Association Française d'Ingénierie Système

AFFILIATE

SESE Tour 2018 – Toulouse May 22

Optimal function modelling with SysML

Authors: Regis Casteran, Xavier Dorel, Raphaël Faudou, David Gouyon, Frederic Risy

Presented by Xavier Dorel (Schneider-Electric)
And Raphaël Faudou (Samares-Engineering)

AGENDA

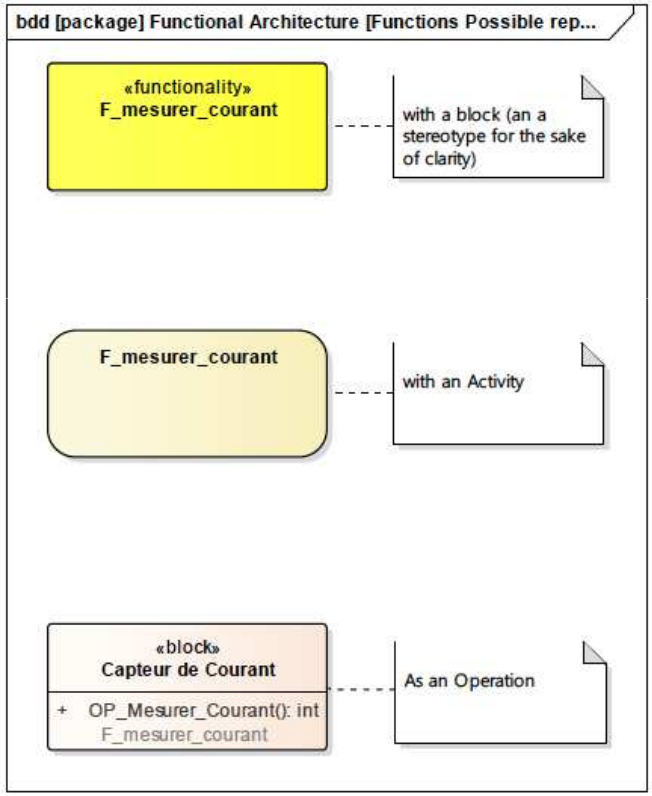
- Motivations
- Key concepts
- Common agreements on methodology
- Engineering activities related to functions
- Sample case for illustration

Who are we?

- Regis Casteran – Assystem Technologies
- Xavier Dorel – Schneider-Electric
- Raphael Faudou – Samares Engineering
- David Gouyon – Université de Lorraine
- Frederic Risy – Airbus DS

Motivations

- **Why we need to address that topic**
  1. In many approaches of Systems Engineering we need to manipulate a “Function” concept, but in SysML, this concept does not natively exist. However, there are several candidate concepts that can fit
  2. We need to get the whole picture to be able to choose the right one for our entity
  3. We are looking for « optimal » approach = best compromise between different notations in order to get something efficient, consistent and executable according with our objectives



- **Different possible representations**
  - **Block, Activity, Block operation**

Key concepts – elements of vocabulary (1)

Function

Transformation of inputs to outputs, by means of some mechanisms, and subject to certain controls, that is identified by a function name.

IEEE Std 1320.1-1998 (R2004) IEEE Standard for Functional Modeling Language — Syntax and Semantics for IDEF0.2.1.53

Note: function may exchange information/energy with system environment or not. Function may be defined at top level of system or at lower level

Key concepts – elements of vocabulary (2)

- **Function definition versus function realization**

  Function definition / specification *aims at defining its inputs/outputs and the transformation rules of its inputs to its outputs, with conditions to the modes and states of the system.*

  Function realization/design *aims at designing a breakdown into lower-level functions such that this breakdown conforms to function definition / specification. Conformance means here that function inputs and outputs are connected to lower level functions inputs and outputs.*

  Note*: there can be several function realizations/designs for one function definition/specification*

- **Functional interface and functional flow**
  - *For each Function, definition of its interfaces (input or output name, types, range of values) and how they are connected with the interfaces of the system of interest and/or with the other functions of the system*

- **Function realization/behavior with data/energy flow**
  - *Definition of how the Functions interfaces are connected to achieve the behavior of upper level Functions*

- **Function realization/behavior with control flow**
  - *Definition of the execution logic of Functions to realize behavior of upper level function: initial node, precedence constraint, decision, merge, fork, join, final node, final flow*

- **Functional architecture**
  - *Functional Architecture defines what the system of interest shall do, whatever its structure (components).*
  - *It is the result of all the system top level functions breakdown into lower level Functions.*
  - *It represents all the lower levels functions behavior with data/energy flow and control flow.*
- **Function allocation to structure**
  - *Function allocation to structure: define for each Function, which structural component(s) of the system gets the responsibility to provide it. When performed for each function, the result represents the allocation of Functional Architecture to Structural Architecture.*
- **Function simulation**
  - *Ability to simulate the behavior of one function or by extension the system functional architecture (through run of code generated from model or through model interpretation/execution)*

Common agreements on methodology

- **Recursive approach => we want to use same representation for different levels of functions (no disruption in representation with regards to level)**
- **Support of functional trade off: ability to define several options for realization/behavior of a given function**
  - ***Consequence: ability to separate function specification and function realization/behavior***
- **Supports traceability**
  - ***Traceability between functions and their "sub functions"***
  - ***Traceability between functions and their source***
- **Consistent functional architecture by construction**
- **Efficient modelling (easy to teach and follow)**
- **Potentially executable**

Engineering activities related to functions

- A. Identification and definition of top level functions
  - *A1. Identification of top level functions from operational scenarios*
  - *A2. Identification of top level functions from modes and states*
  - *A3. Definition of top level functions and functional flows*
- B. Breakdown of functions
  - *B1. breakdown of top level functions into lower level functions*
  - *B2. Definition of lower level functions*
  - *B3. Identification of functional flows between lower level functions*
- C. Functional trade off analysis
  - *C1. Setup different realizations (trades) for a given function definition*
  - *C2. Map functional interfaces of parent function to children functions*
- D. Representation of functional architecture
- E. Allocation of functions to components
  - *D1. Allocation of function definitions to components definitions*
  - *D2: Allocation of functions to component occurrences (deployment)*
- F. Add behavior to a function to support simulation/execution

The subject of this work is too explain how the results of design activities with regards to functions could be formalized using SysML objects (not to explain how we 'find' these functions)

Engineering activities related to functions

Activities	We must represent
A. Identification and definition of external functions	
A1. Identification of top level functions from operational scenarios	A1: The top level functions and their links to operational scenarios
A2. Identification of top level functions from modes and states	A2: The links between top level functions and modes and states
A3. Definition of top level functions	A3 : The specification of top level functions (documentation, diagrams, …)
B. Breakdown of functions	
B1. breakdown of top level functions into lower level functions	B1: Identification of lower level functions from top level functions and hierarchy relationship between them
B2. Definition of lower level functions	B2: The specification of internal (documentation, diagrams, …)
B3. Identification of functional flows between lower level functions	B4: The flows between lower level functions (connection and specification)

Engineering activities related to functions

Activities	We must represent
C. Arrangement of functional flows from upper to lower level – Focus on trade-offs	Concepts to present notion of realization and impacts on global model
C1. Setup different realizations (trades) for a given function definition	C1: Different breakdowns for a given function (external or internal) including the flows
C2. Map functional interfaces of parent function to children functions	C2: An easy to use vision of the flows between functions
D. Representation of functional architecture	D: The resulting functional architecture including all internal functions and all the flows between internal functions and with the system interfaces
E. Allocation of functions to components	
E1. Allocation of function definitions to components definitions	E1: Allocation relationships between function definitions and component definitions
E2: Allocation of functions to component occurrences (deployment)	E2: Allocation relationships between function occurrences and component occurrences
F. Add behavior to a function to support simulation/execution	F: State machine or Activity associated with external function or internal function

A1. Identification of top level functions from operational scenarios

- Use Cases and their associated scenarios help to define how the actors (users, other systems) will get access to the expected features, services the system is supposed to offer them, and how the system shall combine features to get the customers goals accomplished.

- The analysis of those UC and of the sequence of interactions between the system and the actors allows to identify system top level functions

- Very often Use Cases scenarios are represented by Sequence Diagrams (also called “SD”)
  - *Input and output messages then represent flows exchanged by top level function. So SD(s) participate to the definition of top level functions*

A2. Identification of top level functions from modes and states

- Modes and states are represented here by a state machine
- Top level functions identified within the (operational) scenarios analysis may be partitioned within the different states of the system and thus should appear as executed when states transitions are triggered or during a state.
- Conversely, in a state machine, what needs to be executed when a transition is triggered, can be identified as a function of the system

A3. Definition of top level functions

- **Top level functions are mainly represented through Block (and part) or Activity (and action)**
  - ***Definition of function through Block or Activity***
  - ***Usage (call) of function through Part property or Action***

- **Block is a structural element that can represent the function definition**
  - ***Interfaces using "ports"***
  - ***Transformation rules (requirements) can be completed using***
    - ***IBD, State machine diagram, Activity diagram***

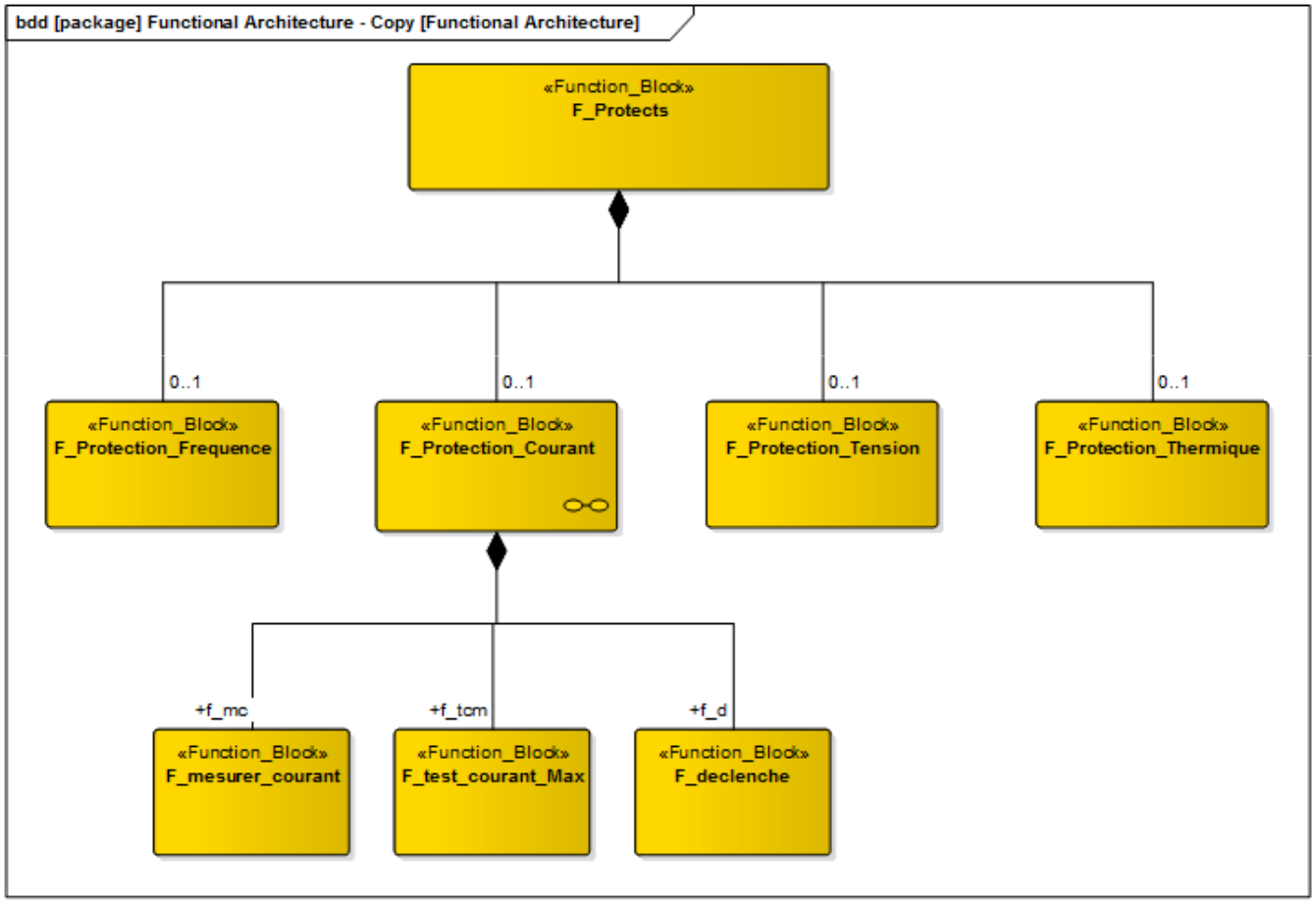
- **Activity is a behavioral element that can also represent the function definition**
  - ***Interfaces using parameters***
  - ***Transformation rules can be completed using control and object flow***

B1.Breakdown of top level functions into lower level functions

- **The most common practice consists in starting from top level function and detail its behavior in terms of lower level functions**
  - *The top level function has knowledge about its lower level functions but lower level functions have no knowledge about top level function*
  - *We often find « composition » relationship between top level function and lower level functions. It means that breakdown is driven by top level function and that lower level function is considered as part of the top level function.*
  - *Sometimes we find "aggregation" relationship between top level function and lower level functions. Difference with composition is that lower level function can be considered in different top level functions (reuse)*
  - *Note: block Definition Diagram can fit for both composition and aggregation*
- **Another approach consists in defining breakdown outside of top level function definition to support reuse with different breakdowns**
  - *We can use inheritance mechanism to get several different breakdowns of same function definition (see trade offs later for more details)*
  - *Breakdown can also be managed as "Assign" stereotype of "MARTE" standard profile. It relates both top level function and lower level functions.*

B1.Breakdown of top level function into lower level functions

Example

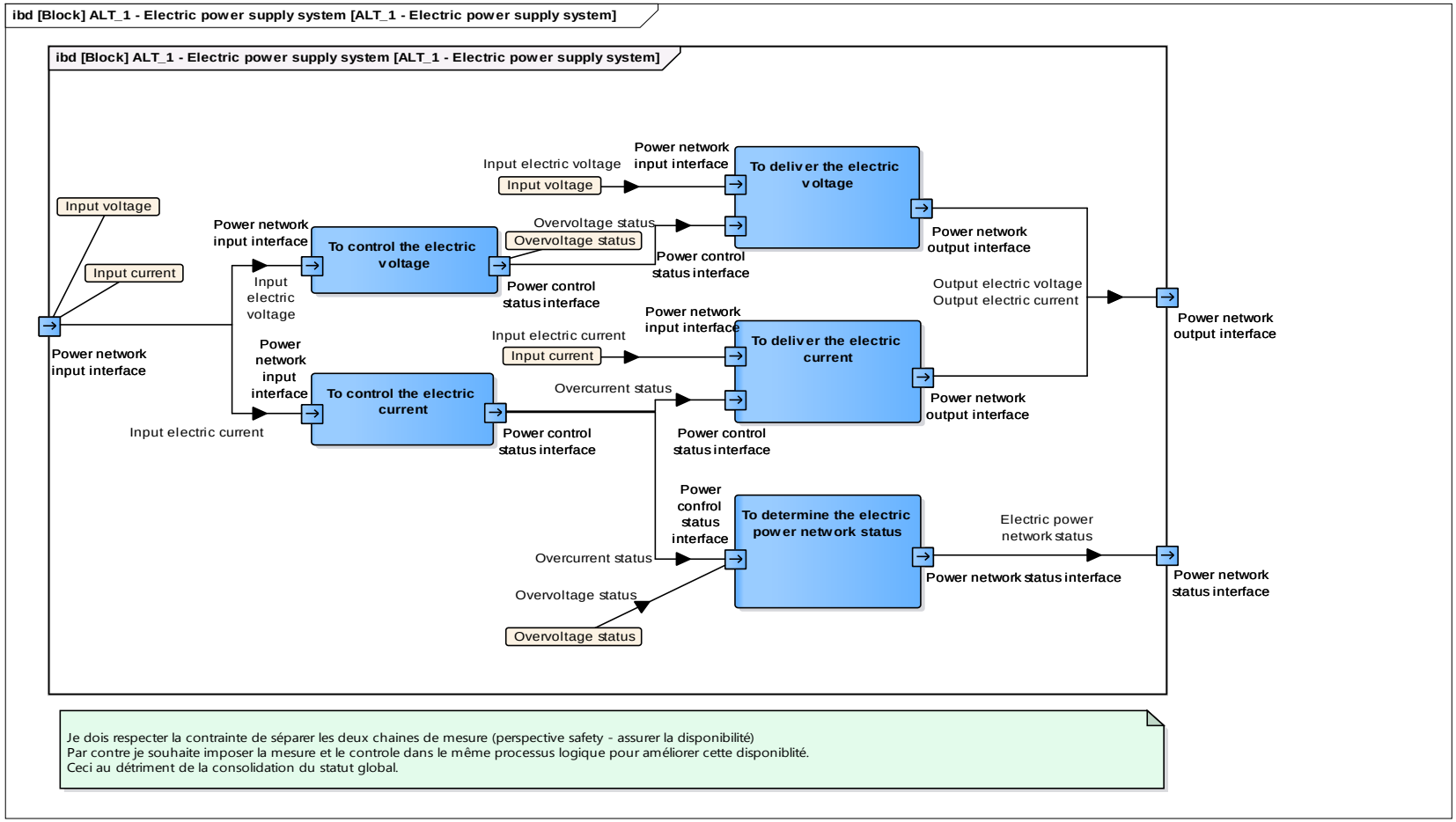


B2.Definition of lower level functions

- **Approach is recursive: we can continue with same representation as with top level functions**
  - ***Either a block if top level function is a block***
  - ***Or an Activity if top level function is an activity***

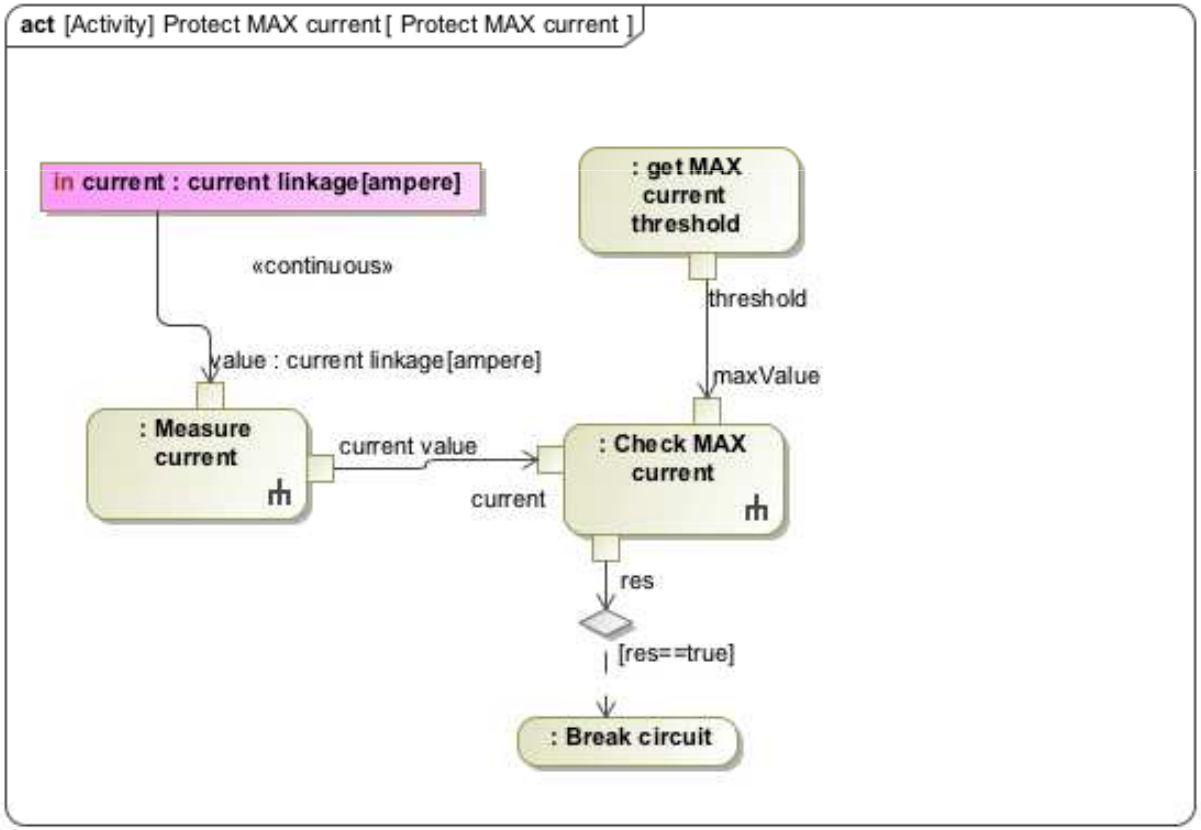
B3. Identify functional flows between functions

- For functions defined as Blocks, flows are represented in an IBD through connectors between ports of parts



B3. Identify functional flows between functions

- For functions defined as Activities, internal flows are represented in an Activity Diagram through object flow between pins of actions (actions can be later mapped to activities => call Behavior Actions)

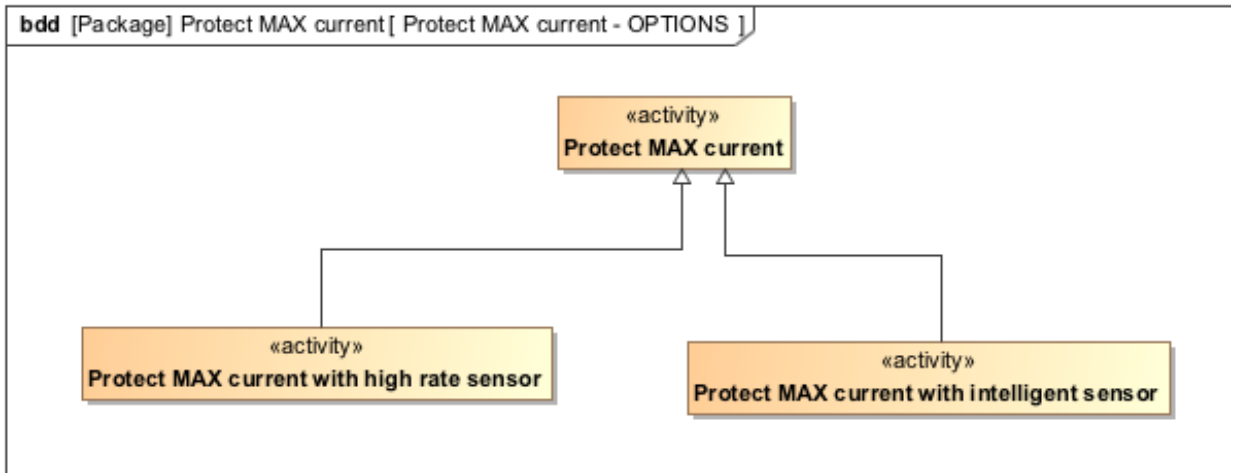


C1: Setup different realizations (trades) for a given function definition

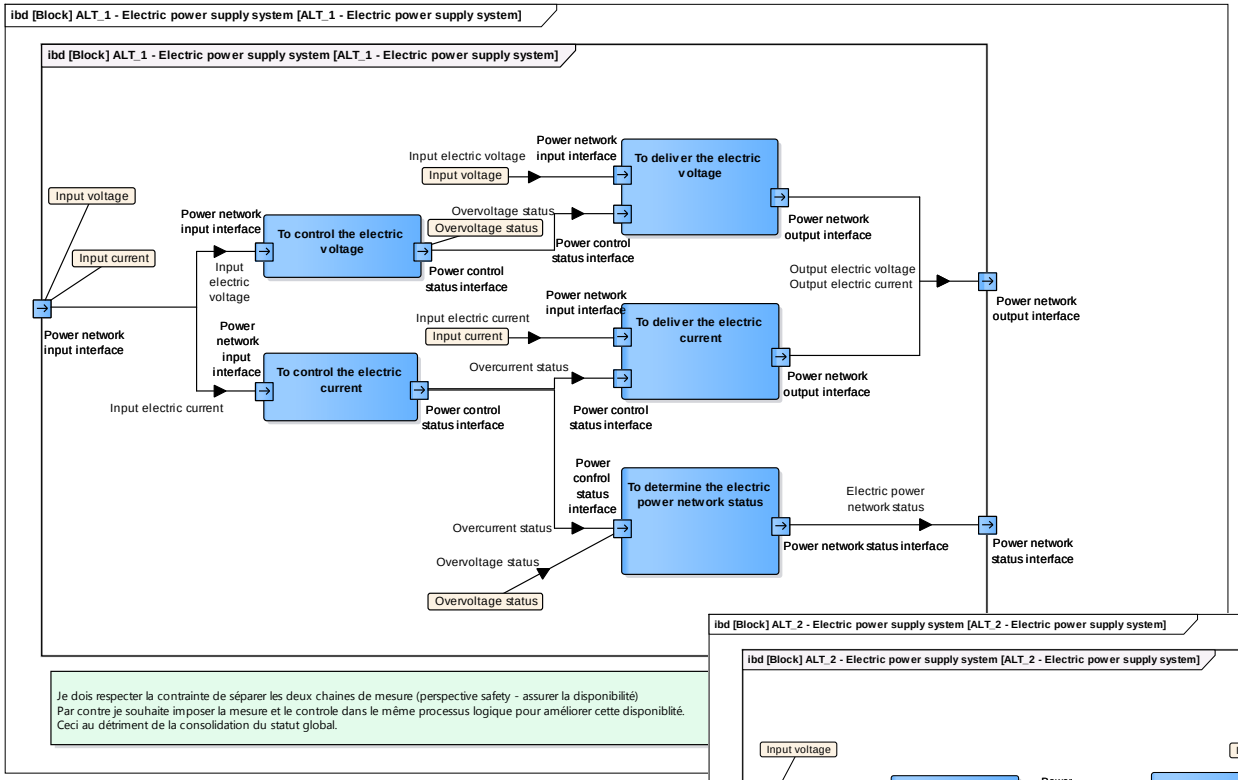
- **Recall: important to provide separation of concerns between definition and realization because there can be several realizations**

- **Key idea = inheritance to represent different options for realization**
  - ***Block inheritance or Activity inheritance ➔ use of a BDD***
  - ***Interest: keep function definition interface (by inheritance)***

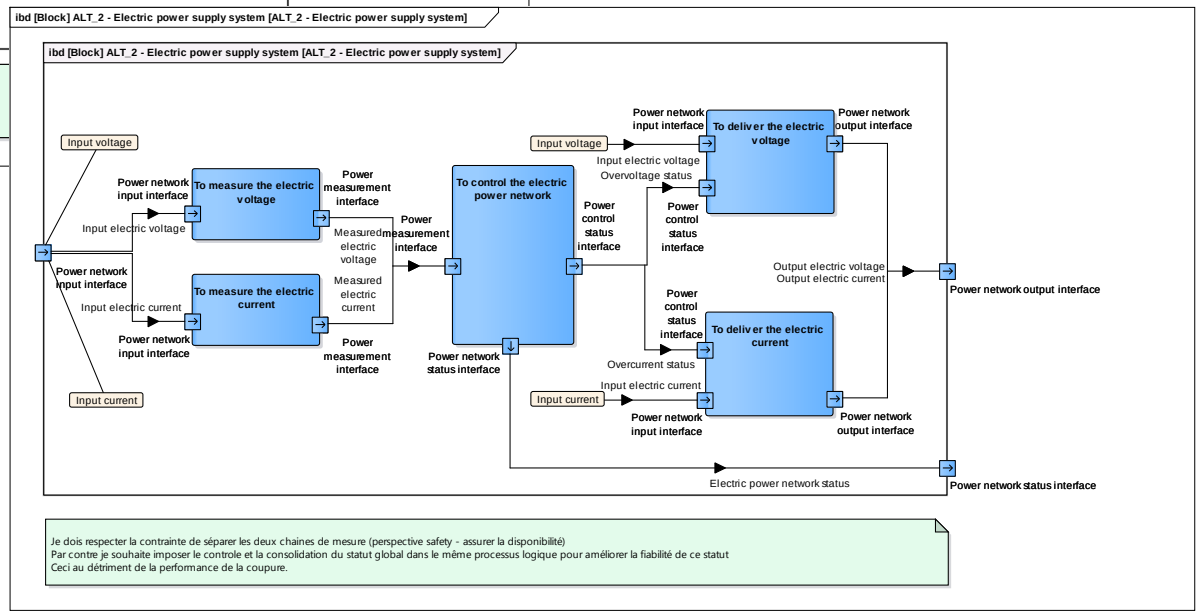
- **Example of 2 options of functions represented as activities**



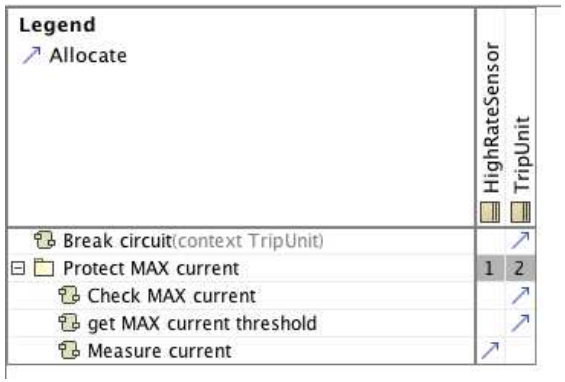
C1: 2 realizations for same function definition (different component allocations)

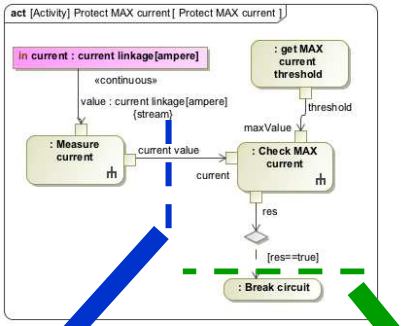


With 2 IBDs



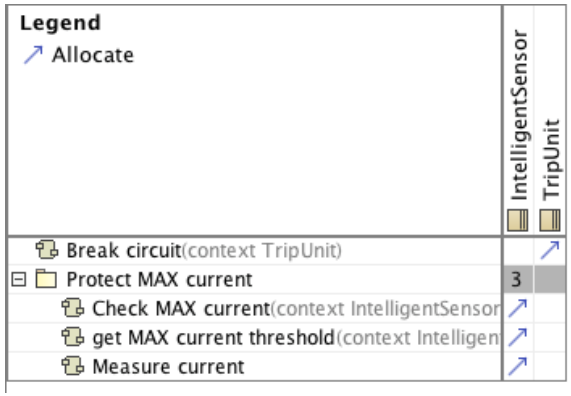
C1: 2 realizations for same function definition (different component allocations)

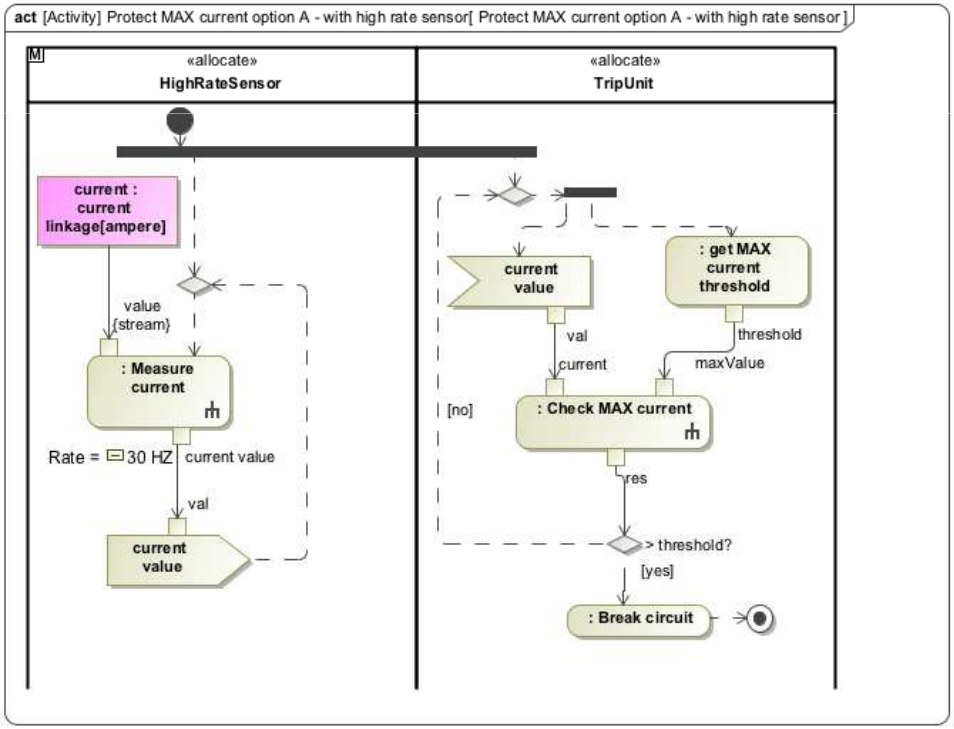


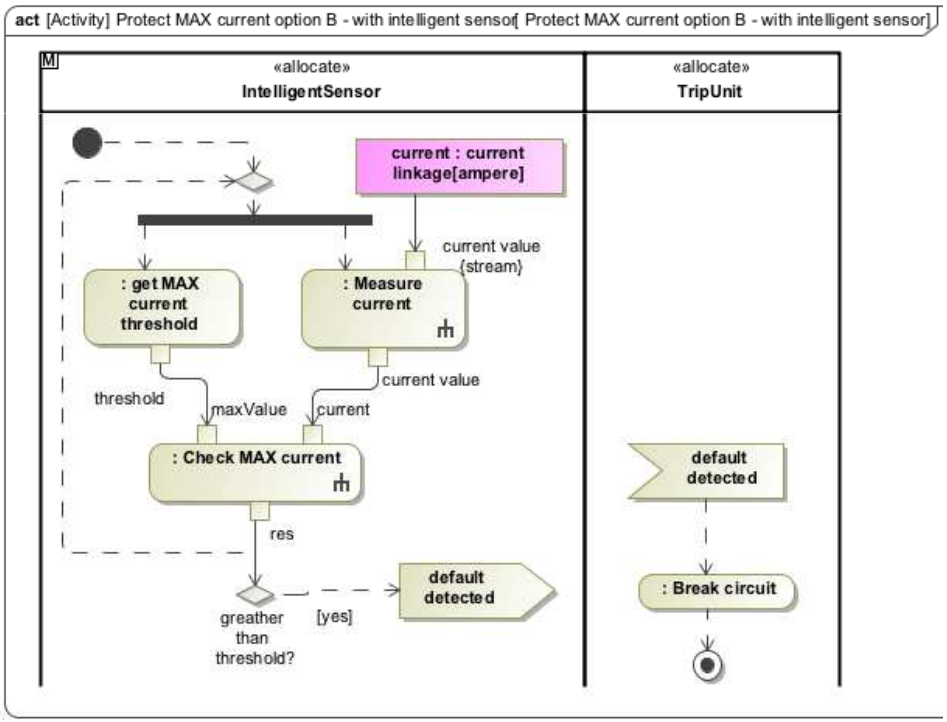


A

B







With 2 Activity Diagrams

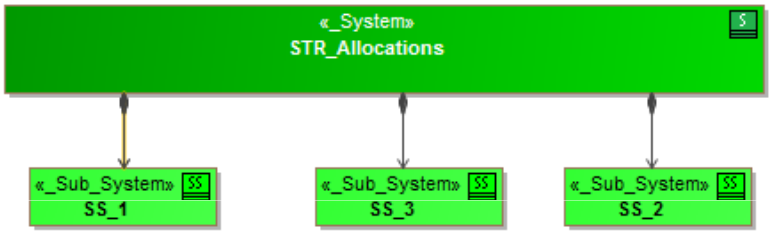
D. Representation of functional architecture

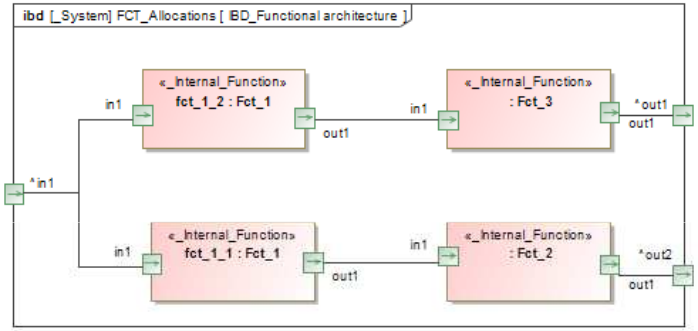
- **Points in common**
  - ***Same "property" mechanism for structural refinement***
    - ***A function as "Block" is refined into "part properties" that have another block as definition***
    - ***A function as "Activity" is refined into "actions" that may have another Activity as definition***

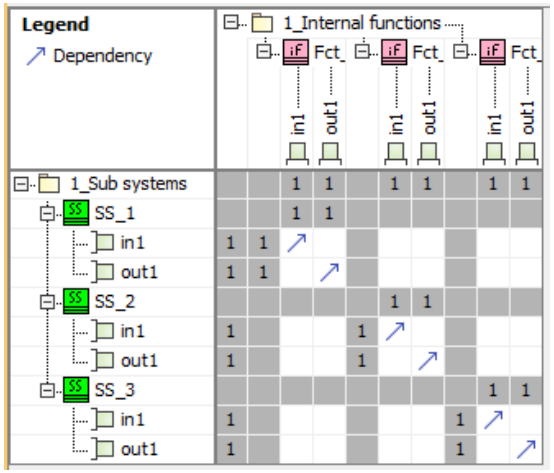
- **Variants**
  - ***Functional flows are expressed with connectors between ports of parts in IBD for functions defined as Blocks***
  - ***Functional flows are expressed with Object Flow between pins of actions in AD for function defined as Activities***
    - ***Tool shall provide synchronization mechanism between action pins and activity parameters – Else such approach becomes really tedious***

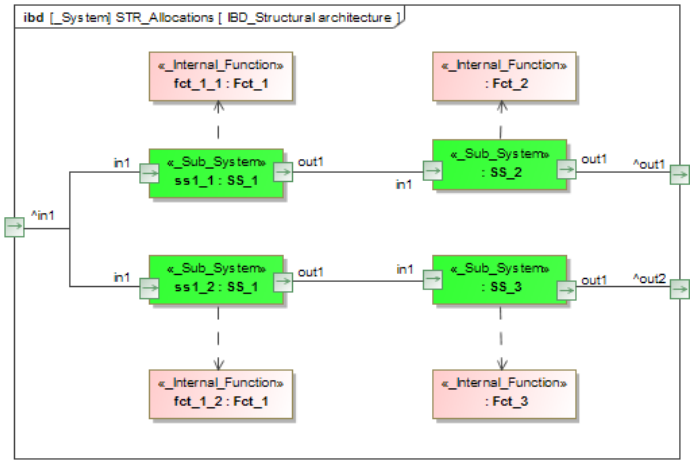
E. Allocation of functions to components

- ***Allocation of definitions versus allocation of occurrences***
  - ***Can allocate function definitions to component definitions***
  - ***Can allocate function occurrences to component occurrences***









F. Add Behavior to a function to support simulation

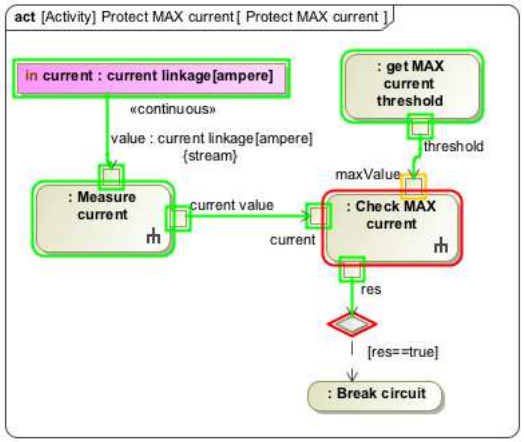
- **Function defined as block only cannot be simulated**
  - ***Function must provide behavior to be simulated***

- **Function definition completed by behavior as state machine or activity can be simulated under some conditions**
  - ***For state machine: there shall be initial node, transitions between nodes and events that trigger transitions shall be defined in the model***
  - ***For activity: activity should conform to some rules and it depends of the simulation engine used:***
    - ***fUML language (standard)***

    for use with Cameo Systems Modeler
    - ***UML action language completed by some***

    specific concepts (especially for guards)

    for IBM Rhapsody, PTC integrity and Sparx EA



Association Française d'Ingénierie Système

AFIS

ENAC

INCOSE
AFFILIATE

Thank you for your attention

xavier.dorel@schneider-electric.com

Schneider Electric

raphael.faudou@samares-engineering.com

SAMARES ENGINEERING
Accelerate Systems Design