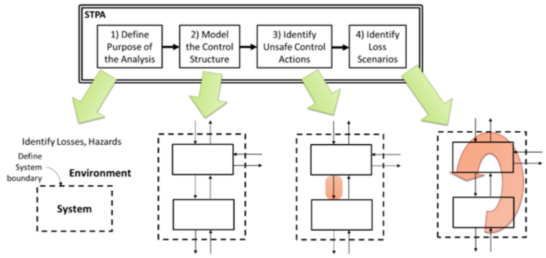
RTS-TK A toolkit to create a Run Time Sentry (RTS) from the STPA-SEC tailored process

Leveraging the System Theoretic Process Analysis from the Kharsansky Satelite Master's Thesis to generate Reference Monitors



Kharsansky, Alan. A systemic approach toward scalable, reliable and safe satellite constellations. Diss. Massachusetts Institute of Technology, 2020. SOURCE

The Kharansky Master's Thesis describes how one might use the STPA process to **re-architect** a system, specifically a satellite constellation, given a proper STPA analysis. However, often re-architecting a system will require modifications to existing components or existing software. This report describes how one might use an STPA notation tool combined with several open source Formal Methods based tools to create code that prevents or protects against specific failure conditions identified in your STPA analysis

Installation

The following git repo can be provided upon request:

- STPA_DOCKER: This is the custom tool developed by MBClark LLC and SCASD Consulting. The tool is a combination of docker containers that serve the jupyter lab application and supporting tools. The tools are provided as government purpose licensed tools.
- STPA_REF: As of this writing OGMA, FRET, and CoPilot have not been directly integrated in STPA_DOCKER. In the next phase of the project, STPA_DOCKER will be incorporated with the tools described in this report. However, the current version of STPA_REF is a series of installation scripts to guide the install of OGMA, FRET, CoPilot and the associated model checkers and compilers required to use these tools

Currently, the entire system can run on Ubuntu 23.x including the supporting tools.

Step 1 Define the Purpose of the analysis

Kharsansky Environment - complete satellite constellation

Problem:

- Provide a service over a particular area of interest. For each particular type of Satellite application.

METHOD (By Means Of):

- Communication: Provide broadband coverage over a specific area of interest
- Asset monitoring and controlling: send and receive data and commands to and from assets over a specific area of interest to a ground network or facility.
- Remote sensing: Obtain scientific or commercial data of the surface or atmosphere of the earth over a specific area of interest
- GNSS: Provide navigation information (position and velocity) to ground, airborne, or space terminals over a specific area of interest.

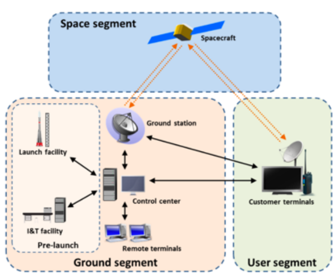


Figure 13 – Sample system overview of a satellite system. Credit: Swpb / CC BY-SA

Step 1.1 Identify the Core Components of the System

This can be derived from your SysML architecture block diagrams. Order the components in order of hierarchy. Physical Blocks with physical stereotypes such as mass, thrust, satellite position or attitude go at the bottom of the hierarchy. These will be referred to as controlled processes or CP.

Logical controllers or sensors that automatically control physical blocks are next in the hierarchy and denoted by AC or Automatic Controller.

Logical controllers that are of a Human or Process stereotype go on the top and are denoted by HC or Human Controller.

It is important to organize your system components in this way so that you can get a better feel for which component has **Authority** over the next

[1]:
```python
## Run this cell first to import the spec-books library
import os
import sys
code_path = os.path.join(os.path.dirname(os.path.dirname(os.getcwd())),'CODE')
if code_path not in sys.path:
    sys.path.append(code_path)
import spec_books as sb
```

[2]:
```python
## Define the excel workbook that contains the spec-books data
WB = 'KSAT.xlsx'
```

Example from paper of components defined in the first architecture

stpa_table creates a javascript table with the structure necessary for both identifying components and then visualizing them as well. HC,AC, and CP designators color code the drawing

[3]:
```
com = sb.stpa_table('Components',WB)
```

Add Row | Delete Row | Save Table

Filter...

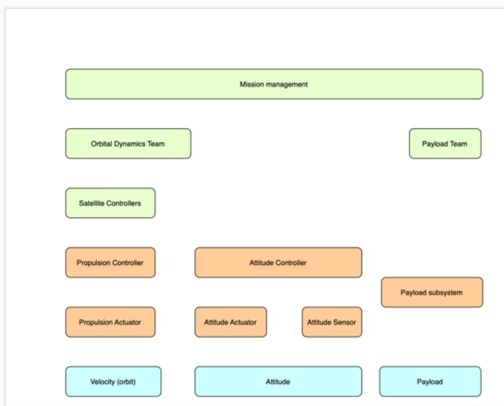
ID	Parent	Name	Order	Type	PositionX	PositionY	Width	Height	Container
E1	1	Mission management	1	HC	100	100	700	50	0
E2	1	Orbital Dynamics Team	2	HC	100	200	210	50	0
E3	1	Payload Team	2	HC	677	200	120	50	0
E4	1	Satellite Controllers	3	HC	100	300	150	50	0
E5	1	Propulsion Controller	4	AC	100	400	150	50	0
E6	1	Attitude Controller	4	AC	317	400	280	50	0
E7	1	Payload subsystem	4	AC	630	450	170	50	0
E8	1	Propulsion Actuator	5	AC	100	500	150	50	0
E9	1	Attitude Actuator	5	AC	317	500	120	50	0
E10	1	Attitude Sensor	5	AC	497	500	100	50	0
E11	1	Velocity (orbit)	6	CP	100	600	160	50	0

Component_Diagram function creates a Drawio visualization of the components in hiearchical order. More information about the location of each component is provided in the Components table in the sample excel data store provided with this tool (KSAT.xlsx)

[4]:
```
components = sb.cross_drawing(WB,'Components','Components',None,'component')
components.show_diagram()
```

Add a Component ->	HC	AC	CP	ST	Save
Add a Transition ->	FB	CA	MSG	SM	

Save & Exit



Step 1.2 - Define Losses

[5]:
```
loss = sb.stpa_table('Losses',WB)
```

Add Row | Delete Row | Save Table

Filter...

ID	Name	Severity
L-1	Inability to provide the service	Catastrophic
L-2	Loss or damage of a member satellite	Catastrophic
L-3	Damage to a non-member satellite	Catastrophic
L-4	Loss of life, injury to people or damage of assets	Catastrophic
L-5	Violation of regulations	Catastrophic

Step 1.3 - Define the Hazards

```
[6]: Haz = sb.stpa_table('Hazards',WB)
```

Add Row | Delete Row | Save Table

Filter...

ID	System	Condition
H-1	Satellite	unable to perform its objectives - incorrectly collects scientific data, relay communications, provide guidance aid
H-2	Satellite	Is in a reentry trajectory towards a populated area
H-3.1	Satellite	Uses a payload over a forbidden target or an allowed area but with forbidden parameters
H-3.2	Satellite	Violates its relative position requirement in the constellation
H-3.3	Satellite	Is in a collision trajectory with another orbiting body
H-4	Satellite	Exceeds safe attitude operating envelope (i.e., sensitive payloads pointing to the sun, pressurized vessels exposed to the sun, radiators exposed to sun, electronics exposed to extreme cold, solar panels not pointing to the sun)

Step 1.4 - Define the Loss / Hazard Relationship

```
[7]: LVH = sb.stpa_table('LvH',WB)
```

Add Row | Delete Row | Save Table

Filter...

Hazards ↓	L-1	L-2	L-3	L-4	L-5
H-1	T				
H-2					T
H-3.1	T	T	T	T	T
H-3.2	T	T	T		
H-3.3	T	T	T		
H-4		T	T		

Step 1.5 - Define the Constraints

```
[8]: Con = sb.stpa_table('Constraints',WB)
```

Add Row | Delete Row | Save Table

Filter...

ID	System	Condition	Hazards
C-1	Satellite members	must always be able (have the appropriate conditions) to perform its objectives	H-1
C-2	Satellite members	Payloads must always be used over allowed targets and with the allowed configuration as defined by international and national regulations.	H-2
C-3	Satellite members	payload is incorrectly used over a forbidden target or with a forbidden configuration, it must be detected and corrected.	H-2
C-4	Satellite members	positions must be maintained according to the relative position requirements. (If AD1=Tight)	H-3.1
C-5	Satellite members	position requirement is violated, it must be detected and corrected. (If AD1=Tight)	H-3.1
C-6	Satellite members	must not be in collision trajectories with other orbiting bodies.	H-3.2
C-7	Satellite members	Must detect and avoid collisions with other satellites even if violates other constraints	H-3.2
C-8	Satellite members	must avoid reentry trajectories towards populated areas.	H-3.3
C-9	Satellite members	If reentry toward populated area, trajectory must be corrected immediately	H-3.3
C-10	Satellite members	attitude must be kept within an operational envelope	H-4
C-11	Satellite members	attitude is outside the operational envelope, it must be corrected immediately.	H-4

Step 1.6 Define the Responsibilities

In this example we will focus on the **Propulsion Controller** only. From the text we get the following high level description of the responsibilities (from the constraints) assigned to the **Propulsion Controller**

The propulsion controller implements the lowest level of automation related to controlling the high-speed loops of the propulsion elements through an automated controller. The primary responsibilities are to:

- Produce thrust according to the requested commands by controlling, for example, the pressure of tanks, the amount of electrical power to a Hall effect thruster, and opening or closing of valves.
- Oversee the safety of the elements of the subsystem (not over-pressurizing a tank for example)

```
[9]: res = sb.stpa_table('Responsibilities',WB,120)
```

Add Row | Delete Row | Save Table

Filter...

ID	System	Responsibility
R-1	Propulsion Controller	Produce thrust according to the requested commands
R-2	Propulsion Controller	Oversee the safety of the elements of the subsystem (not over-pressurizing a tank for example)

Step 2 Model the Control Structure

Using the component definitions above, Identify the actions that are taken on and by each component. These actions can be in the form of a Control Action or Command (CA), a Feedback response (FB), or a message from another component that doesn't immediately result in an action (MSG)

In this example, we focused only on the actions that directly influence the **Propulsion Controller**

[10]:
```
msg = sb.stpa_table('Messages',WB)
```

Add Row | Delete Row | Save Table

Filter...

ID	From	To	Label	Type	exit_x	exit_y	entry_x	entry_y	Parent
CT1	Satellite Controllers	Propulsion Controller	Arm/Disarm	CA	0.25	1	0.25	0	1
CT2	Propulsion Controller	Satellite Controllers	Mode Thrust Telemetry	FB	0.75	0	0.75	1	1
CT3	Propulsion Controller	Propulsion Actuator	Thrust	CA	0.25	1	0.25	0	1
CT4	Propulsion Actuator	Propulsion Controller	Telemetry	FB	0.75	0	0.75	1	1
CT5	Propulsion Actuator	Velocity (orbit)	Thrust	CA	0.5	1	0.5	0	1

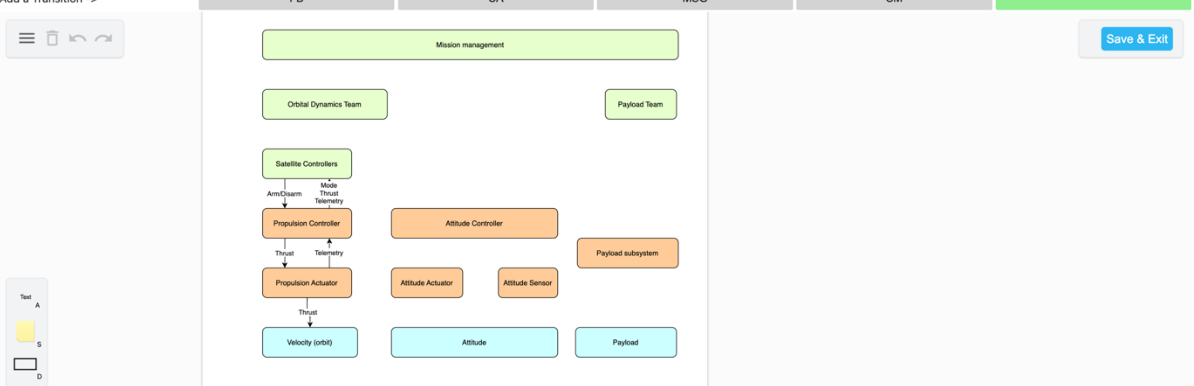
The Control Diagram

The Control_Diagram function relies on two tables, Components and Messages. Both tables in the DATA/KSAT.xlsx file have additional columns that specify the orientation, size, and direction of each component and each arrow or **action** on or from each component

[11]:
```
control = sb.cross_drawing(WB,'Control_Diagram','Components','Messages','control')
control.show_diagram()
```

Add a Component -> | HC | AC | CP | ST | Save

Add a Transition -> | FB | CA | MSG | SM



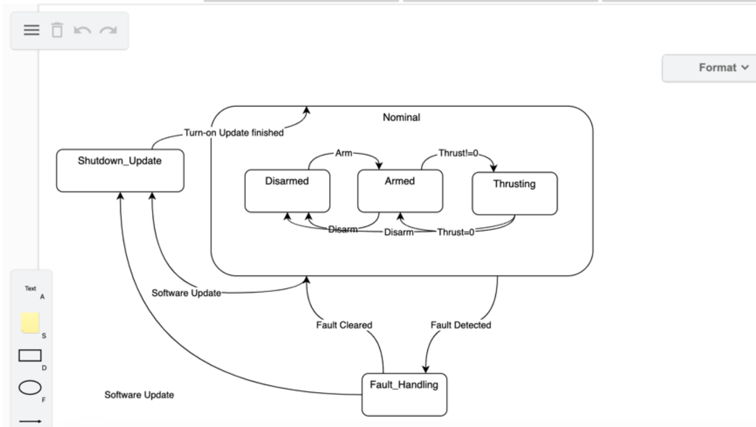
Step 2.2 Identify the High-level operational modes of the Components

In this example we dove a little deeper into the **Propulsion Actuator** component. Again imagine pulling this information from an existing activity or state diagram that defines the high level behavior of your components. The tables that define this behavior are found in KSAT.xlsx.

[12]:
```
propulsion_sm = sb.cross_drawing(WB,'State_Machine','States','Transitions','state')
propulsion_sm.show_diagram()
```

Add a Component ->	HC	AC	CP	ST	Save
Add a Transition ->	FB	CA	MSG	SM	

Save & Exit



Step 3 Identify the Unsafe Control Actions for the Propulsion Controller

From the components, constraints, responsibilities - look at each control action and define the potential unsafe or hazardous conditions that could result by providing, not providing, providing too early, too late, stopping too soon or applying too long.

Again, in this example we focused on the the Thrust command in the **Propulsion Controller**

[13]:
```
uca = sb.stpa_table('UCAs',WB)
```

Add Row | Delete Row | Save Table

Filter...

ID	Component	Type	Control Action	Description	Hazard
UCA-A1.1	Propulsion Controller	Not_Providing	Thrust	does not generate thrust when it is armed and is commanded to do it	H3
UCA-A1.2	Propulsion Controller	Providing	Thrust	generates thrust in an incorrect direction or magnitude when it is armed and commanded to do it	H3
UCA-A1.3	Propulsion Controller	Too_Early	Thrust	generates thrust when it is armed, but it was not commanded to do it	H3
UCA-A1.4	Propulsion Controller	Too_Late	Thrust	generates thrust with more than TBD of delay when it is armed and commanded to do it.	H3
UCA-A1.5	Propulsion Controller	Stopped_too_soon	Thrust	stops generating thrust when there is no alarm, it was commanded to, and the system is armed.	H3
UCA-A1.6	Propulsion Controller	Applied_too_long	Thrust	keeps generating thrust after being commanded to stop by a thrust=0 or disarm command.	H3

Step 4 Define the Scenarios

Scenarios are intended to be free form text. The javascript table currently does not have rich text capability. However, the next steps of the process will guide you as to how to take the free form text and formalize it.

Consider UCA-A1.1: from Pages 127-128

[14]:
```
sen = sb.stpa_table('Scenarios',WB)
```

Add Row | Delete Row | Save Table

Filter...

ID	UCA	Scenario
S-1	UCA-A1.1	The Propulsion subsystem physical controller goes into fault-handling or shut-down when a propulsion command is issued due to a physical controller failure, causi ng the thrust not to be generated. As a result, the maneuver is flawed, and the satellite ends in an inadequate orbit.
S-2	UCA-A1.1	The propulsion subsystem controller is armed and is commanded to generate thrust by a propulsion command. The propulsion subsystem does not generate thrust i n because the algorithm to control the actuators is not designed for the particular actuators in the satellite (it was incorrectly updated with the software for another satellite). As a result, no thrust is generated, the orbital maneuver is flawed, and the satellite ends in an inadequate orbit.
S-3	UCA-A1.1	The propulsion subsystem controller is armed and is commanded to generate thrust by a propulsion command. The propulsion subsystem does not generate thrust because the algorithm to control the actuators become inadequate over time. This can happen if, for example, a valve degraded and needs additional power to ope n. As a result, no thrust is generated, the orbital maneuver is flawed, and the satellite ends in an inadequate orbit. [
S-4	UCA-A1.1	The propulsion subsystem controller is armed and is commanded to generate thrust. The propulsion subsystem does not generate thrust because provided informat ion during the launch preparation of the satellite incorrectly loaded the amount of propellant as empty. As a result, no thrust is generated, the orbital maneuver is fla wed, and the satellite ends in an inadequate orbit.
S-5	UCA-A1.1	The propulsion subsystem controller is armed and is commanded to generate thrust by a propulsion command. The actuators are behaving as expected, but the pro

Step 4.1 Refine and Formalize the Scenarios

In order to use STPA to generate machine readable **Reference Monitors** that protect a system from potential hazardous control actions, the scenarios must be formalized. Simply, this means applying the ontology defined above in the Control Diagram, the State Diagram, and the UCAs to the textual description. This also allows us to identify any additional "internal" variables that might guide decisions of the **Propulsion Controller**

In each scenario, highlight the any states or modes, component or subsystem of interest, and any possible variables:

Legend

- Mode or State
- Conditions
- Component
- Timing
- Response, Result, or Hazard

Scenario 1:

The Propulsion Controller goes into fault-handling or shut-down mode when a Thrust command is issued due to a Propulsion_Actuator_Fault, causing the Thrust not to be generated. As a result, the maneuver is flawed, and the satellite ends in an inadequate orbit.

Scenario 2:

The Propulsion Controller is armed. and is commanded to generate Thrust The Propulsion Actuator does not generate thrust because the algorithm is not designed for the particular Propulsion Actuator in the satellite (it was incorrectly updated with the software for another satellite - i.e. Actuator_Miss_Match_Fault). As a result, no thrust is generated, the maneuver is flawed, and the satellite ends in an inadequate orbit.

Step 5 Design a Reference Monitor from Scenarios

Step 5.1 Create Requirements State Machine Language Table

Consider the Propulsion Controller State Machine provided earlier. This can be translated into the Requirements State Machine Language

```
[17]: propulsion_sm.show_diagram()
```

```
GridspecLayout(children=(Label(value='Add a Component ->', layout=Layout(grid_area='widget001', height='auto',…
HBox(children=(Diagram(cell_ids=('0', '1', 'S4', 'S1', 'S2', 'S3', 'T1', 'T2', 'T3', 'T4', 'T5', 'S5', 'S6', '…
```

A STATE MACHINE or STATECHART can be translated to plain english in the form of scenarios. For example, the Propulsion Controller State Chart has 6 states, three parent states and three child states, and has 10 transitions. Consider the transition from the **ARMED** to **THRUSTING** states.

LEGEND

- Mode or State
- Conditions
- Component
- Timing
- Response, Result, or Hazard

Propulsion Controller State Chart Scenario transitioning from ARMED to THRUSTING:

If the Propulsion Controller receives telemetry feedback that the Satellite is Deviating from Orbit **AND** is in the Nominal:Armed state **THEN** the next state **SHALL** be the Nominal:Thrusting **AND** the Thrust Control Action output will be generated.

The Requirements State Machine Language (**RSML**) is a method to translate a statechart into a tabular form amenable to both data entry and analysis. The columns S1 - S10 represent traces or conditions from input to output of the behavior of the Propulsion Controller StateChart. S1-S10 represent the logic of the statechart in disjunctive normal form. Meaning, the rows represend **AND** conditions and the columns represent **OR** conditions.

```
[16]: rsml = sb.stpa_table('Propulsion_RSML',WB)
```

Add Row | Delete Row | Save Table

Filter...

Parent	Variable Type	Variable	Condition	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
Propulsion Controller	Input CA	Arm_Command	Arm						T				
			Disarm							T	T		
	Input FB	Telemetry	In Orbit									T	
			Deviating From Orbit										T
Propulsion Controller	State	Current_State	Nominal		T		T		T	T	T	T	T
			Shutdown_Update	T		T							
			Fault_Handling					T					
	PMV	Software_Update	Finished	T									
			Requested		T	T							
		Fault	Detected				T						
			Cleared					T					

Step 5.2 Add Scenarios the RSML Table

Now that we have an RSML table that represents the behavior of the current design, we can evaluate the unsafe control actions and identify where in the system these actions could occur.

For the purposes of this demonstration, we will translate UCA A1.1 - Scenario 2 (page 127).

UCA A1.1 - Scenario 2:

The Propulsion Controller is armed. and is commanded to generate Thrust The Propulsion Actuator does not generate thrust because the algorithm is not designed for the particular Propulsion Actuator in the satellite (it was incorrectly updated with the software for another satellite - i.e. Actuator_Miss_Match_Fault). As a result, no thrust is generated, the maneuver is flawed, and the satellite ends in an inadequate orbit.

Hopefully it can be seen already that there is a descrepancy in the State Machine model of the the Propulsion Controller and the following scenario. Namely, there is no command input to the Propulsion Controller to generate thrust. Also, there is no feedback to the propulsion controller that there is an Actuator miss match. We will modify the scenario slightly based on RSML-S10:

Revised UCA A1.1 - Scenario 2 (RSML S11)

If the Propulsion Controller is in the Nominal:Armed state

AND

receives telemetry feedback that the Satellite is Deviating from Orbit ***Changed to represent telemetry feedback as the thrust command*

AND

the Propulsion Controller provides the Thrust Control Action

AND

Propulsion Actuator does not generate Thrust because the algorithm is not designed for the particular in the satellite (it was incorrectly updated with the software for another satellite - i.e. Actuator_Miss_Match_Fault).

THEN

no Thrust is generated

AND

The Hazard the maneuver is flawed, and the satellite ends in an inadequate orbit occurs

---

[17]:
```
rsml_sen = sb.stpa_table('P_RSML_Scenario',WB)
```

Add Row | Delete Row | Save Table

Filter...

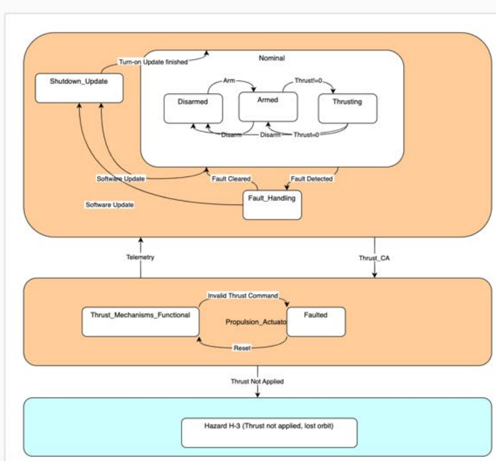
Parent	Variable Type	Variable	Condition	S10	S11
Propulsion Controller	Input CA	Arm_Command	Arm		
			Disarm		
	Input FB	Telemetry	In Orbit		
			Deviating From Orbit	T	T
Propulsion Controller	State	Current_State	Nominal	T	T
			Shutdown_Update		
			Fault_Handling		
	PMV	Software_Update	Finished		
			Requested		
		Fault	Detected		
			Cleared		

[18]:
```
propulsion_scenario = sb.cross_drawing(WB,'PState','PS_Scenario','PT_Scenario','state')
propulsion_scenario.show_diagram()
```

Add a Component -> | HC | AC | CP | ST | Save

Add a Transition -> | FB | CA | MSG | SM

Save & Exit



Step 5.3 Defining a new Responsibility

Once the hazard has been identified, a potentially a new constraint and a new responsibility can be added. Consider the option that a timer is used in conjunction with the telemetry error. This would introduce two new internal variables. For simplicity, consider refining the current telemetry variable to have three possible values rather than just two. This is simplier than a real valued variable and is an abstraction of the actual control system to illustrate the point. However, often these abstractions can be implemented in hardware.

Consider the Telemetry input to have the values [On Orbit, Near Orbit, Far from Orbit]. This provides a relative scale of error of the satellite vs its planned orbit.

Consider the **Scenario**: **LEGEND**

- Mode or State
- Conditions
- Component
- Timing
- Response, Result, or Hazard

In state=Nominal:Thrusting **IF** the Timer > X seconds **AND** the Telemetry=Far_From_Orbit the Propulsion Controller **SHALL SATISFY** state=Shutdown_Update:Revert_to_last_firmware

```
[20]: Res_new = sb.stpa_table('Responsibilities_New',WB,140)
```

Add Row | Delete Row | Save Table

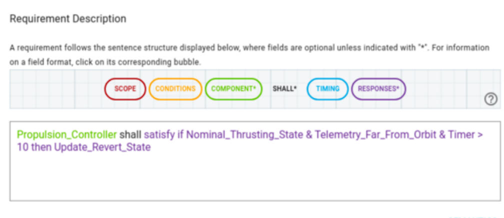
Filter...

ID	System	Responsibility
R-1	Propulsion Controller	Produce thrust according to the requested commands
R-2	Propulsion Controller	Oversee the safety of the elements of the subsystem (not over-pressurizing a tank for example)
R-3	Propulsion Controller	In state=Nominal:Thrusting IF the Timer > X seconds AND the Telemetry=Far_From_Orbit the Propulsion Controller SHALL SATISFY state=Shutdown_Update:Revert_to_last_firmware

Step 5.2 Translate RSML to FRET Requirements

"FRET is a framework for the elicitation, specification, formalization and understanding of requirements. Users enter system requirements in a specialized natural language. FRET helps understanding and review of semantics by utilizing a variety of forms for each requirement: natural language description, formal mathematical logics, and diagrams. Requirements can be defined in a hierarchical fashion and can be exported in a variety of forms to be used by analysis tools." SOURCE

The color coding legend for the formalization of STPA scenarios was derived from the FRET grammar.



Propulsion_Controller shall **satisfy if Nominal_Thrusting_state & Timer > 10 & Telemetry_Far_From_Orbit then Update_Revert_State**

FRET takes the natural language requirement and converts it to Past Time Linear Temperal Logic (PLTL)

(O (((Nominal_Thrusting_State & Telemetry_Far_From_Orbit) & (Timer > 10)) -> Update_Revert_State))

This can also be translated to RSML and the State Machine views

RSML:

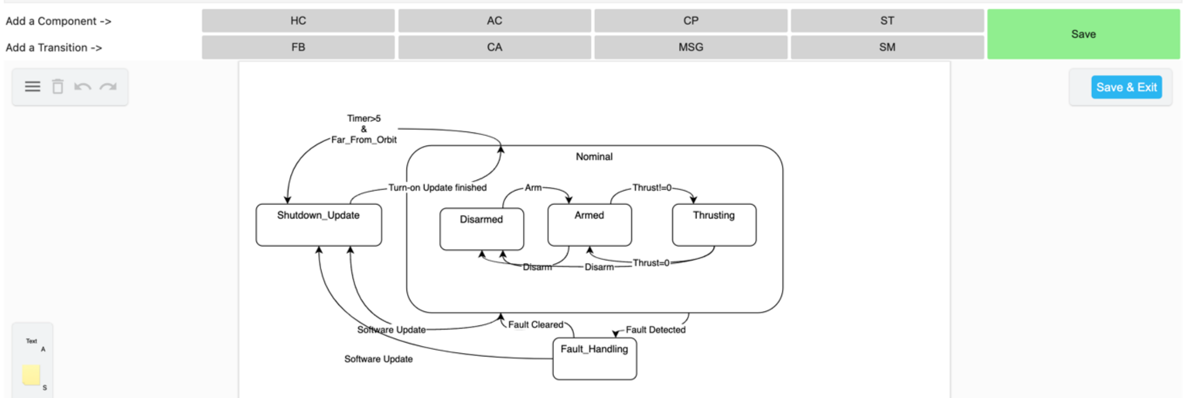
```
[21]: p_rsml = sb.stpa_table('P_RSML_REVISED',WB)
```

Add Row | Delete Row | Save Table

Filter...

Parent	Variable Type	Variable	Condition	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11
Propulsion Controller	Input CA	Arm_Command	Arm						T					
			Disarm							T	T			
	Input FB	Telemetry	On Orbit									T		
			Near Orbit										T	
			Far From Orbit											T
Propulsion Controller	State	Current_State	Nominal		T		T		T	T	T	T	T	T
			Shutdown_Update	T		T								
			Fault_Handling					T						
	PMV	Software_Update	Finished	T										
			Requested		T	T								
		Timer	> 5											T

```
[22]: propulsion_sm_revised = sb.cross_drawing(WB,'PState','PS_REVISED','PT_REVISED','state')
      propulsion_sm_revised.show_diagram()
```

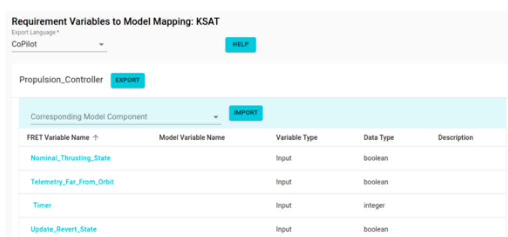


Step 5.3 Use OGMA to translate FRET Requirements into Reference Monitor

"Ogma is a tool to facilitate the integration of safe runtime monitors into other systems. Ogma extends Copilot, a high-level runtime verification framework that generates hard real-time C99 code." SOURCE

Once the requirements / scenarios are formalized, they can be exported to a json specification.

- Before export you need to declare the types of all the variables



Requirement Variables to Model Mapping: KSAT

Export Language *

CoPilot

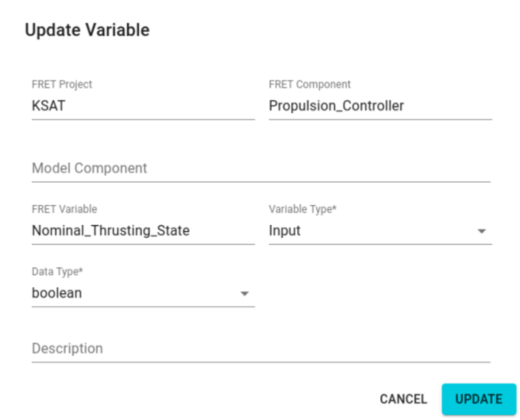
HELP

Propulsion_Controller EXPORT

Corresponding Model Component IMPORT

FRET Variable Name	Model Variable Name	Variable Type	Data Type	Description
Nominal_Thrusting_State		Input	boolean	
Telemetry_Far_From_Orbit		Input	boolean	
Timer		Input	integer	
Update_Revert_State		Input	boolean	

Most of the time, the variable type will be input or output. The c code that is generated is intended to be used in conjunction with other existing code. So, the reference monitor will use the inputs and outputs specified. Only Process Model Variables could be internal depending on how you decide to implement the monitor.



Output of formalized requirement (Propulsion_ControllerSpec.json)

Once you export the file, FRET will ask for a location for a zip. Extract the contents. Each json file in the container will be defined by the component. For example, Propulsion_Controller.

Below is a sample json output:

```
{
"Propulsion_ControllerSpec":{
"Internal_variables":[],
"Other_variables":
[
  {"name":"Nominal_Thrusting_State", "type":"bool"},
  {"name":"Telemetry_Far_From_Orbit", "type":"bool"},
  {"name":"Timer", "type":"int"},
  {"name":"Update_Revert_State", "type":"bool"}
],
"Functions":[],
"Requirements":[{
"name": "1", "ptLTL": "(O (((Nominal_Thrusting_State & Telemetry_Far_From_Orbit) & (Timer > 10)) -> Update_Revert_State))",
"CoCoSpecCode": "O((((Nominal_Thrusting_State and Telemetry_Far_From_Orbit) and (Timer > 10)) => Update_Revert_State))",
"fretish": "Propulsion_Controller shall satisfy if Nominal_Thrusting_State & Telemetry_Far_From_Orbit & Timer > 10 then Update_Revert_State"}
]}}
```

Assuming the tools have been installed, in a terminal type:

```
ogma fret-component-spec --cocospec --fret-file-name Propulsion_ControllerSpec.json > Propulsion_ControllerSpec.hs
```