

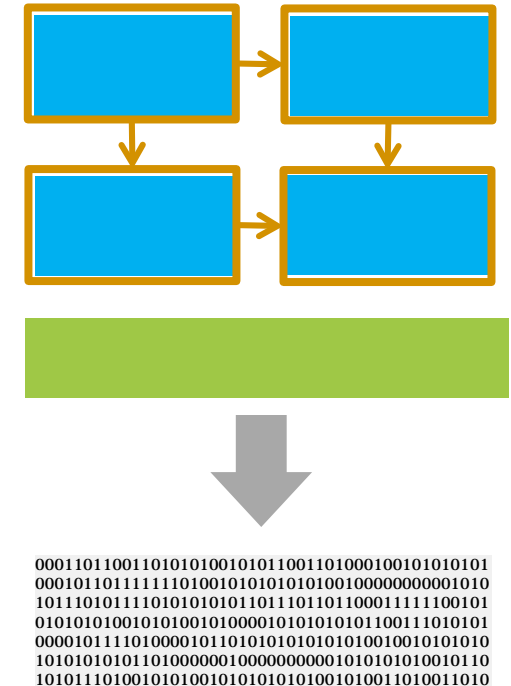


# ARCHITECTURE-DRIVEN ASSURANCE

JENNIFER DAVIS, PH.D.  
TRUSTED SYSTEMS  
OCTOBER 2019

# APPROACH: ARCHITECTURE-DRIVEN ASSURANCE

1. Architecture model is correct
  - Properties, structure, behavior, interaction of components, interfaces, contracts
  - Analyzable
2. Components are correct
  - Consistent/realizable contracts
  - Components verified to implement contracts
3. System does what the model says
  - No other information flows (memory safety, isolation)
  - OS executes model correctly (incl. timing)
4. System implementation corresponds to model
  - Automatic build from component and architecture models



# SELECTED PROGRAMS LEVERAGING ARCHITECTURE-DRIVEN ASSURANCE

- DARPA High Assurance Cyber Military Systems (HACMS)
- DARPA Cyber Assured Systems Engineering (CASE)

# HIGH ASSURANCE CYBER MILITARY SYSTEMS (HACMS) TECHNOLOGIES



## Open source tools, languages, software

1. Architecture modeling and analysis tools (AADL)
  - Assume-Guarantee Reasoning Environment (AGREE)
  - Architecture-based assurance cases (Resolute)
2. Ivory/Tower embedded Domain Specific Languages
  - Memory safe component software
  - Code generation from high-level specification
3. seL4 formally verified OS kernel
  - Isabelle/HOL proof of correctness
  - Security properties proven to binary level
4. Automated build from models
  - Support for seL4, eChronos, VxWorks, Linux



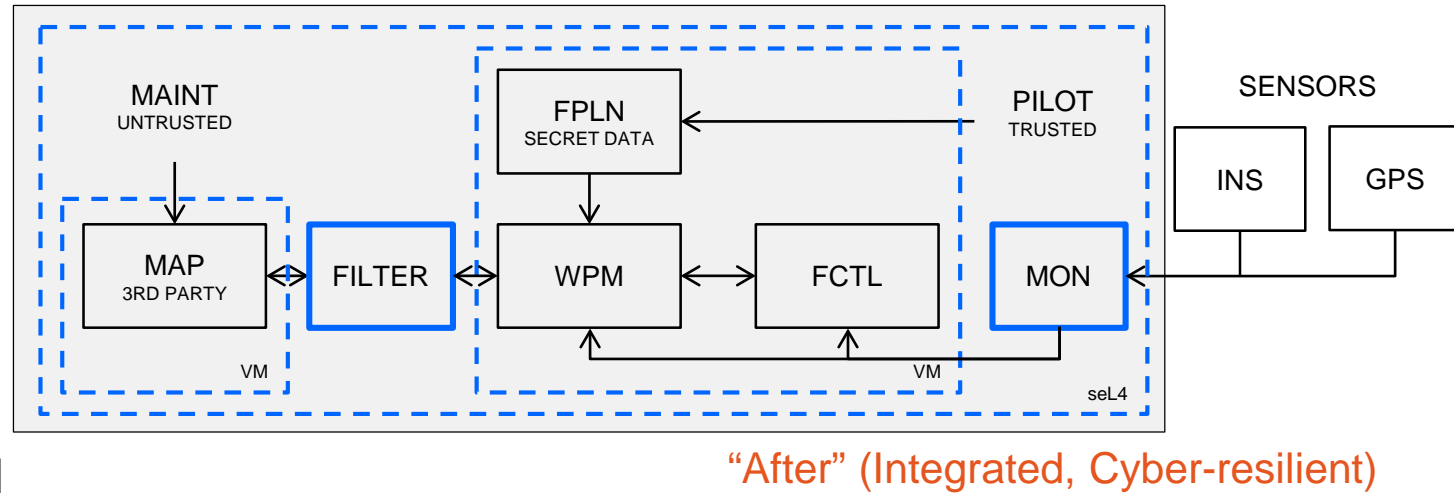
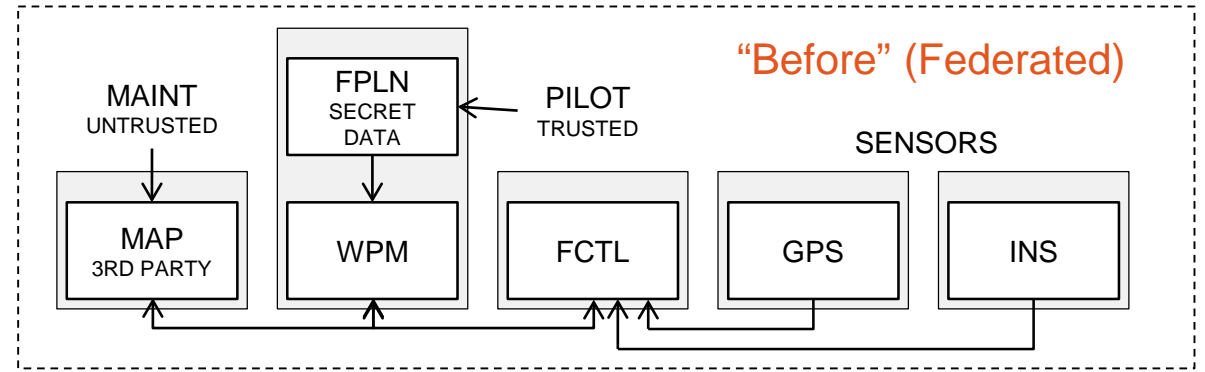
# CYBER ASSURED SYSTEMS ENGINEERING (CASE)

- The goal of CASE is to develop the necessary design, analysis and verification tools to allow system engineers to design-in cyber resiliency and manage tradeoffs as they do the other non-functional properties when designing complex embedded computing systems
  - Cyber resiliency means that the system is tolerant to cyberattacks in the same way that safety critical systems are tolerant to random faults – they recover and continue to execute their mission function
  - Cyber security requirements are addressed today by penetration testing late in the development, resulting in expensive rework
  - Cyber requirements are often “shall not” statements about the system, and so are not testable (formal methods required)

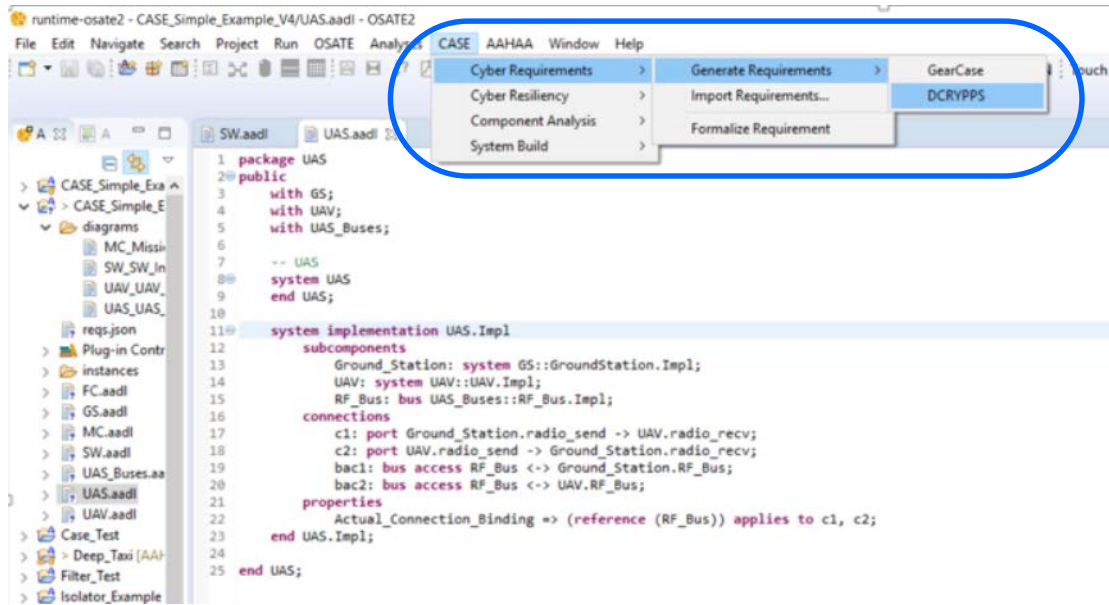
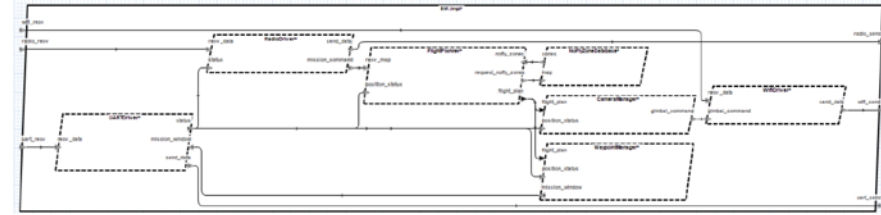


# APPROACH

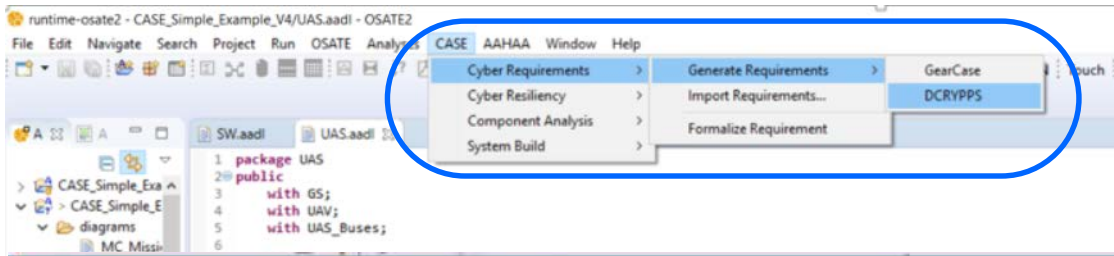
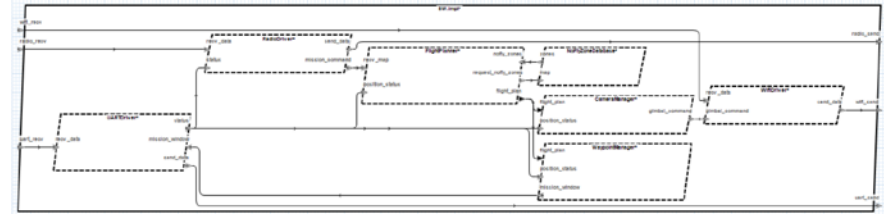
- Start with initial design, new or legacy
  - Federated avionics system
- Generate new cyber requirements
  - Possibly based on modified system architecture
- Tool-assisted transformations of system architecture
  - Satisfy cyber requirements
  - Manage other design trade-offs
  - Insertion/synthesis of high-assurance components may be needed
- Verification of cyber resiliency
- Generate system from architecture model



# CYBER REQUIREMENTS



# CYBER REQUIREMENTS



The screenshot shows the 'Import Cyber Security Requirements' dialog box. It is divided into two main sections: 'Requirements Browser' and 'View/Edit Requirement'.

**Requirements Browser:**

Status	Type	ID	Short Description
Formalize	well_formed	Req003_FlightPlanner	The FlightPlanner shall only accept well-formed messages from
ToDo	monitored		The output of Third-party software shall be monitored for con
ToDo	trusted_source		The FlightPlanner shall only accept messages from a trusted G
ToDo	isolated		Third-party software shall be isolated from critical component

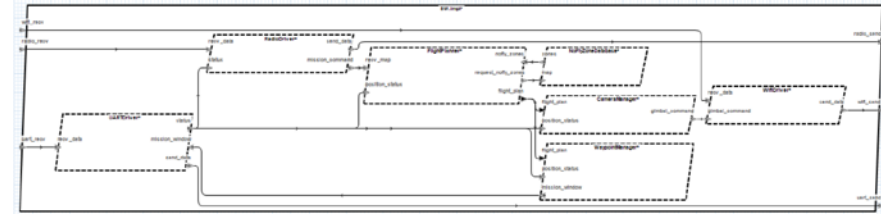
**View/Edit Requirement:**

Status: Formalize (dropdown)  
Generation Tool: DCRYPPS  
Type: well\_formed  
ID: Req003\_FlightPlanner  
Description: The FlightPlanner shall only accept well-formed messages from the GroundStation  
Component: SW::SW.Impl.FPLN  
Reason for omission: N/A

Buttons: OK, Cancel



# CYBER REQUIREMENTS

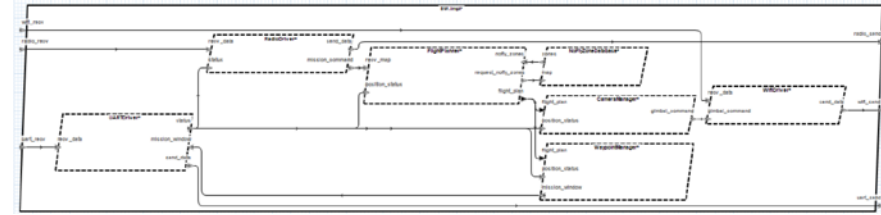


```
runtime-osate2 - CASE_Simple_Example_V4/UAS.aadl - OSATE2
File Edit Navigate Search Project Run OSATE Analysis CASE AAHAA Window Help
Cyber Requirements > Generate Requirements > GearCase
Cyber Resiliency > Import Requirements... DCRVPPS
Component Analysis > Formalize Requirement
System Build >

120 thread FlightPlanner
121 -- The FlightPlanner is an abstraction for UxAS.
122 -- It accepts a command message containing a map and flight pattern, and generates a mission.
123 -- The FlightPlanner also has access to a No-Fly zone database, which it uses to generate the mission
124 -- to avoid specified no-fly zones
125 features
126 flight_plan: out data port Mission;
127 rcv_map: in event data port RF_Msg.Impl;
128 request_nofly_zones: out event data port Map;
129 nofly_zones: in event data port MapArray;
130 position_status: in event data port Coordinate.Impl;
131 annex agree {**
132 assume Req001_FlightPlanner "The Flight Planner shall receive a valid message from the Ground Station" : VALID_MESSAGE(rcv_map);
133 guarantee Req002_FlightPlanner "The Flight Planner shall generate a valid mission" : good_mission(flight_plan);
134 assume Req003_FlightPlanner "The FlightPlanner shall only accept well-formed messages from the GroundStation" : WELL_FORMED_MESSAGE(rcv_map);
135 **};
136 end FlightPlanner;
137
138 thread implementation FlightPlanner.Impl
139 end FlightPlanner.Impl;
```

```
260
261 with CASE_Model_Transformations;
262
263 annex resolve {**
264 Req003_FlightPlanner(context : component, property_id : string) <=
265 ** "[well_formed] The FlightPlanner shall only accept well-formed messages from the GroundStation" **
266 agree_prop_checked(context, property_id)
267
268 **};
```

# CYBER REQUIREMENTS



```
runtime-osate2 - CASE_Simple_Example_V4/UAS.aadl - OSATE2
File Edit Navigate Search Project Run OSATE Analysis CASE AAHAA Window Help
Cyber Requirements > Generate Requirements > GearCase
Cyber Resiliency > Import Requirements... DCRVPPS
Component Analysis > Formalize Requirement
System Build

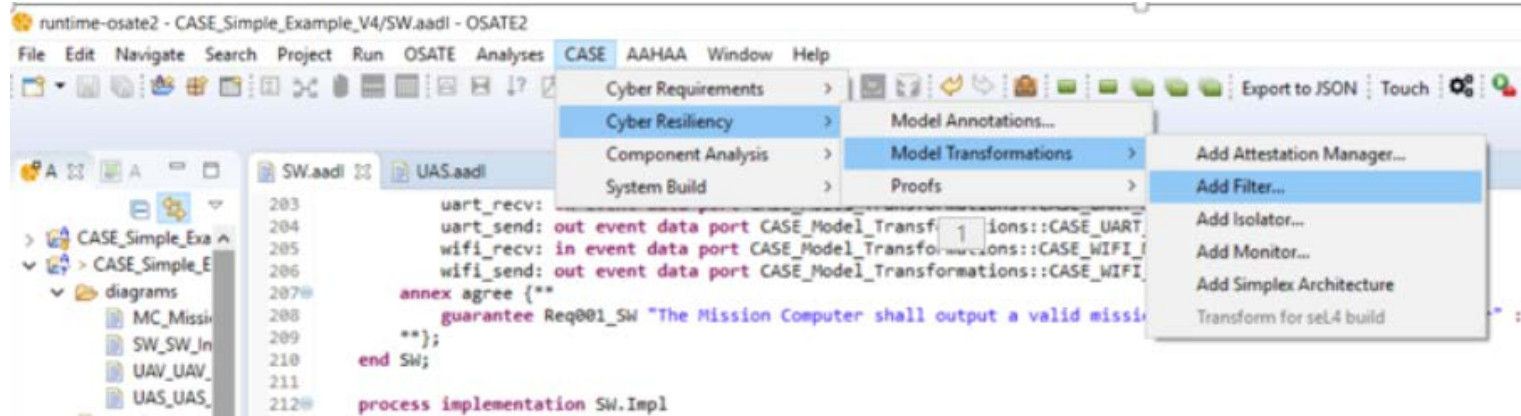
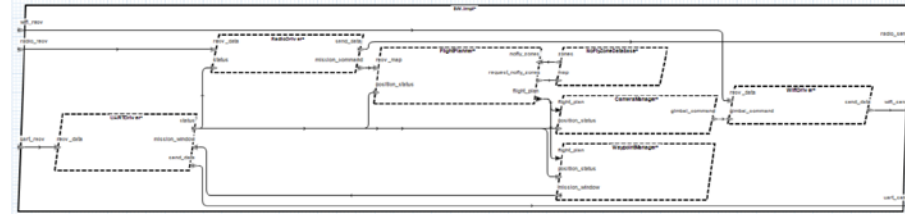
120 thread FlightPlanner
121 -- The FlightPlanner is an abstraction for UxAS.
122 -- It accepts a command message containing a map and flight pattern, and generates a mission.
123 -- The FlightPlanner also has access to a No-Fly zone database, which it uses to generate the mission
124 -- to avoid specified no-fly zones
125 features
126 flight_plan: out data port Mission;
127 recv_map: in event data port RF_Msg.Impl;
128 request_nofly_zones: out event data port
129 nofly_zones: in event data port MapArray;
130 position_status: in event data port Coord
131 annex agree {**
132 assume Req001_FlightPlanner "The Flight P
133 guarantee Req002_FlightPlanner "The Fligh
134 assume Req003_FlightPlanner "The FlightPl
135 **};
136 end FlightPlanner;
137
138 thread implementation FlightPlanner.Impl
139 end FlightPlanner.Impl;
```

Problems Properties Assurance Case AGREE Results Error Log Console

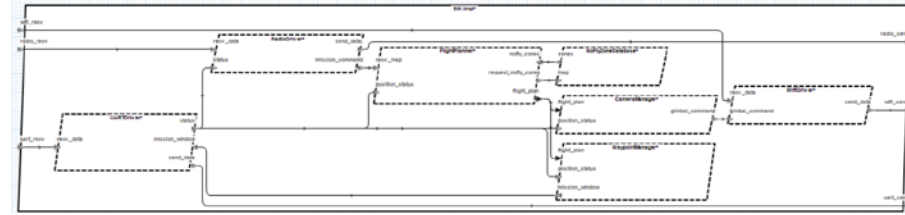
- Req003\_FlightPlanner(SW\_Impl\_Instance : SW::SW.Impl, "Req003\_FlightPlanner")
  - [well\_formed] The FlightPlanner shall only accept well-formed messages from the GroundStation
    - AGREE properties passed
      - AGREE analysis was run on the current version of the model

```
260
261 with CASE_Model_Transforma
262
263 annex resolve {**
264 Req003_FlightPlanner(context : component, property_id : string) <=
265 ** "[well_formed] The FlightPlanner shall only accept well-formed messages from the GroundStation" **
266 agree_prop_checked(context, property_id)
267
268 **};
```

# TRANSFORMATION: FILTER ADDED TO AADL MODEL



# TRANSFORMATION: FILTER ADDED TO AADL MODEL

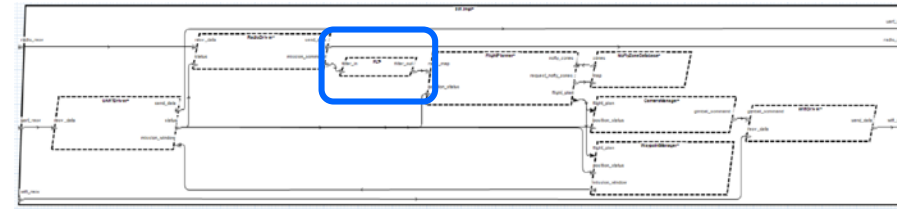


The screenshot shows the OSATE2 software interface. The main window displays the 'CASE Simple Example V4/SW.aadl - OSATE2' project. The 'CASE' menu is open, and the 'Add Filter...' option is selected. The 'Add Filter' dialog box is displayed, allowing the user to enter filter details. The dialog box contains the following fields and options:

- Filter implementation name: FLT
- Filter implementation language: CakeML
- Create log port:  None  Event  Data  Event Data
- Requirement: Req003\_FlightPlanner
- Preserve Guarantees from RadioDriver:  Req001\_RadioDriver "Only valid command messages shall be forwarded to message destination components"
- Filter AGREE contract: guarantee Req001\_Filter "The filter shall allow only well-formed messages to pass" : WELL\_FORMED\_MESSAGE(filter)

The dialog box has 'OK' and 'Cancel' buttons at the bottom right.

# TRANSFORMATION: FILTER ADDED TO AADL MODEL



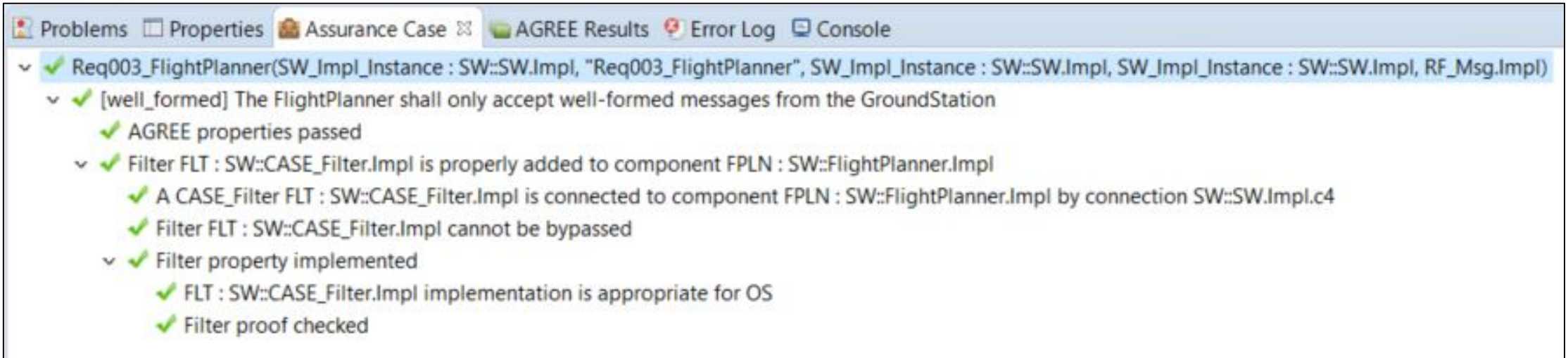
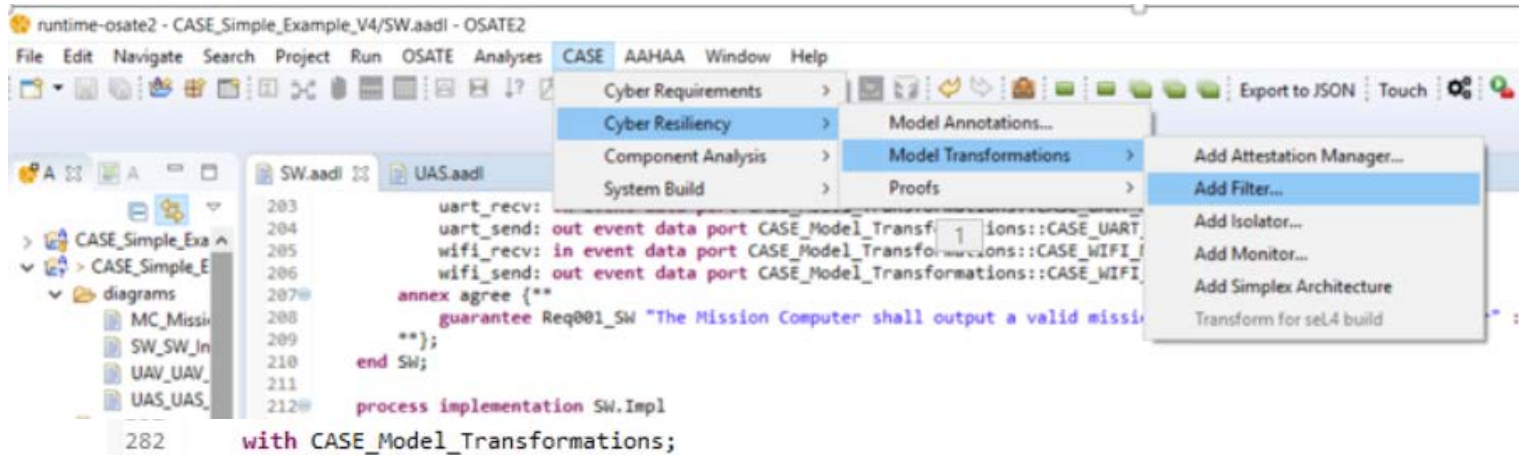
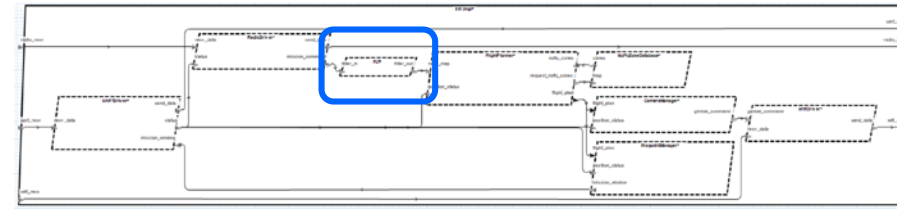
```

runtime-osate2 - CASE_Simple_Example_V4/SW.aadl - OSATE2
File Edit Navigate Search Project Run OSATE Analyses CASE AAHAA Window Help
Cyber Requirements >
Cyber Resiliency > Model Annotations...
Component Analysis > Model Transformations > Add Attestation Manager...
System Build > Proofs > Add Filter...
Add Isolator...
Add Monitor...
Add Simplex Architecture
Transform for sel4 build

203    uart_recv:
204    uart_send: out event data port CASE_Model_Transformations::CASE_UART
205    wifi_recv: in event data port CASE_Model_Transformations::CASE_WIFI
206    wifi_send: out event data port CASE_Model_Transformations::CASE_WIFI
207    annex agree {**
208    guarantee Req001_SW "The Mission Computer shall output a valid missi
209    **};
210    end SW;
211
212    process implementation SW.Impl
282    with CASE_Model_Transformations;
283
284    annex resolute {**
285    Req003_FlightPlanner(context : component, property_id : string, filter : component, connection_name : string, message_type : data) <=
286    ** "[well_formed] The FlightPlanner shall only accept well-formed messages from the GroundStation" **
287    agree_prop_checked(context, property_id) and add_filter(context, filter, connection_name, message_type)
288
289    **};
290
291    -----
292    -- MODEL TRANSFORMATIONS --
293    -----
294
295    -- Top-level claim for proper insertion of a filter
296    add_filter(context : component, filter : component, conn_name : string, msg_type : data) <=
297    ** "Filter " filter " is properly added to component " context **
298    filter_exists(filter, context, conn_name) and filter_not_bypassed(filter, context, msg_type) and filter_prop_checked()
299
300

```

# TRANSFORMATION: FILTER ADDED TO AADL MODEL



# CYBER RESILIENT ARCHITECTURE PATTERNS

- Library of general, tool-assisted *architecture model transformations* that mitigate vulnerabilities or address cyber requirements
- Automatic insertion and verification of transform properties as assume-guarantee contracts and assurance case claims
- Examples
  - Filter
  - Attestation
  - Isolation
  - Monitor/Simplex
  - Distributed Action (e.g., Zeroize)
  - seL4 implementation

# CASE TARGETS



- Experimental platform: AFRL UxAS

- Demonstration platform: CH-47 CAAS



# FOR MORE INFORMATION...

- Send me an email at [jen.davis@collins.com](mailto:jen.davis@collins.com)
- Releases of CASE tool suite (includes Resolute and AGREE)  
<https://github.com/loonwerks/formal-methods-workbench/releases>
- HAMR (for System Build): <https://github.com/sireum/hamr-plugin-update-site>
- More information on CASE project at <http://loonwerks.com/projects/case.html>
- Tool descriptions and papers
  - Resolute: <http://loonwerks.com/tools/resolute.html>
  - AGREE: <http://loonwerks.com/tools/agree.html>
- The seL4 Microkernel: <https://sel4.systems/>

The tools are open source  
and free to use!