Model Based Engineering for Software Assurance

Lui Wang, Michel Izygon* - NASA /JSC
Spacecraft Software Engineering Branch
Software Robotics Simulation Division
* Tietronix Software Inc.
John Evans - NASA / OSMA

Presented by
Tim Crumbley
NASA Software Assurance Technical Fellow

October 1, 2019

The Challenge

- Spacecraft designers and operation stakeholders create models and artifacts of the same system with different processes, tools, and representations.

- These uncoordinated modeling approaches create locally successful products but also create a communication barrier among the various stakeholders (the “Tower of Babel” Effect).

- The same information is captured multiple times, in multiple places, with multiple representations, creating a maintenance challenge.

Addressing Engineering Complexity

- **Model Based Systems Engineering (MBSE)**

  From: document-centric (presentation slides, textual documents, miscellaneous spreadsheets)

  To: model-centric (objects, their attributes, and interrelationships)

- **Systems engineering information of:**
  components, their organization, the functions they perform, the requirements they fulfill, the behaviors they exhibit, the ways they may fail, …

MBSE Benefits

Characteristics	Benefits
Agreed-upon* meaning	Avoid ambiguity & misunderstanding
Single source of truth	Avoid replication and subsequent inconsistencies
Formal/mathematical	Computer support for checks (e.g., for missing things), analyses, queries, views

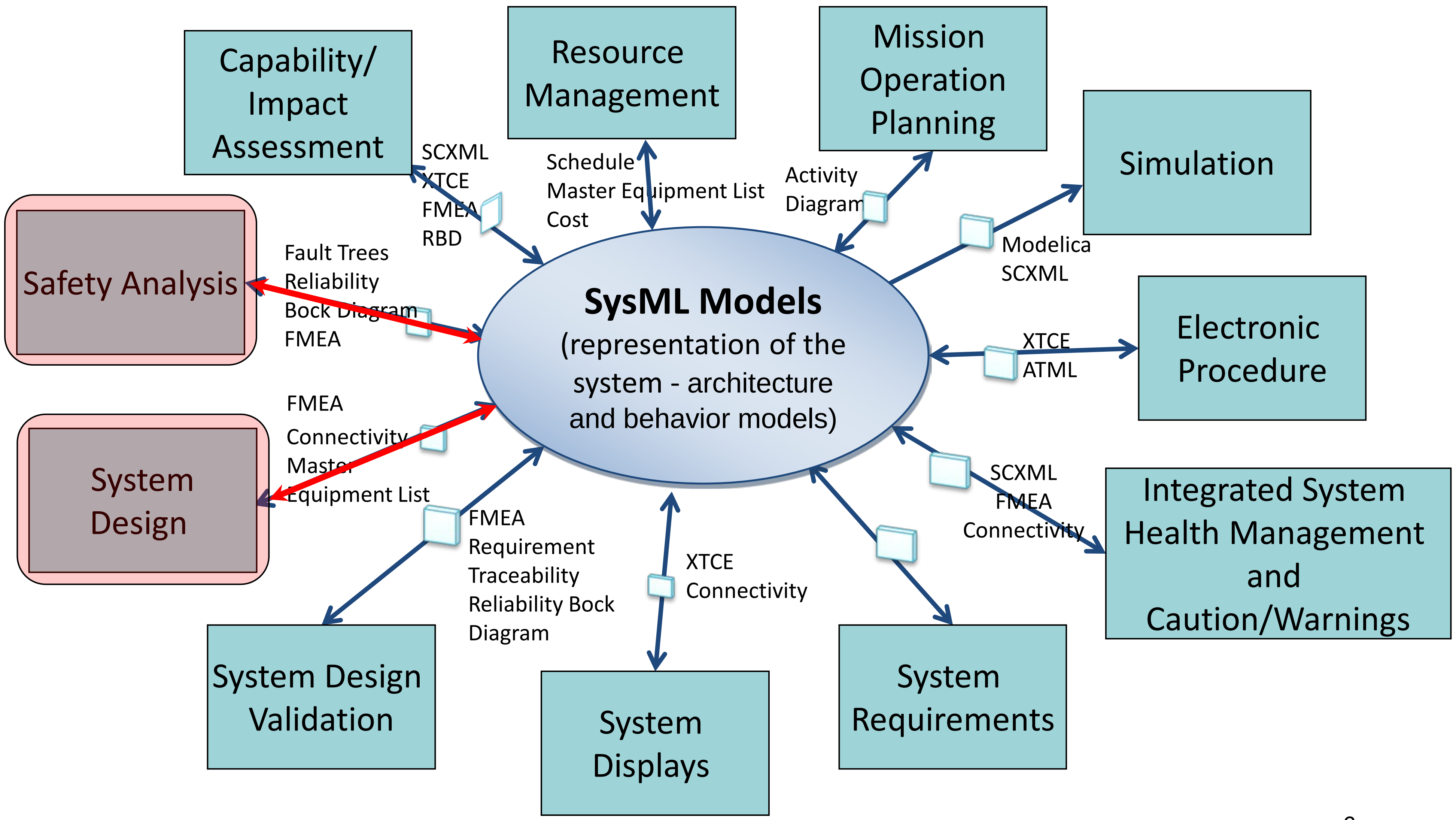
* Standard representations of generic systems engineering information (e.g., component/ subcomponent), with domain-specific extensions (e.g., “mission”, “trajectory”)

Software Assurance challenges

- Traditional approaches in performing mission assurance largely decoupled from the main system engineering activities, and occurs late in the design process.
- Mostly reactive, evaluating a system to determine if the system is safe.
- Risk based design approach starts to engage Safety and Mission Assurance earlier in the design. But still often requires creating independent model to support Safety Assurance analysis.

Uses of System Models

Model Once and Use Many Times



Experience with SysML Modeling

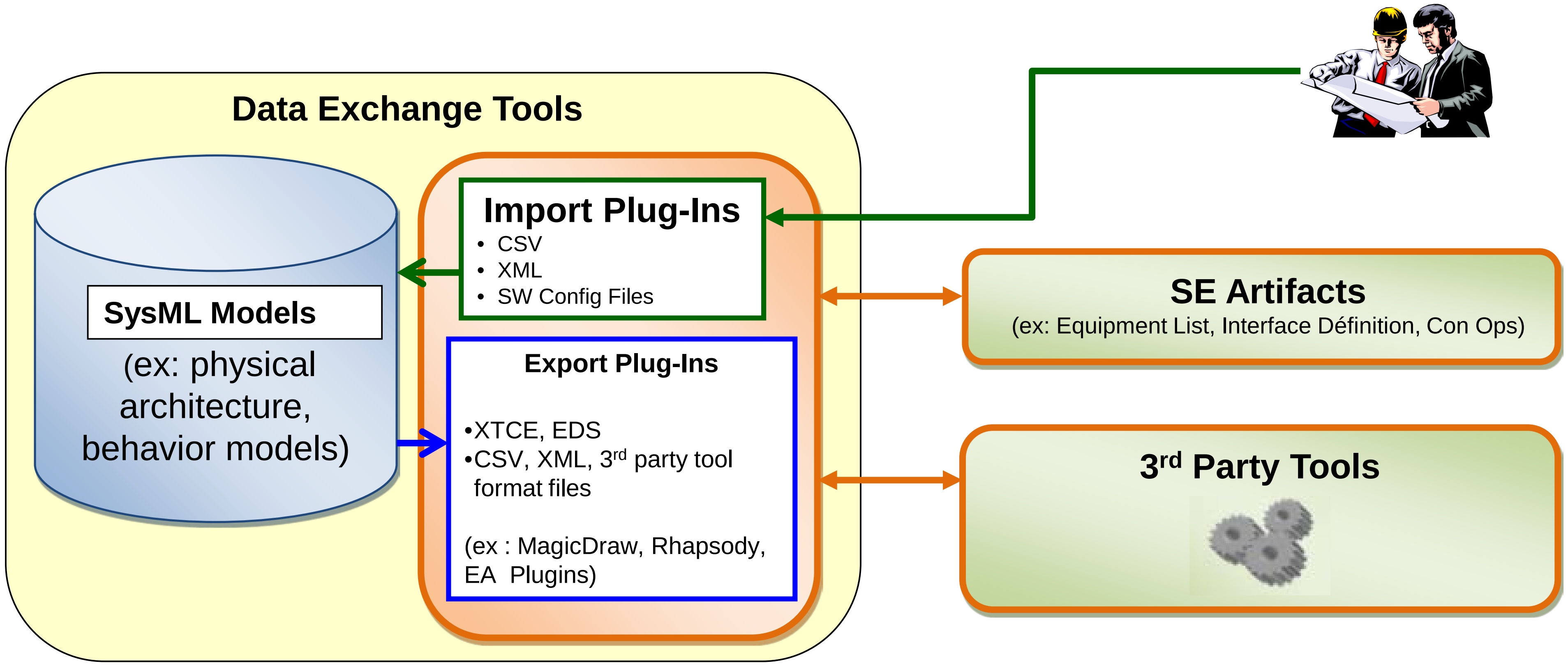
- Developed a modeling methodology
  - System Engineering Design and Analysis
  - Model Based Fault Management Engineering (MBFME)

- Developed SysML Models for Multiple NASA Projects:
  - Deep Space Habitat (DSH)/ Habitat Demonstration Unit (HDU)
  - Exploration Augmentation Module (EAM)
  - Advanced Exploration System (AES) Life Support System (LSS)
    - Cascade Distiller System (CDS)
    - Capillary Brine Residual in Containment (CapiBric) System
    - AES LSS reference architecture
  - Human Exploration Testbed Integration and Analysis (HESTIA)
  - AES Modular Power Systems (AMPS)
  - Integrated Power and Avionics System (IPAS)
  - Ascent Abort 2 – Software Modeling
  - NASA Gateway – SE&I

- Developed SysML Library Repository
  - Collection of SysML Models

Experience with SysML Tool Set Development

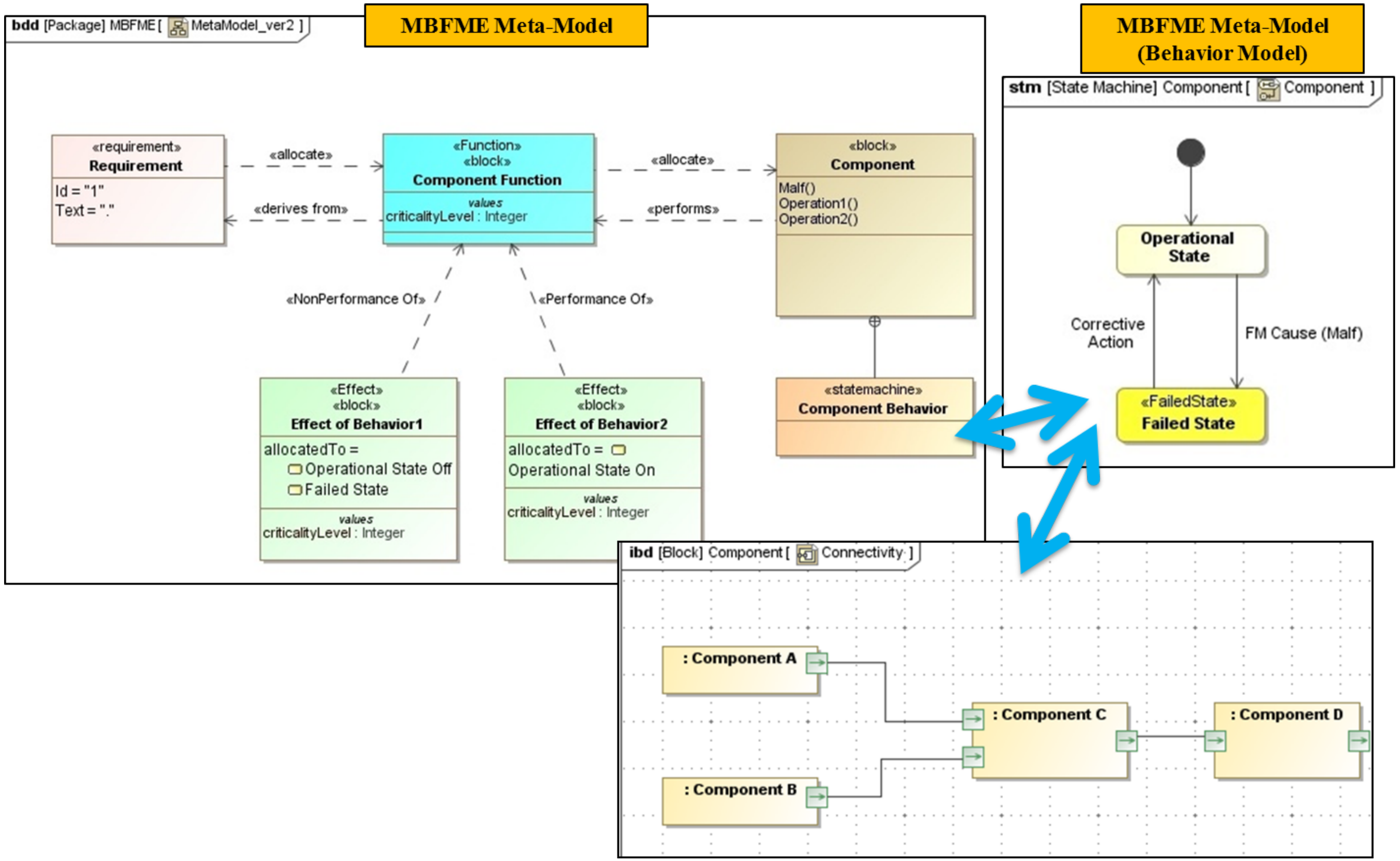
- **Provide a suite of data exchange tools**
  - *To extract System Engineering products from the models*
  - *To build models by extracting automatically or semi automatically information from existing sources*
  - *To support Fault Management engineering*



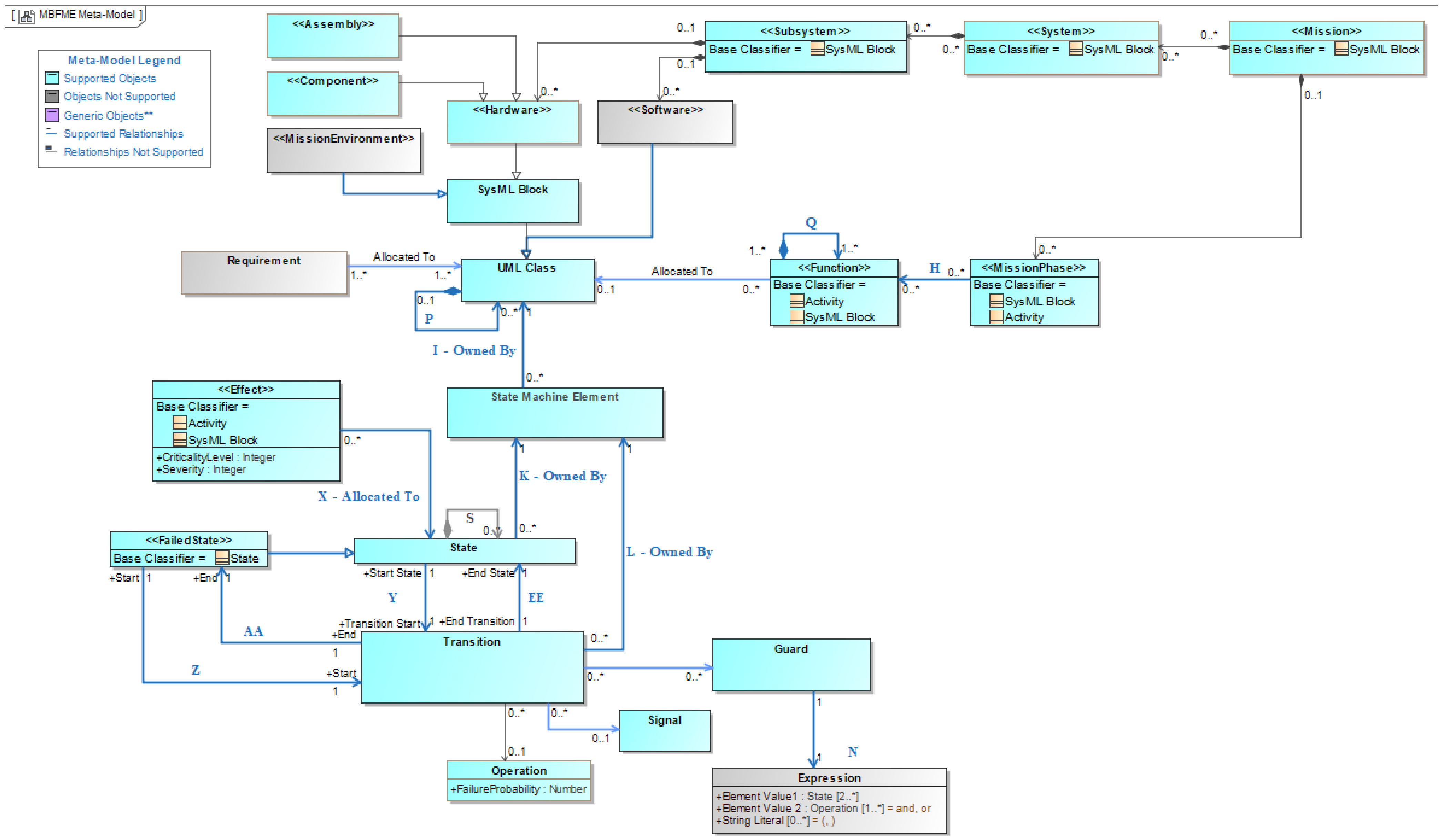
Data Exchange Tools
SysML Models
(ex: physical architecture, behavior models)
Import Plug-Ins
• CSV
• XML
• SW Config Files
Export Plug-Ins
•XTCE, EDS
•CSV, XML, 3rd party tool format files
(ex : MagicDraw, Rhapsody, EA Plugins)
SE Artifacts
(ex: Equipment List, Interface Définition, Con Ops)
3rd Party Tools

Model Based Fault Management Engineering (MBFME) Approach

- Metamodel Structure


MBFME Meta-Model
bdd [Package] MBFME [MetaModel_ver2]
«requirement» Requirement
Id = "1"
Text = "."
«allocate»
«derives from»
«Function» «block» Component Function
values
criticalityLevel : Integer
«allocate»
«performs»
«block» Component
Malf()
Operation1()
Operation2()
«NonPerformance Of»
«Performance Of»
«Effect» «block» Effect of Behavior1
allocatedTo = Operational State Off, Failed State
values
criticalityLevel : Integer
«Effect» «block» Effect of Behavior2
allocatedTo = Operational State On
values
criticalityLevel : Integer
«statemachine» Component Behavior
MBFME Meta-Model (Behavior Model)
stm [State Machine] Component [Component]
Operational State
Corrective Action
FM Cause (Malf)
«FailedState» Failed State
ibd [Block] Component [Connectivity]
: Component A
: Component B
: Component C
: Component D

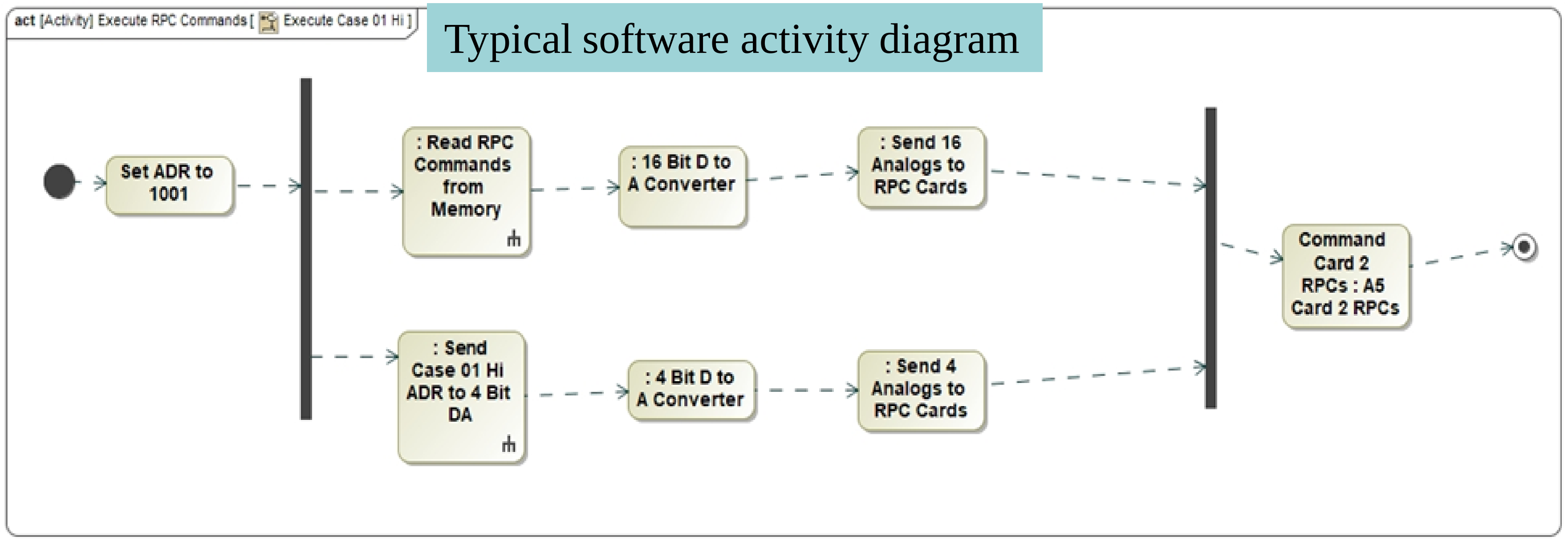
MBFME Meta Model for Hardware-Software



MBFME Hardware Modeling Approach

- Allocate Requirements to “Function” Blocks
- Allocate “Function” Blocks to “Component” Blocks
- Define a criticality level for the project
- Assign criticality levels to functions
- Identify the Operational States and Potential Failed States for a component
- Identify the immediate effects of each state
- Assign criticality levels to the “Effect” Blocks
- Identify the transitions (aka: the causes) between states
- derive Fault and Effect propagation paths and determine End (system and mission) effects

MBFME Software Modeling Approach


act [Activity] Execute RPC Commands [Execute Case 01 Hi]
Typical software activity diagram
Set ADR to 1001
: Read RPC Commands from Memory
: 16 Bit D to A Converter
: Send 16 Analogs to RPC Cards
: Send Case 01 Hi ADR to 4 Bit DA
: 4 Bit D to A Converter
: Send 4 Analogs to RPC Cards
Command Card 2 RPCs : A5 Card 2 RPCs

Methodology to extract fault propagation paths from the activity diagrams:

- Start from each Control Flow stereotyped as "Error".
- Trace to the End point of the Activity diagram.
- If encounter a Join action treat it as an AND gate. All inputs are required to generate an output. Include the additional Control Flows as part of the fault path. May need to trace the additional Control Flows back to the start of the module.
- The Immediate Effect is the first Action element encountered in the path
- Module Effect (System Effect) is the last Action element encountered before exiting the module

MBFME Hardware Software Integration

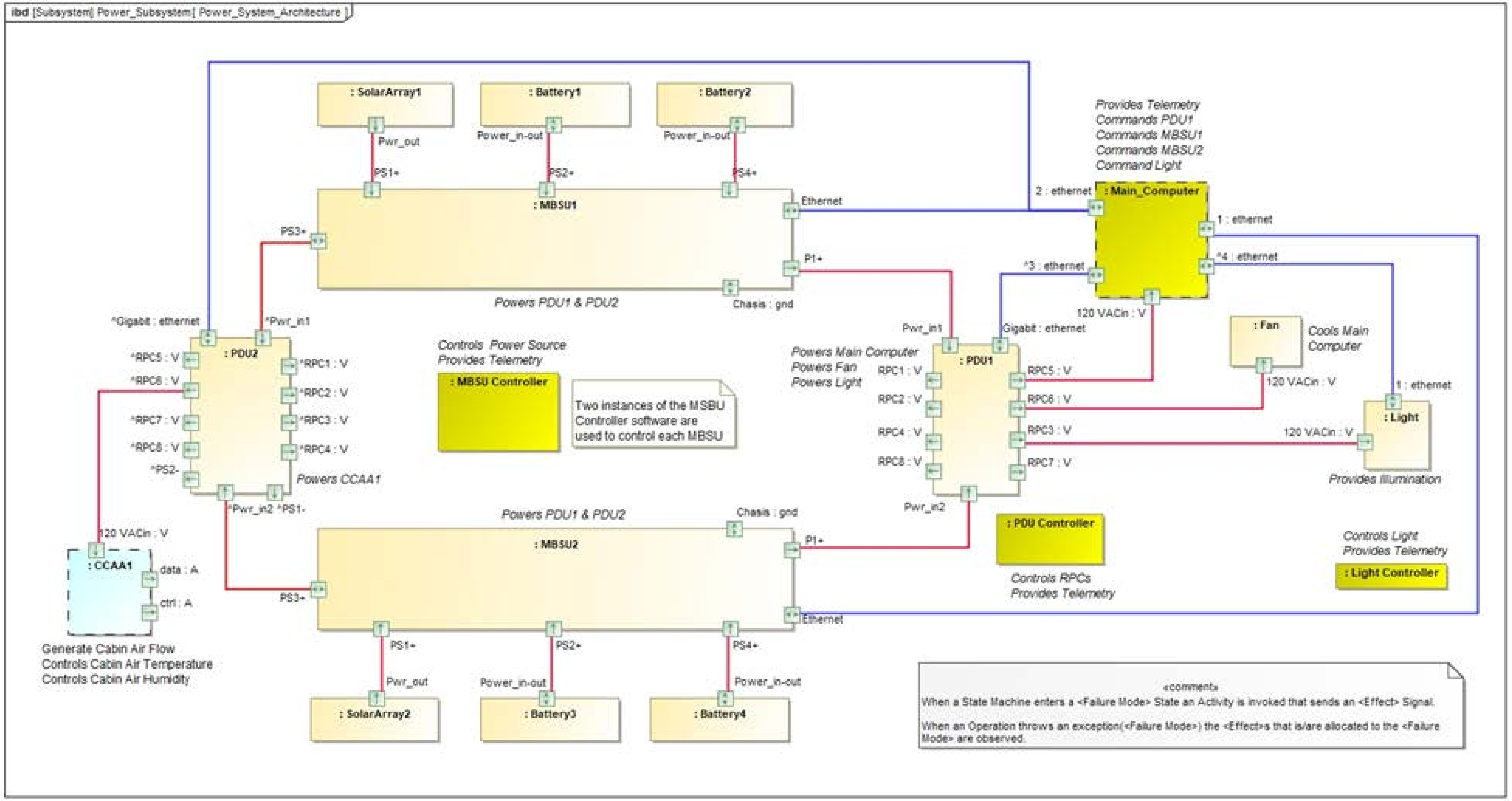
- **Modeling both Hardware and Software components:**
  - Hardware components are controlled and monitored by software components.
  - Faulty behavior of software components can cause improper behavior of the hardware and contribute toward failure of the system.
  - Faulty behavior of hardware components can cause software to fail functioning.

  - **Hardware Component**:
    - Structural: Modeled as a *Block*
    - Behavioral: Modeled using *State Machine and Activity* Diagrams
  - **Software Component**:
    - Structural: Modeled as a *Class*
    - Behavioral: Modeled using *Method* (Activity) which describes the Operation at a high level.
      - Focus on Commands, Failure of commands execution, and Effects of these failures

- **Command Mapping:**
  - Mapping from software commands to the hardware using Activity with stereotype <<*Command Mapping*>>

MBFME Hardware Software Integration

- Example of Hardware and Software components



Two instances of the MSBU Controller software are used to control each MBSU

«comment»
When a State Machine enters a <Failure Mode> State an Activity is invoked that sends an <Effect> Signal.

When an Operation throws an exception(<Failure Mode>) the <Effect>s that is/are allocated to the <Failure Mode> are observed.

Next Steps and Future Work

- **Continue to expand the SE & FM Modeling Methodology in support of Hardware/Software integration and software assurance cases**
  - Refine methodology and toolset
  - Expand modeling method to support Fault Recovery and FDIR (Fault Detection /Isolation/ Recovery)
  - Work on generating software assurance cases

- **Tailoring of the modeling methodology and plugins for specific NASA project**
  - Apply to multiple NASA projects
  - Support ARTEMIS (Lunar Gateway)

Conclusions

- **Model Based System Engineering**
  - Value added through product generation from a single source
  - Supporting end to end communication
  - Allows to maintain traceability from initial stakeholders expectations to the realized product
- **Tools & Methods are critical to cost effective model creation**
  - Accelerate model development
  - Provides capability to import, maintain, and stay current with model updates
  - Capture system design knowledge that can be shared among all the stakeholders
- **Allows early involvement of stakeholders (ex: Safety Mission Assurance)**
- **Supports risk informed decision making**
  - From an overall functionality perspective
  - Architecture Trades
  - Reliability Trades

➢ **Instead of being reactive, evaluating a system to determine if the system is safe, we used MBSE to be pro-active in design for safety mission assurance**