FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Resilience Modeling and Model-based Design for CPS

Gabor Karsai, Daniel Balasubramanian, Abhishek Dubey, Tihamer Levendovszky, Nag Mahadevan

Vanderbilt University/ISIS

Berkeley UNIVERSITY OF CALIFORNIA

Massachusetts Institute of Technology

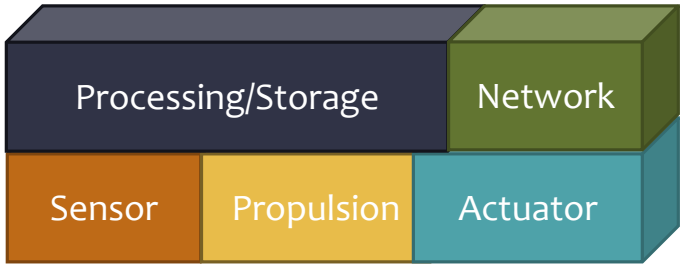
NSF

MICHIGAN

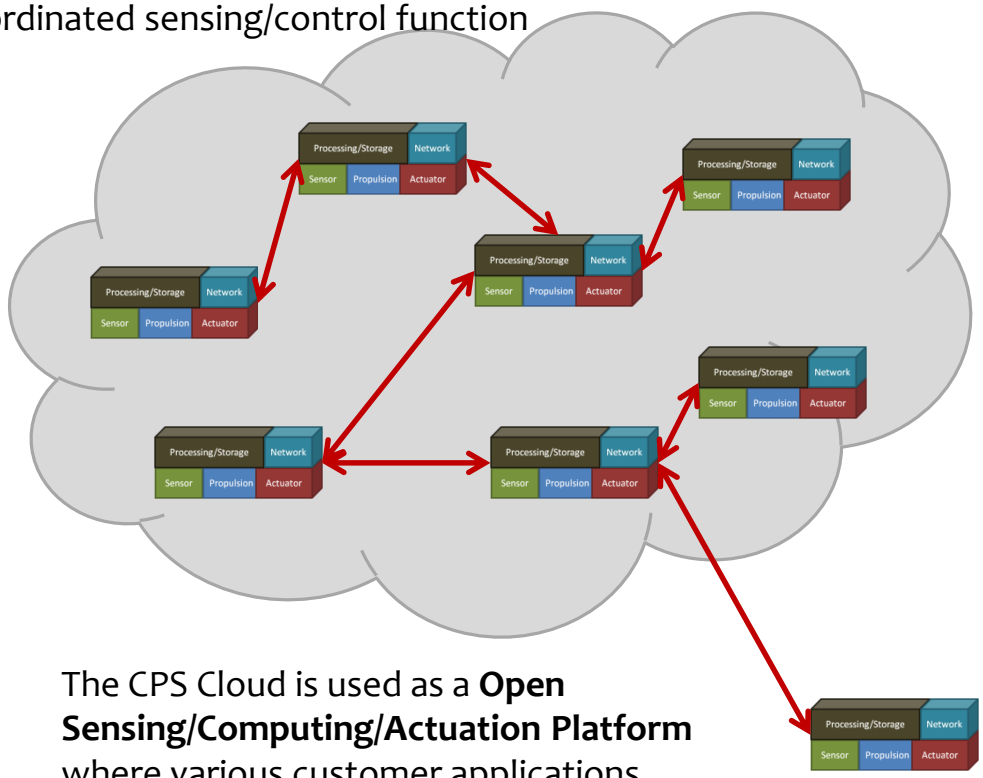
VANDERBILT UNIVERSITY

CPS Cloud:
A Distributed Sensor/Control Network Platform

Networked node with local processing and storage, sensors, actuators, and propulsion system:



Nodes for an ad-hoc network that has 1+ ground-link and performs a coordinated sensing/control function



Examples:

- Swarm of UAVs performing tornado damage surveillance
- Fleet of UUVs performing collecting climate change data from oceans
- *DARPA System F6:* Fractionated Spacecraft – A Space Global Common

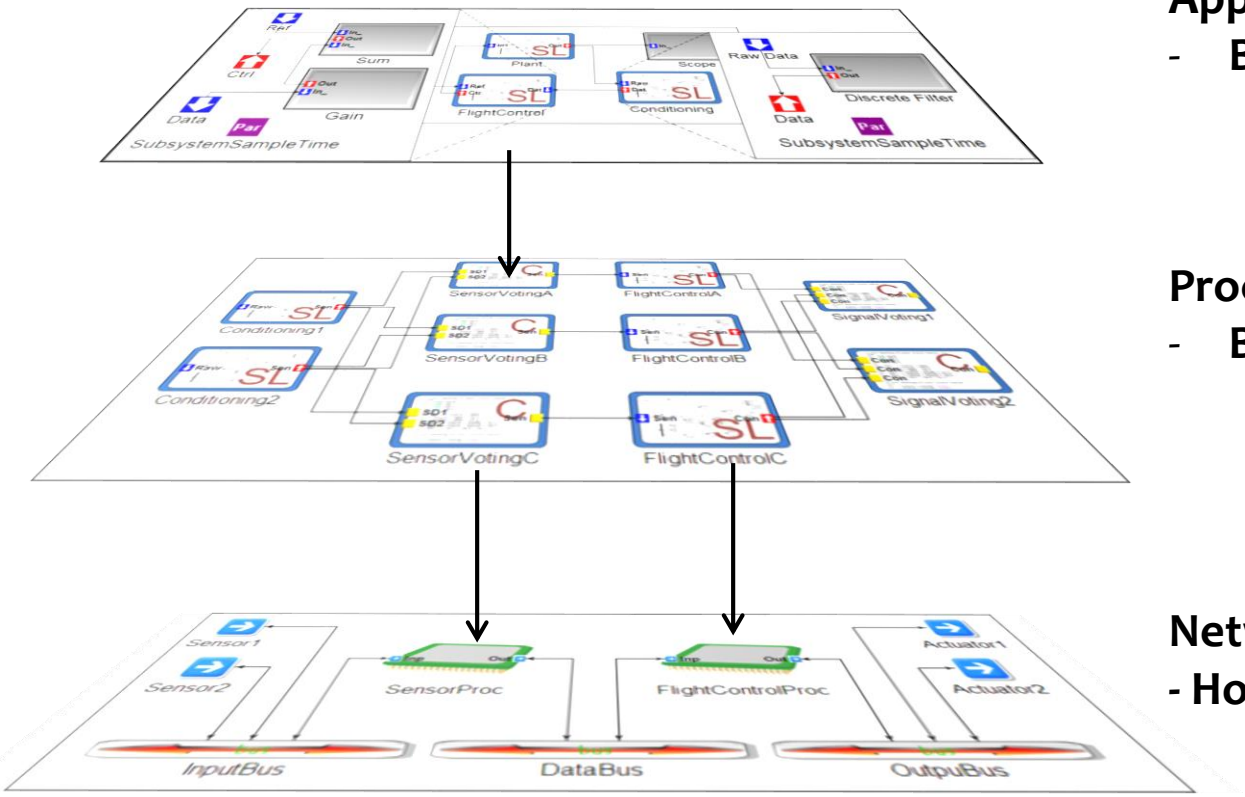
Challenges:

- Networked, distributed control
- Fault-resilience
- Applications with different trust and security levels must run side-by-side

The CPS Cloud is used as a **Open Sensing/Computing/Actuation Platform** where various customer applications can run, side-by-side.

FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

CPS Applications deployed on the Platform



Applications
- **Built from Processes**

Processes
- **Built from Components**

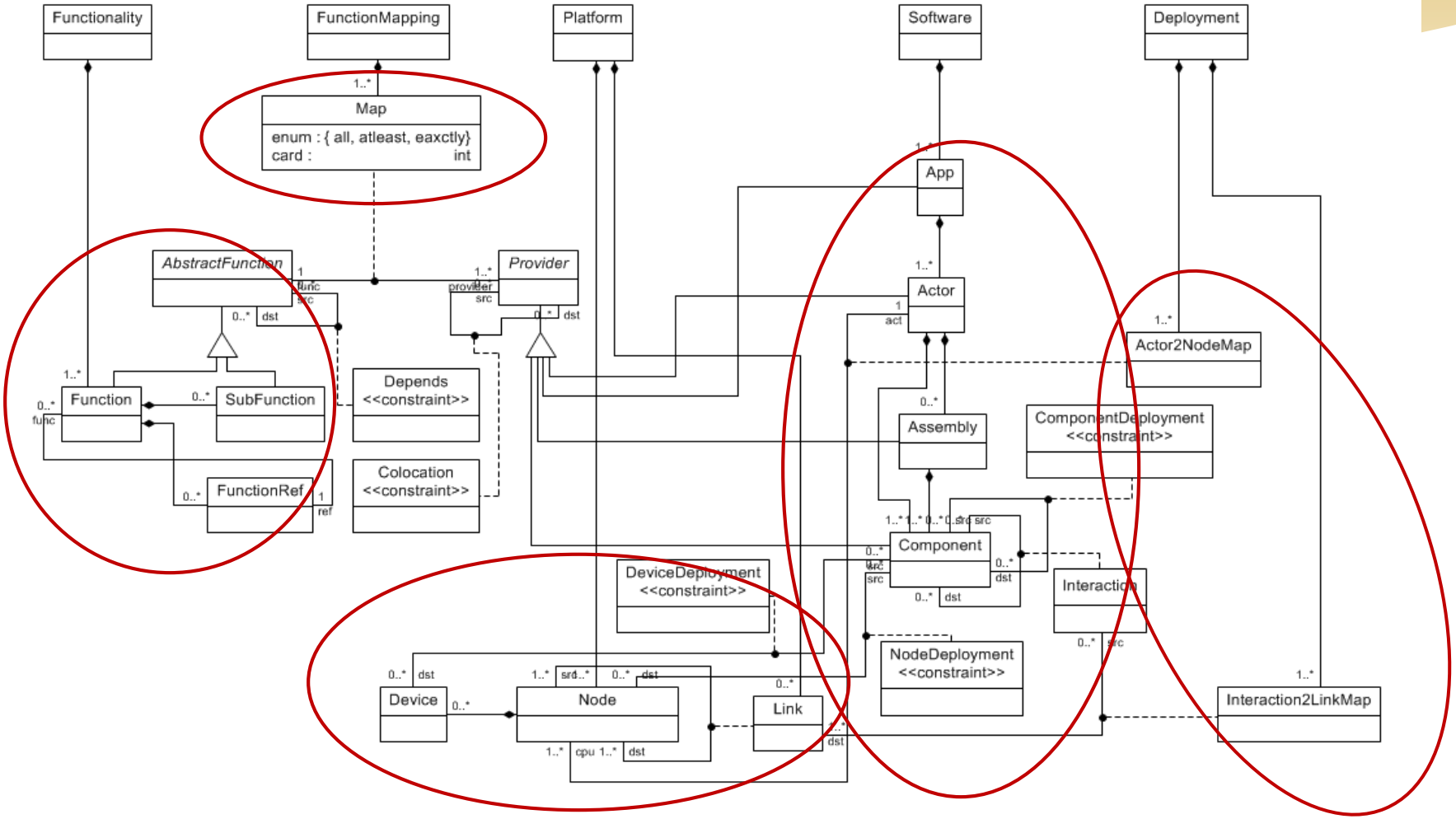
Network nodes
- **Host processes**

FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

CPS Cloud Application Platform Architecture 'Language'

Concept	Definition	Notes
Node	Computing node	Hosts processes
Link	Network link	Facilitates interactions
Component	Software component	Unit of concurrency
Process	Software process	Hosts components
Application	Software applications	Consists of processes
Interaction	Component interactions	Pub/sub or service (sync/async)
Device	Physical device	Assigned to node
Function	Functionality	Decomposable with dependencies

Resilient CPS Platform Concepts



FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Why resilient?

* A System Function *can be* allocated to various (combinations of) providers: Applications / Processes / Components
* Processes / Components *can be* allocated to various (combinations of) platform Nodes
* When a Node / Link / Process / Component fails (compromised), functionality can be restored by an
  * alternative allocation of *functions* to *providers*, or
  * alternative allocation of *providers* to *platform* nodes

Evaluating Architectures

Architecture: set of related designs.

- Related designs
  - Common elements are captured as a seed design
- A set of…
  - Variations are captured as configuration constraints
- Resilience of an architecture
  - Capability of eventual recovery from loss of functionality
- Comparison: Resilience metric
  - A pair of integers
  1. Measures the margin of recovering capabilities in the *worst case* (redundancy of the system along the most vulnerable/critical path)
  2. Measures the margin of recovering capabilities in the *optimistic case* (overall redundancy of the system)

Definitions

* Component availability refers to the availability of a component for usage at any time instant during the operation. I.e., it is deployed and is active.
* Function availability refers to the availability of a function for operation. For a function to be available, all the components required for the realization of this function should be available.
* If a function can be carried out even when a component becomes unavailable, then we can conclude that there is active redundancy in the function with respect to that component.
* If a function can be carried out even when a component becomes unavailable by activating/deploying another set of components or migrating the affected component to another node, then it can be concluded that there is deployment redundancy in the function with respect to that component.

Resilience metric[m1, m2]

- Definition 1
  - The *worst case resilience* is defined as the least number of failures that will make the mission infeasible.
  - The *best case resilience* is defined as the maximum number of failures that can be sustained while the mission remains feasible
- Definition 2
  - m1: # of node disjoint paths in the reliability block diagram
  - m2: # of parallel paths in the reliability block diagram
- Measure the level of *active* and *deployment* redundancy in the system
- The set of deployment constraints in the system and number of alternative choices affect these numbers. Example of deployment constraints:
  - HRImaging component requires a node with HR camera device.
  - A node cannot host more than one instance of a high performance computing component.
  - Organization A's actors must never be collocated on a node with Organization B's actors.

Calculating the resilience metric

* Encode the problem as a Satisfiability Modulo Theory (SMT) problem over integers
  * SMT problems are Boolean satisfiability problems (SAT) in which some of the binary variables are replaced by predicates over a suitable set of non-binary variables. A predicate is basically a binary-valued function of non-binary variables that relies on a 'theory'.
* Use an SMT solver to calculation the solution
  * Solution = valuation of variables
  * We use the Z3 solver (best-of-breed from MSR)

SMT Solver – Problem setup

Fact	Description
Component (instances)	Deployed software component instances that operate at the same time. The smallest software unit that can fail. Pre-deployed, inactive backups are *not* modeled as component instances. Binary resources can be modeled as component instances, too.
Nodes	Hardware nodes that are capable of executing component instances.
Links	Interaction links between the nodes. Provided as an adjacency matrix of the graph consisting of nodes and links between them.
Countable component resources	Provided as an integer matrix: ***ccr[i,j]*** means the resource **requirement** of component ***i*** of resource ***j***
Countable node resources	Provided as an integer matrix: ***cnr[i,j]*** means the **availability** of resource ***j*** on node ***i***.
Actors	Defines component groups. If an actor fails, all included component fails.

SMT Solver: Constraint Groups

Constraints	Description	Examples
Component to node constraints	Describes the conditions under which a component can be deployed onto a node	Collocation constraints (C_1 and C_2 must/cannot be deployed on the same node); C_1 must be deployed on N_1.
Component to component dependencies	A component's availability is dependent on the availability of another component.	An image processor component C_{im} needs a camera component C_c.
Interaction constraints	In order for two components to interact, there must be a link between the nodes they are deployed on.	An image processor component C_{im} needs a connection to communicate with camera component C_c if they are deployed on a separate node.
Function realization	Describes the dependency between a function and the components it uses.	Function F_1 is realized by components C_1 and C_2.
Functional decomposition	Describes the dependency between functions.	Function F_1 uses functions F_2 or F_4.

SMT Solver: Basic System Configuration Primitives

Primitive	Description	Examples in Python
Equals (variable list)	**Input:** a list of variables; **Output:** constraint set that equals all the variables in chain	**Components of the same actor must be deployed an the same node.** *act_c2n2d = [self.Equals([c2n[compIx][n] for compIx in actor]) for n in range(self.NO_OF_NODES) for actor in self.actors]*
Enabled (component)	**Input:** an index of the component in the component instance node matrix. **Output:** a Boolean expression that is true if the component is assigned to a node; false otherwise	**C_3 depends on C_2.** *Implies(self.Enabled(3),self.Enabled(2))* **C_0 depends on C_1 or C_2.** *Implies(self.Enabled(0),Or(self.Enabled(1), self.Enabled(2)))*
Communicates (componentA, componentB)	**Input:** two components **Output:** a constraints that enforces that there is a link between the nodes the components are deployed on. If the two components are on the same node the constraint is always satisfied.	**Connections to link; if C_3 communicates with C_2 and they are on different node, there should be a link between the nodes.** *comm = self.Communicates(3,2)*

SMT Solver:

Basic Node-Component Assignment Primitives

Primitive	Description	Examples in Python
CollocateComponents (component list)	**Input:** a list of component instances; **Output:** an constraint that ensures that the component instances must be assigned to the same node	**C_2 collocated with C_0 OR C_2 is collocated with C_1.** *Or(And(self.CollocateComponents([2,0])),And(self.CollocateComponents([2,1])))*
DistributeComponents (component list)	**Input:** a list of component instances; **Output:** an constraint that ensures that the component instances must be assigned to different nodes	**C_2 cannot be collocated with C_0 AND C_2 cannot be collocated with C_1.** *And(And(self.DistributeComponents([2,0])), And(self.DistributeComponents([2,1])))*

SMT Solver: Basic Function Realization Primitives

Primitive	Description	Examples in Python
Implies (function, binary component expression)	Dependency between functions and components: a function requires the binary component expression to be true.	**F_0 requires C_2** *self.solver.add(Implies(f[0], self.Enabled(2)))*
ForceExactly (function, component list, n)	**Input:** a function and a list of components; a positive integer n **Output:** constraints that makes sure that **exactly** *n* of the components in the list must be enabled to provide the function	**F_0 requires exactly two of C_0, C_1, C_2** *f2c = self.ForceExactly(f[0], [0,1,2],2);* *self.solver.add(f2c)*
ForceAtleast (function, component list, n)	**Input:** a function and a list of components; a positive integer n **Output:** constraints that enforces that **at least** *n* of the components in the list must be enabled to provide the function	**F_0 requires at least two of C_0, C_1, C_2** *f2c = self.ForceExactly(f[0], [0,1,2],2)* *self.solver.add(f2c)*
ForceAtmost (function, component list, n)	**Input:** a function and a list of components; a positive integer n **Output:** constraints that enforces that **at most** *n* of the components in the list must be enabled to provide the function	**F_0 requires at most two of C_0, C_1, C_2** *f2c = self.ForceExactly(f[0], [0,1,2],2)* *self.solver.add(f2c)*

SMT Solver: Failure behavior primitives

Primitive	Description	Examples in Python
ComponentFails (component)	**Input:** a component that fails **Output:** None Adds a constraint that makes sure that the component cannot be assigned to any node.	**Component C_2 failed** *s= GPSConfigurationSolver()* *s.ComponentFails(2)*
NodeFails (node)	**Input:** a node that fails **Output:** None Adds a constraint that makes sure that the node cannot host any component	**Node N_2 failed** *s= GPSConfigurationSolver()* *s.NodeFails(2)*
ActorFails (actor)	**Input:** a node that fails **Output:** None Adds a constraint that makes all the component belonging to the actor fail	**Actor A_3 failed** *s= GPSConfigurationSolver()* *s.ActorFails(3)*
LinkFails (n1,n2)	**Input:** a pair of node between which a link fails **Output:** None Adds a constraint that makes sure that the components assigned to the n1 n2 cannot interact	**Tle link between N_3 and N_4 failed** *s= GPSConfigurationSolver()* *s.LinkFails(3,4)*

SMT Solver: Services / Queries

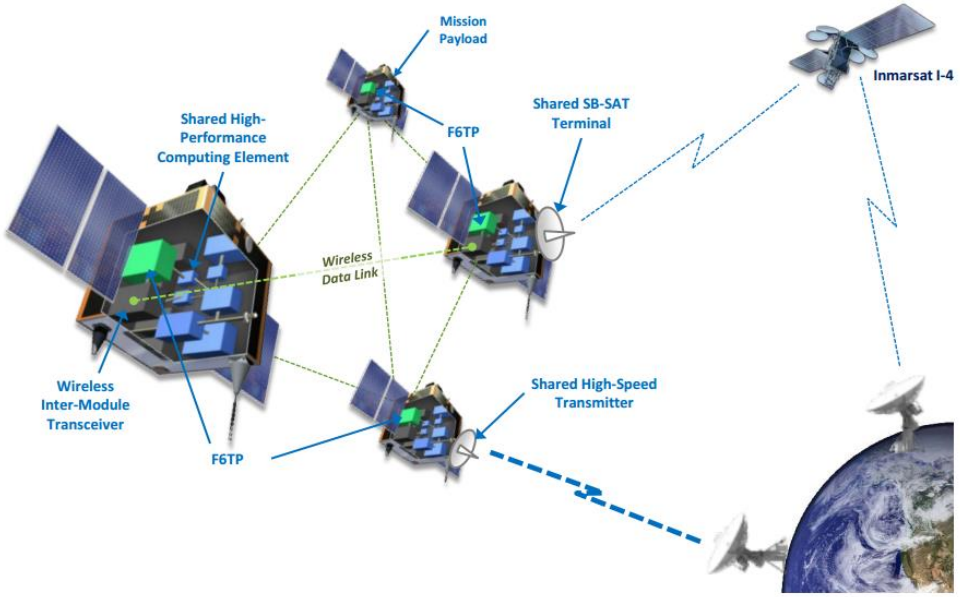
Primitive	Description	Examples in Python
Single deployment configuration	Provides a single deployment configuration that satisfies the constraints. Possible outputs: (i) a solution, (ii) no solution exists, (iii) not known.	*s= GPSConfigurationSolver()* *s.solve()*
All possible deployment configurations	Provides all possible deployment configuration that satisfies the constraints. Possible outputs: (i) solutions, (ii) no solution exists, (iii) not known.	*s= GPSConfigurationSolver()* *solutions = s.get_models()*
All independent configurations	Provides all possible deployment configuration that satisfies the constraints and do not share any components or nodes. Possible outputs: (i) independent solutions, (ii) no solution exists, (iii) not known.	*s= GPSConfigurationSolver()* *solutions = s. get_independent_models()*
Resilience Metrics	Provides resilience metrics (a pair of integers). If there are no solutions it returns [0,0].	*s= GPSConfigurationSolver()* *[m1,m2] = s.computeMetrics()*

An example

Resilience metric calculations

Satellite Resources

* Three similar satellites
* Sat 1
  * HR Camera
  * LR Camera
  * GPU
  * Ground link
* Sat 2
  * HR Camera
  * GPU
  * Ground link
* Sat 3
  * LR Camera
  * Ground Link
* All satellites have two wireless links that they can use to communicate with each other



FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Applications

* Cluster Flight Application (CFA)
  * `GroundInterface`: An actor that provides access to the ground station. The ground uses this actor to send commands to the cluster.
  * `SatelliteBusInterface`: An actor that provides access to the satellite bus hardware
  * `TrajectoryPlanner`: Runs the trajectory planning service. It receives the commands from ground and then updates the orbit.
  * `OrbitManager`: Runs the control loop. Disseminates position to other satellites and commands the satellite thruster via the bus interface to adjust the orbit as required.

Applications (continued)

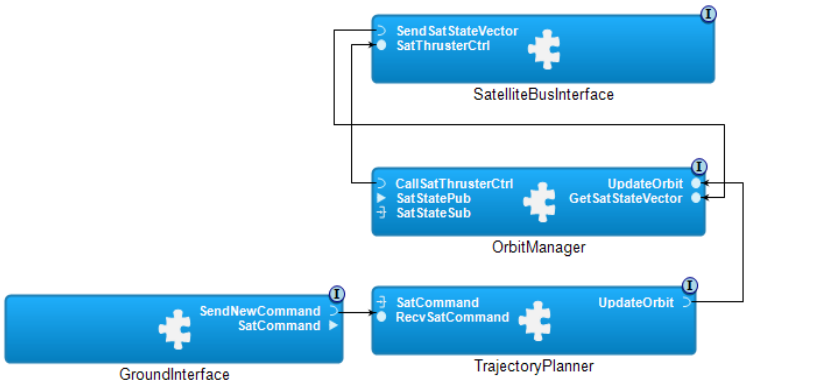
* Wide area imaging application
  * Uses the high resolution and/or low resolution cameras different nodes to create a combined image.
  * Each satellite runs an image grabber component.
    * It can provide service either through the high resolution facet or low resolution facet or both, depending upon the hardware available on the satellite.
  * Only one instance of image processor component runs in the cluster at any time.
    * But it can be redeployed as required.

Physical Resource Requirements

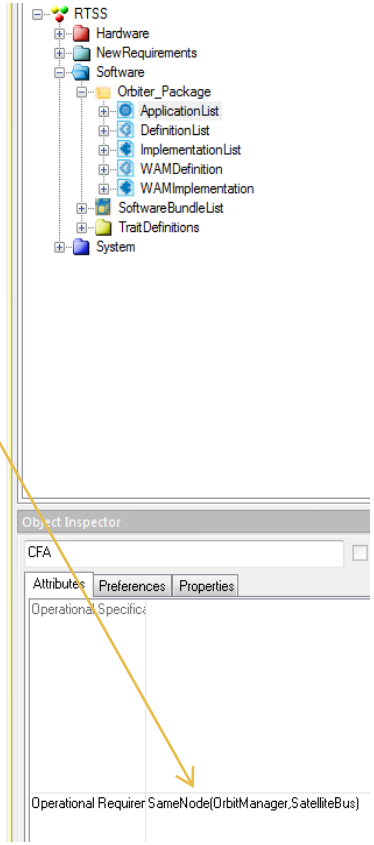
* `GroundInterface` **requires** `GroundLink`
* `ImageGrabber`
  * LR_Img: LR_CAMERA (LR_img port needs LR camera)
  * HR_Img: HR_CAMERA (HR_img port needs HR camera)
* `ImageProcessor` **1,2 requires a** `GPU`

CFA

* One instance of CFA runs on each node
* An application instance requires the orbit manager and satellite bus interface from the same node.



Trajectory planner can receive commands either by peer to peer connection of via pub sub



FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS

Operational requirements

- All components/Application have operational requirements
  - CFA Application
    - SameNode(OrbitManager,SatelliteBus)
  - OrbitManager
    - SameNode(CallSatThrusterCtrl)
    - SameNode(GetStateVector)
  - TrajectoryPlanner
    - Atleast(1, (SatCommand_Subscriber,ReceiveSatCommand))
  - ImageGrabber
    - ImageGrabber _1: Atleast(1,(HR_1,LR_1))
    - ImageGrabber _2: Atleast(1,(HR_2))
    - ImageGrabber _3: Atleast(1,(LR_3))
  - ImageProcessor
    - ImageProcessor_1: Atleast(1,GPU_1)
    - ImageProcessor_2: Atleast(1,GPU_2)
    - Atmost(1,(ImageProcessor_1, ImageProcessor_2, ImageProcessor_3))

Functional requirements

* Capture the functional breakdown required for the mission
  * Cluster flight
  * Wide area imaging
* All functions map to application/component instances
* Failure of one component/hardware resource/network link is used to compute whether the mission function is unavailable.
* Thereafter an alternative configuration (if available) can be chosen to recover the functionality.

Resilience Metric and Scenarios

* Metric = [2,23]
  * Assumption: all 6 functions are required
  * The system is 2-fault tolerant, but can operate as many as 23 faults

Scenarios:

* Complete failure of Sat2
  * ImageProcessor on Sat2 is out, another ImageProcessor on Sat1 or Sat3 should be activated.
* Failure of GPU on Sat1
  * GPU is required by the Image Processor
  * Therefore, a reconfiguration is required which activates image processor on Sat3
* Failure of Ground Link on Sat 1
  * No reconfiguration is required. The ground command is disseminated by either Sat2 or Sat3 via pub/sub ports

Summary: Resilience Modeling

* Resilient architecture can be modeled as: software components and architecture + hardware platform + functions + constraints
* A constraint solver can calculate
  * The resilience metric (to compare architectures)
  * Interesting fault scenarios that break the system (to increase resilience by design)
  * Novel deployments for the system (to do reconfiguration)
* Relies on:
  * Robust supervisory layer / platform that manages system reconfiguration

FORCES
FOUNDATIONS OF RESILIENT CYBER-PHYSICAL SYSTEMS