship acknowledged



Introduction

- I work in the Small Scale Flight Software (FSW) Group
- FSW development is challenging
 - FSW is complex and concurrent
 - It must meet rigorous standards of correctness and performance
- Small-scale FSW has particular challenges
 - Compressed budgets and schedules
 - Under-specified and rapidly changing requirements
- **This talk:** Tools that enable FSW development



Introduction

- I work in the Small Scale Flight Software (FSW) Group
- FSW development is challenging
 - FSW is complex and concurrent
 - It must meet rigorous standards of correctness and performance
- Small-scale FSW has particular challenges
 - Compressed budgets and schedules
 - Under-specified and rapidly changing requirements
- **This talk:** Tools that enable FSW development

Developing high-quality software



Introduction

- I work in the Small Scale Flight Software (FSW) Group
- FSW development is challenging
 - FSW is complex and concurrent
 - It must meet rigorous standards of correctness and performance
- Small-scale FSW has particular challenges
 - Compressed budgets and schedules
 - Under-specified and rapidly changing requirements
- **This talk:** Tools that enable FSW development

Developing high-quality software

Meeting tight constraints on budget and schedule



Topics

- **F Prime:** A framework for developing small-scale FSW
 - Provides a modular architecture based on components and ports
 - Provides a complete development ecosystem
- **FPP (F Prime Prime):** A modeling language for F Prime
- **STest:** A framework for **scenario-based testing**
 - Raises the level of abstraction in specifying tests
 - User describes desired behavior with **rules** and **scenarios**
 - Framework automatically generates tests
- **TNet:** A language for specifying **task networks**
 - Provides a flexible way to command spacecraft
 - Supports FSW development by generating code



Outline

- **F Prime FSW framework**
- FPP modeling language
- STest testing framework
- TNet language for autonomy
- Future Work
- Conclusion



The F Prime FSW Framework

- Free and open-source; developed at JPL
- Comprises several elements
 1. A modular architecture based on components and ports
 2. A C++ framework providing core capabilities
 3. Tools for specifying models and generating code
 4. A collection of ready-to use components
 5. Tools for unit and integration testing
- Runs on a variety of platforms

<https://github.com/nasa/fprime>



F Prime: Architecture

- Key concepts
 - **Component:** A unit of FSW function (like a C++ class)
 - **Port:** A point of connection between component instances
 - **Topology:** A directed graph of instances and connections
- Component instances
 - Communicate only through ports
 - Have no compile-time dependencies on other components
- Port connections
 - Are typed and statically specified
 - May be synchronous or asynchronous



F Prime: Architecture

- Key concepts
 - **Component:** A unit of FSW function (like a C++ class)
 - **Port:** A point of connection between component instances
 - **Topology:** A directed graph of instances and connections
- Component instances
 - Communicate only through ports
 - Have no compile-time dependencies on other components
- Port connections
 - Are typed and statically specified
 - May be synchronous or asynchronous

Provides structure to FSW applications



F Prime: Architecture

- Key concepts
 - **Component:** A unit of FSW function (like a C++ class)
 - **Port:** A point of connection between component instances
 - **Topology:** A directed graph of instances and connections
- Component instances
 - Communicate only through ports
 - Have no compile-time dependencies on other components
- Port connections
 - Are typed and statically specified
 - May be synchronous or asynchronous

Provides structure to FSW applications
Enables automatic checking of correctness properties



F Prime: Architecture

- Key concepts
 - **Component:** A unit of FSW function (like a C++ class)
 - **Port:** A point of connection between component instances
 - **Topology:** A directed graph of instances and connections
- Component instances
 - Communicate only through ports
 - Have no compile-time dependencies on other components
- Port connections
 - Are typed and statically specified
 - May be synchronous or asynchronous

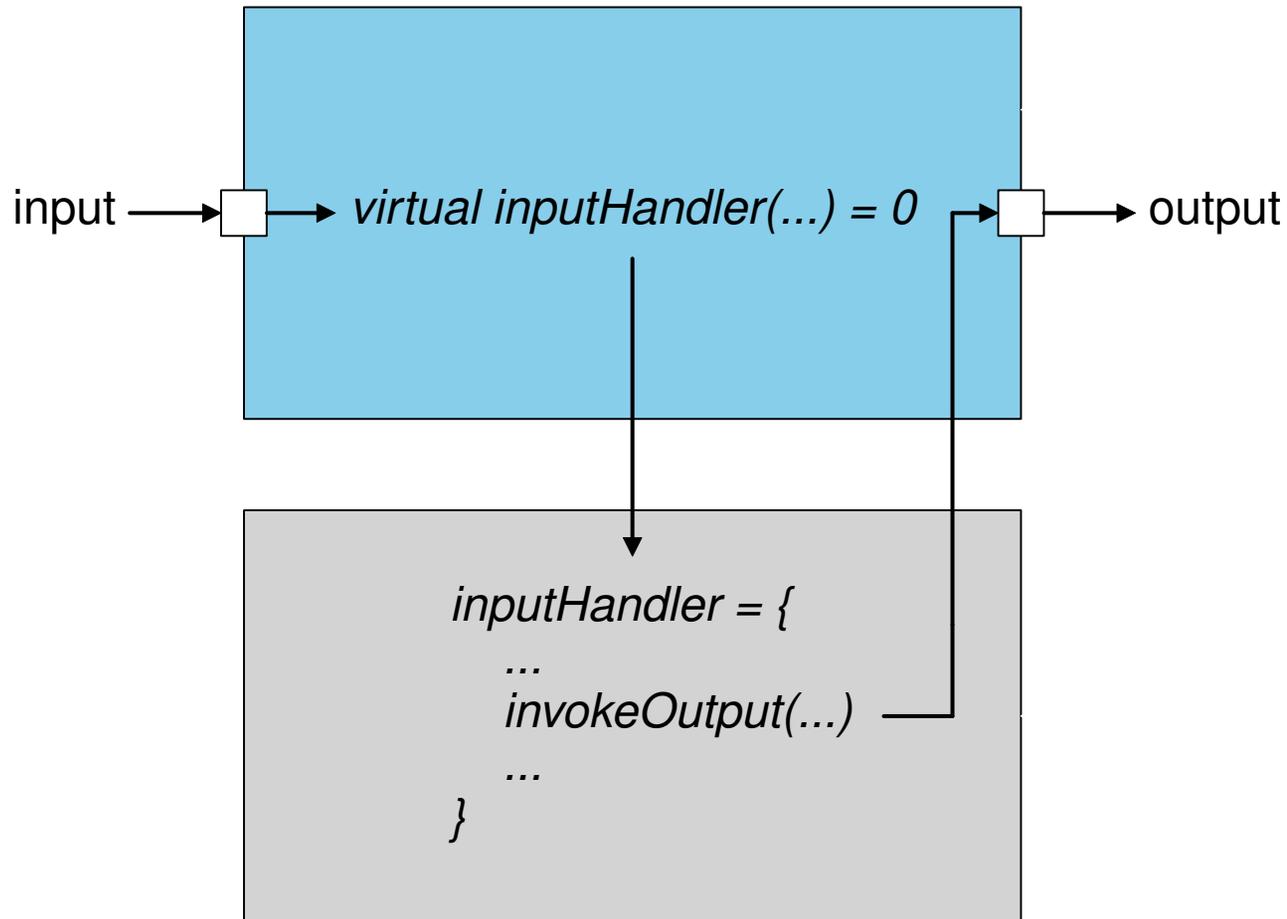
Provides structure to FSW applications

Enables automatic checking of correctness properties

Enhances reusability of FSW components

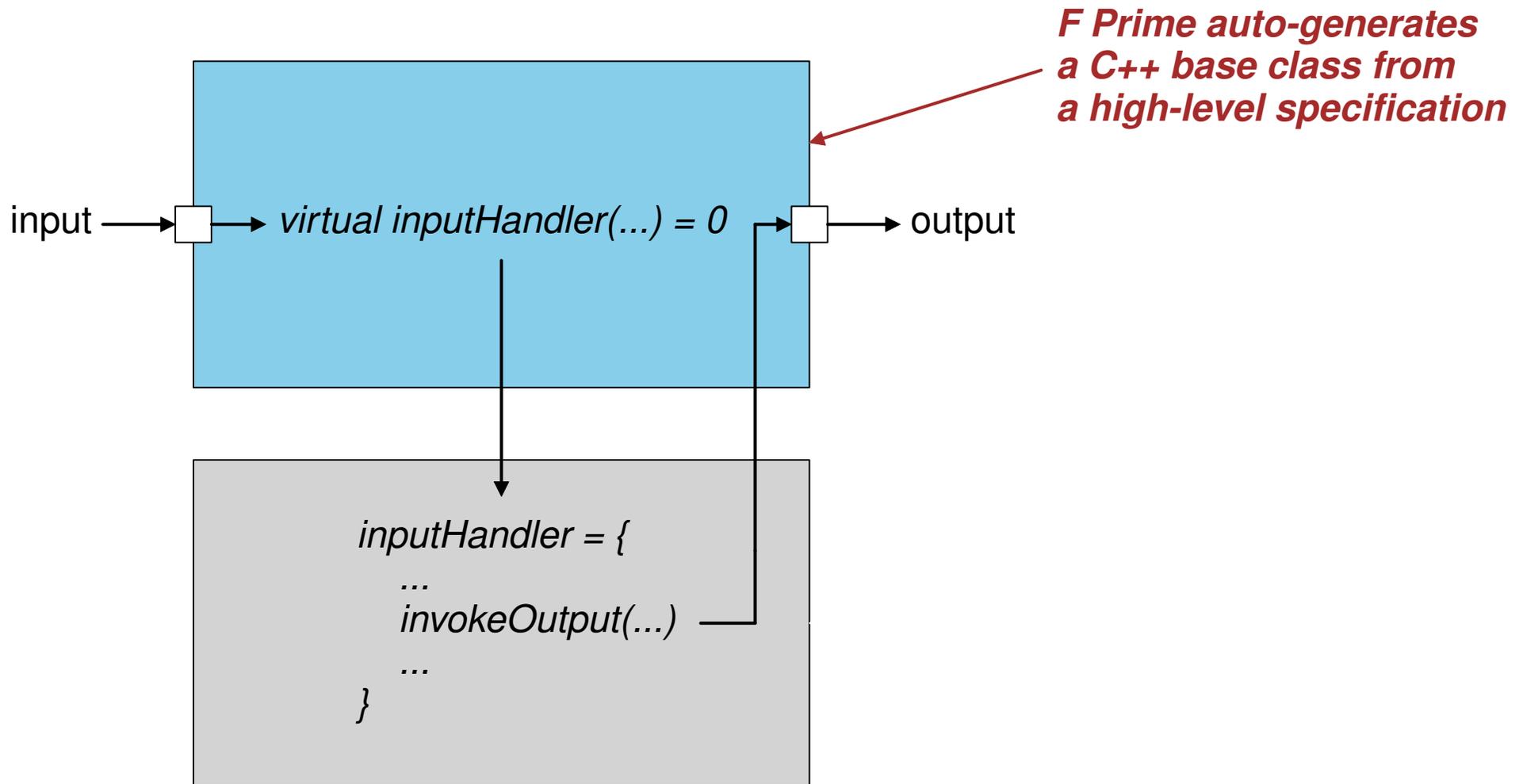


F Prime: C++ Framework



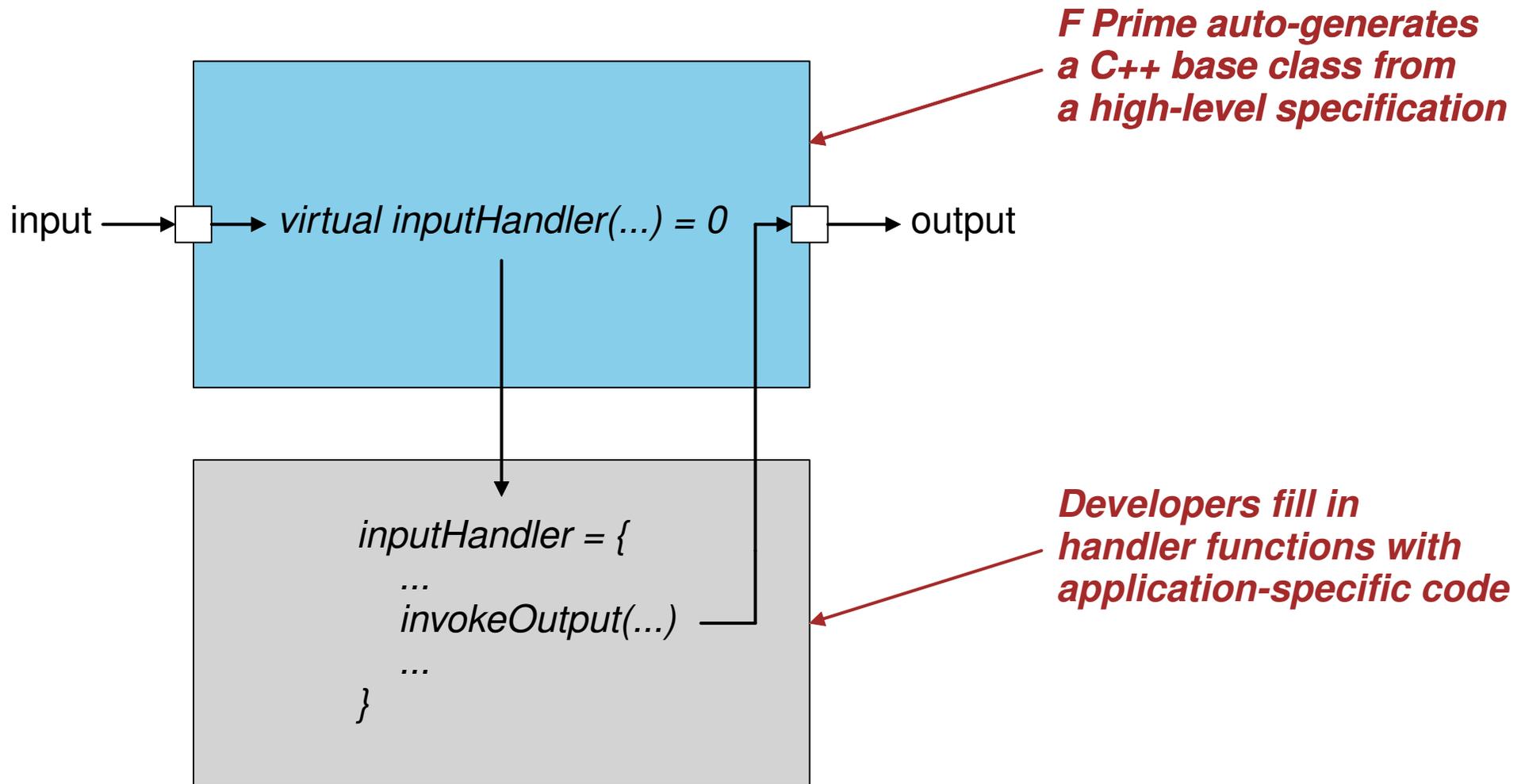


F Prime: C++ Framework





F Prime: C++ Framework





F Prime: Modeling



- F Prime developers write high-level models
 - Define components and ports
 - Specify connections in a topology
 - Define flight-ground interface (commands, events)
- F Prime tools generate
 - Component base classes
 - Code for connecting the ports
 - Command and event dictionaries



F Prime: Testing

- Testing is both labor-intensive and critical
- F Prime provides robust support for testing
- Unit tests of components
 - F Prime automatically generates a tester base class
 - It is the mirror image of the component base class
 - Tests go in a class derived from the tester base
- Integration tests of deployments (executable builds)
 - F Prime provides a complete ground data system
 - It includes a GUI for interactive tests
 - It also includes a python API for scripted tests



Experience with F Prime

- We have used F Prime on several space missions
 - **ISS RapidScat** scatterometer (flew)
 - **ASTERIA** CubeSat space telescope (flying now)
 - **Mars Helicopter** (in development)
 - **Lunar Flashlight** CubeSat (in development)
 - **Near Earth Asteroid (NEA) Scout** CubeSat (in development)
- We have used F Prime for research and education
 - JPL R&D project on autonomous FSW
 - Collaborations with CMU and other universities
- F Prime reduces the cost of developing FSW
 - Enables sharing and reuse among projects
 - Lets developers focus on mission-specific code



ASTERIA





Mars Helicopter





Outline

- F Prime FSW framework
- **FPP modeling language**
- STest testing framework
- TNet language for autonomy
- Future Work
- Conclusion



FPP (F Prime Prime)

- A modeling language and visualization tool for F Prime
- Intended to replace our current modeling approach
 - Plugin for a commercial tool called MagicDraw
 - Handwritten XML
- Goals
 - Free and open source
 - Simple and easy to use
 - Well integrated with the rest of F Prime
- Developed in collaboration with CMU MSE program
- Uses Acme Studio for architecture checking



FPP: Modeling Ports

```
namespace Fw

porttype Cmd {

    comment = "Command port"

    arg opcode : FwOpcodeType {
        comment = "Command opcode"
    }

    arg seqNum : U32 {
        comment = "Command sequence number"
    }

    arg args : CmdArgBuffer {
        pass_by = reference
        comment = "Buffer containing arguments"
    }

}
```



FPP: Modeling Components

```
namespace Svc

component CmdDispatcher {

    kind = active

    comment = "A component for dispatching commands"

    port cmdOut : Fw.Cmd {
        direction = out
        number = NumCmdPorts
        comment = "Dispatches commands"
    }

    ...

}
```



FPP: Modeling Topologies

```
namespace Ref

instance cmdDisp : Svc.CmdDispatcher {
  base_id = 0x100
  base_id_window = 0x100
}

instance cmdSeq : Svc.CmdSequencer {
  base_id = 0x200
  base_id_window = 0x100
}

...

topology CommandResponse {

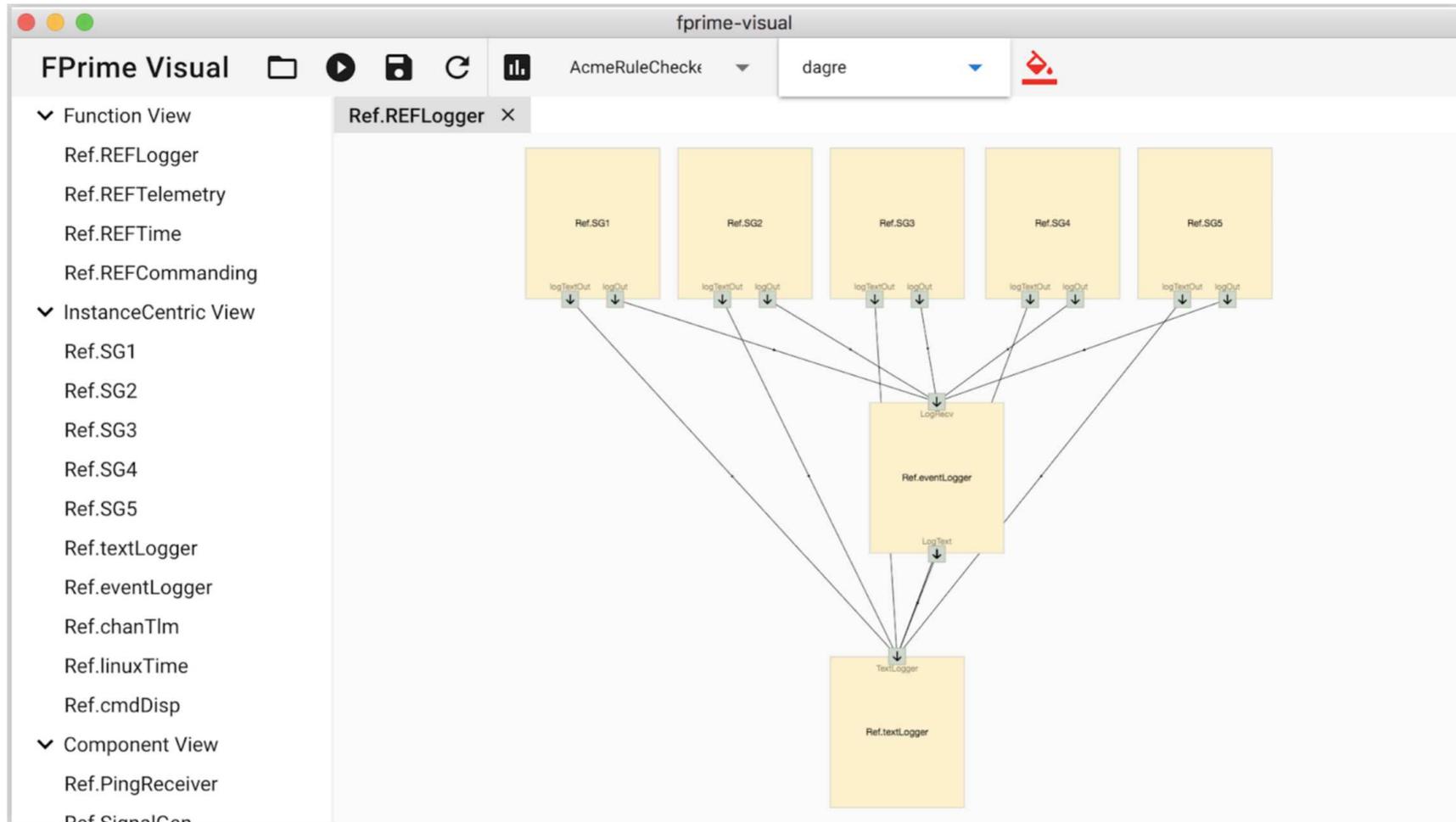
  cmdSeq.cmdResponseOut -> cmdDisp.cmdResponseIn

  ...

}
```



FPP Visualizer





Outline

- F Prime FSW framework
- FPP modeling language
- **STest testing framework**
- TNet language for autonomy
- Future Work
- Conclusion



The STest Testing Framework

- A C++ framework for writing tests
- Approach
 1. The developer uses **rules** to describe desired system behavior
 2. The developer uses rules to write **scenarios**
 3. The framework uses scenarios to generate tests
- Advantages
 - Factors tests into small reusable pieces
 - Separates system behavior from test construction
 - Enables generation of many tests (potentially millions)



STest: Rules

- A rule R has two parts:
 1. The **precondition**: When to apply R
 2. The **action**: What to do and what to check when applying R
- Example: Allocating a buffer from a memory manager
 - **Precondition**: A buffer is available and s is a legal buffer size
 - **Action**:
 - Request a buffer of size s
 - Check that the action succeeded
 - Check that the action produced a buffer of size s
- We can write similar rules for
 - Successful deallocation
 - Error cases



STest: Scenarios

- A scenario is a recipe for using rules to construct tests
- Example scenarios
 - S1. A fixed sequence of rules R_1, \dots, R_n
 - S2. All valid random sequences of a fixed set of rules $\{R_i\}$
 - S3. All stepwise interleavings of a fixed set of scenarios $\{S_i\}$
- STest provides several operations for constructing scenarios
 - Nondeterministic choice
 - Repetition
 - Conditional execution
 - Interleaving
- One scenario can automatically generate millions of tests



STest: Tool Integration

- *kontest*
 - An experimental tool developed at JPL
 - Uses **concolic testing** to generate test inputs
 - Run with input I_0 and collect constraints
 - Solve constraints to generate inputs I_1, \dots, I_n
 - Repeat for I_1, \dots, I_n
 - Extends *klee*; operates on LLVM bitcode
 - Integrated with STest for picking test inputs
- Spin
 - A widely used explicit-state model checker
 - Remembers and systematically explores system states
 - Integrated with STest for picking rules to apply



Experience with STest

- We have applied STest and *kontest* to
 - Several F Prime components
 - A software simulation for an attitude control system
 - The file system for the Curiosity Mars rover
- We found tricky corner-case bugs in
 - The Curiosity file system
 - The ASTERIA Communication component
- Challenging to find with traditional testing



Outline

- F Prime FSW framework
- FPP modeling language
- STest testing framework
- **TNet language for autonomy**
- Future Work
- Conclusion



Task Networks

- A **task network** is a way of commanding a spacecraft
- It divides spacecraft activity into **tasks** consisting of
 - Conditions on system state
 - Commands to perform
 - **Impacts** to **timelines** (updates to modeled state variables)
- Example: Taking an image
 - **Precondition:** Imager is on
 - **Command:** Take the image
 - **Impact:** Add one to count of images taken
- Supports on-board autonomous behavior



The TNet Language

- A statically typed domain-specific language (DSL)
 - Uses structural typing
 - Supports aggregate values: structures, arrays, ranges, sets
- Goals
 1. Provide a convenient way to specify task networks
 2. Generate task networks to submit to an on-board planner
 3. Generate C++ code for inclusion in FSW
- C++ code generation includes
 - Representations of state data structures
 - Code for instantiating **templates** (tasks with unbound parameters)



TNet: Status

| <u>Language Feature</u> | <u>Design</u> | <u>Implementation</u> |
|------------------------------|---------------|-----------------------|
| Types and constants | ✓ | ✓ |
| States and timelines | ✓ | ✓ |
| Task definitions | | |
| Task templates | | |
| Instantiating task templates | | |

Development started in May 2018



TNet: Types and Constants

```
module GNC {  
  
  module Position {  
  
    @ The type of a position  
    type t = { x : F64, y : F64, z : F64 }  
  
    @ Componentwise minimum position value  
    constant min = { x = -1, y = -1, z = -1 } : t  
  
    @ Componentwise maximum position value  
    constant max = { x = 1, y = 1, z = 1 } : t  
  
  }  
  
}
```



TNet: Enumerations

```
module GNC {  
  
    @ Spacecraft body vector  
    enum BodyVector {  
  
        @ The solar panel  
        SOLAR_PANEL = 1  
  
        @ The camera boresight  
        CAMERA_BORESIGHT = 2  
  
        ...  
  
    }  
  
}
```



TNet: State

```
module GNC {  
  
  module State {  
  
    @ The type of a GNC state value  
    type t = {  
      @ Position  
      position : Position.t  
      @ Body vector  
      bodyVector : BodyVector  
      ...  
    }  
  
    @ The GNC state value  
    state s : t = {  
      position = { x = 0, y = 0, z = 0 }  
      bodyVector = BodyVector.SOLAR_PANEL  
      ...  
    }  
  
  }  
  
}
```



TNet: Timelines

```
module GNC {  
  
    module State {  
  
        @ The position timeline  
        timeline s.position in Position.min..Position.max  
  
        @ The body vector timeline  
        timeline s.bodyVector  
  
    }  
  
}
```



Outline

- F Prime FSW framework
- FPP modeling language
- STest testing framework
- TNet language for autonomy
- **Future Work**
- Conclusion



Future Work

- FPP
 - Current tools are alpha versions
 - Make the tools more robust and feature complete
 - Add more analysis capabilities
- STest
 - Use Promela (Spin modeling language) to write scenarios
 - Develop a general language for describing scenarios
- TNet
 - Continue to develop the language
 - Integrate with extended ASTERIA mission
 - Integrate with Autonomy research



Outline

- F Prime FSW framework
- FPP modeling language
- STest testing framework
- TNet language for autonomy
- Future Work
- **Conclusion**



Conclusion

- FSW development is challenging
- We are developing several tools that can help
 - F Prime FSW framework
 - FPP modeling language
 - STest testing framework
 - TNet language for autonomy
- We have several directions for future work