# Minimum MBSE

Takeshi Kouno

Sparx Systems Japan

MBSE (Model-Based Systems Engineering) is now familiar to many organizations and companies. When we use Enterprise Architect for MBSE, we usually use SysML for modeling language. There are nine diagrams in SysML, and there are many methodologies about MBSE, so there are many methods to use Enterprise Architect for MBSE.

But usually, any method requires lots of diagrams and lots of elements. It must take much time to create a MBSE model. You may hesitate to adopt MBSE since you have not understood its value and effect yet.

I introduce the simple MBSE method ‘Minimum MBSE’ to do MBSE with Enterprise Architect and SysML. The Minimum MBSE is a simple method by cutting off much usually-important information, so this method is suitable for trial. If you understand the value and effect of MBSE, Enterprise Architect, and SysML, you should choose another approach and try for full-scale MBSE method.

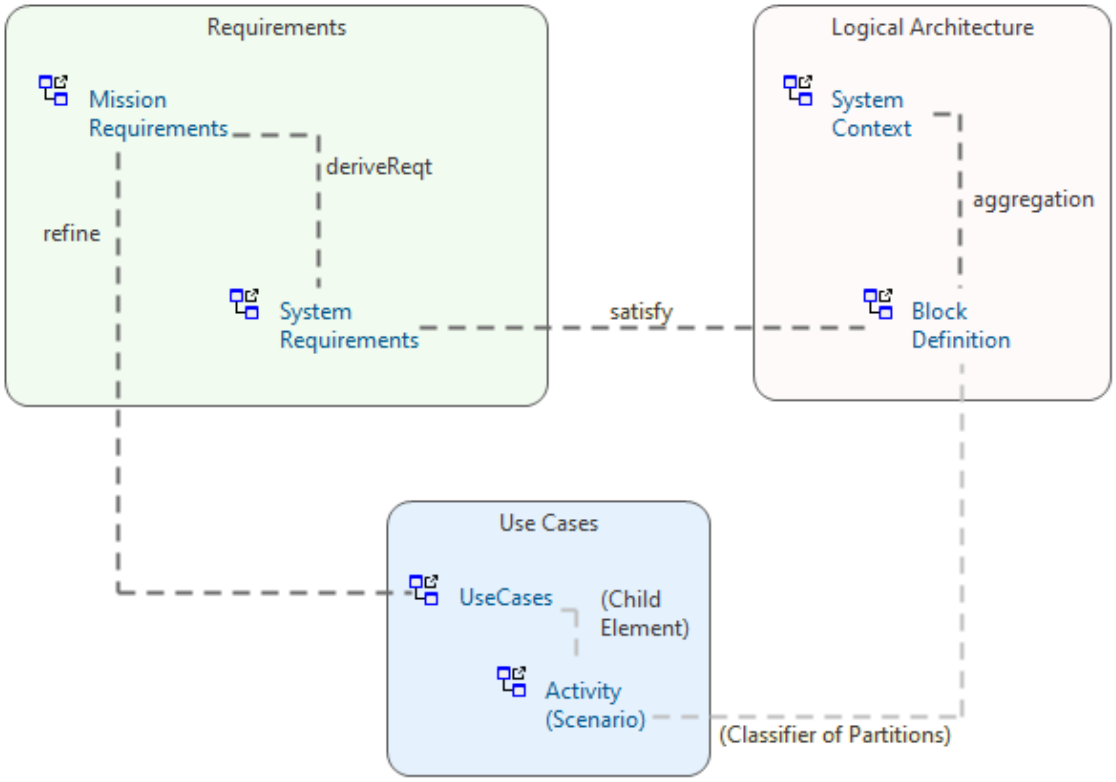
The Minimum MBSE is best for the following situations.

- You already have enough knowledge about the target system and/or domain.
- The target system already exists, and you need to enhance or modify the system.
- The system size is not so large. Only one or a couple of engineers will create a model.

So, the Minimum MBSE is NOT suitable for the following situations.

- You do not know enough about the target system and/or domain.
- The target system is new.
- The target system is large and/or complex.

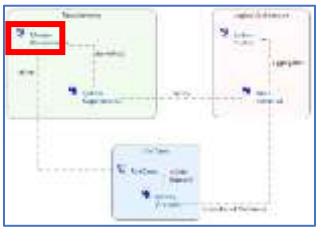
The following figure shows artifacts and their relationships by this method.



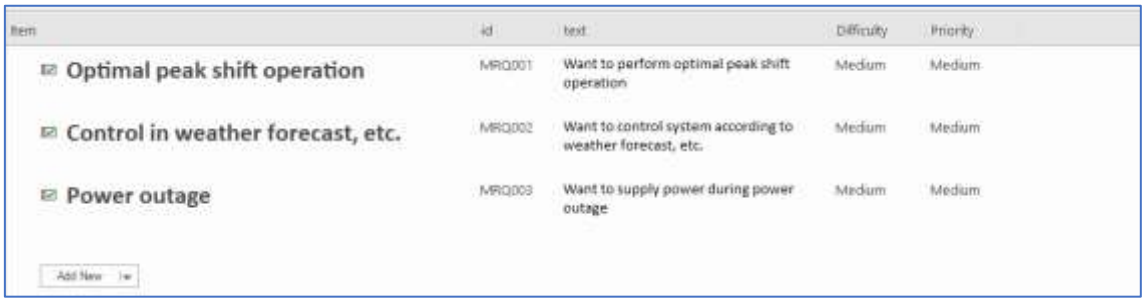

I introduce the summary of each step in this document. Please take care; we cannot complete each step in a single work. We need to return and repeat previous steps to create a complete and consistent model.

I use the Traceability Suite Add-In to display some traceability as a matrix and map in this article. For detail about the Add-In, visit https://www.sparxsystems.jp/en/trace/.

## 1. Identify and Clarify Mission Requirements



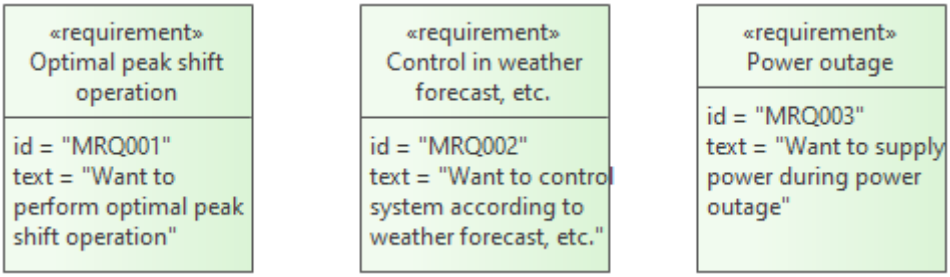
At first, we identify and clarify mission requirements to consider what system we will create. Stakeholders of the target system usually define these requirements. We use the SysML Requirement diagram and define requirements as SysML Requirement. Then, add ID and description as text. Enterprise Architect's Specification Manager is helpful in this step.



| Item | id | text | Difficulty | Priority |
|---|---|---|---|---|
| Optimal peak shift operation | MRQ001 | Want to perform optimal peak shift operation | Medium | Medium |
| Control in weather forecast, etc. | MRQ002 | Want to control system according to weather forecast, etc. | Medium | Medium |
| Power outage | MRQ003 | Want to supply power during power outage | Medium | Medium |

Add New

These requirements are displayed in the Requirement diagram.

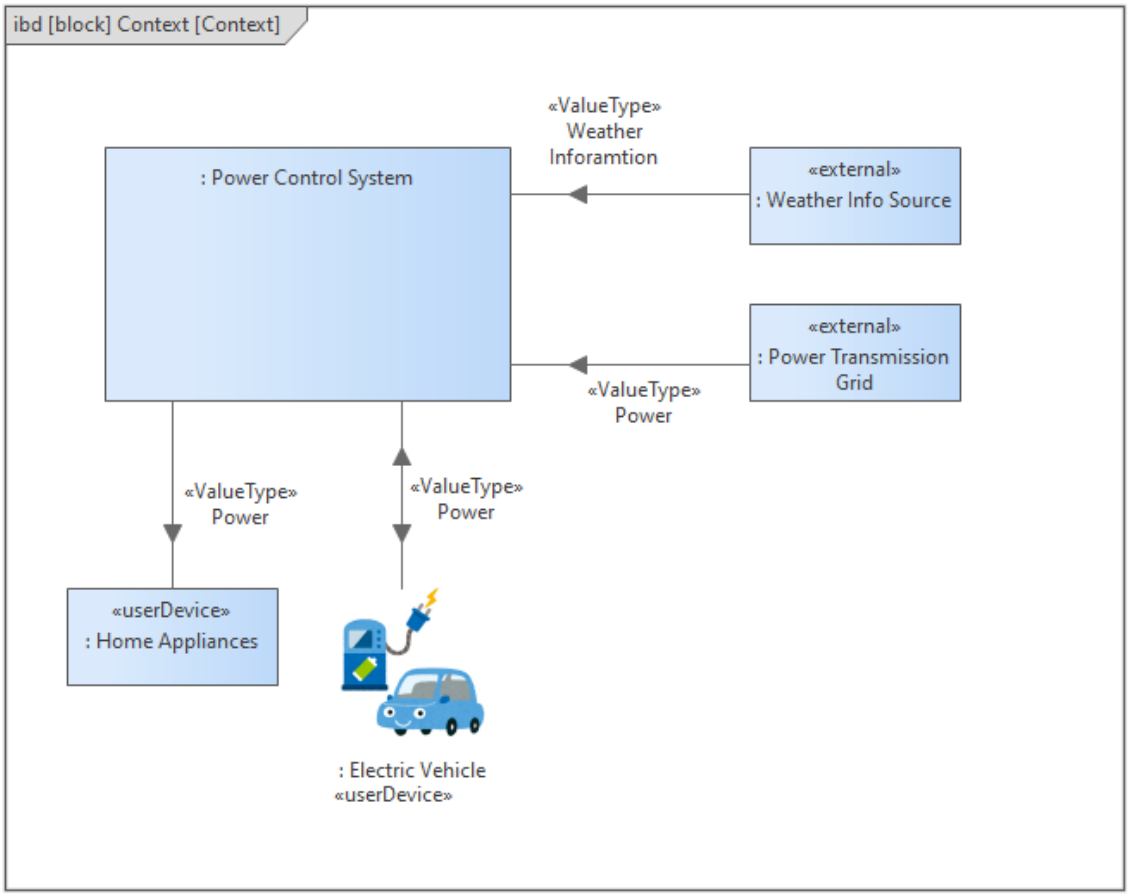



These mission requirements contain both functional and non-functional requirements. (In this example model, only functional requirements are defined.)

## 2. Clarify Context of the System

The second step is creating a Context diagram to clarify the scope of the system. By creating a context diagram, we can understand what we will create and what we will not create. Except for the target system (in this example, the Power Control System), all other systems, devices, services, etc. (‘external elements’ in this document) are out of the scope. We do not need to consider them in detail.

In this Minimum MBSE, usually, we omit various kinds of data to keep it simple. But in the context diagram, it is an excellent option to define data between external elements and the target system because input/output data help to understand the context. When we try with other full-scale MBSE methods, the data must be defined and applied to various model areas.

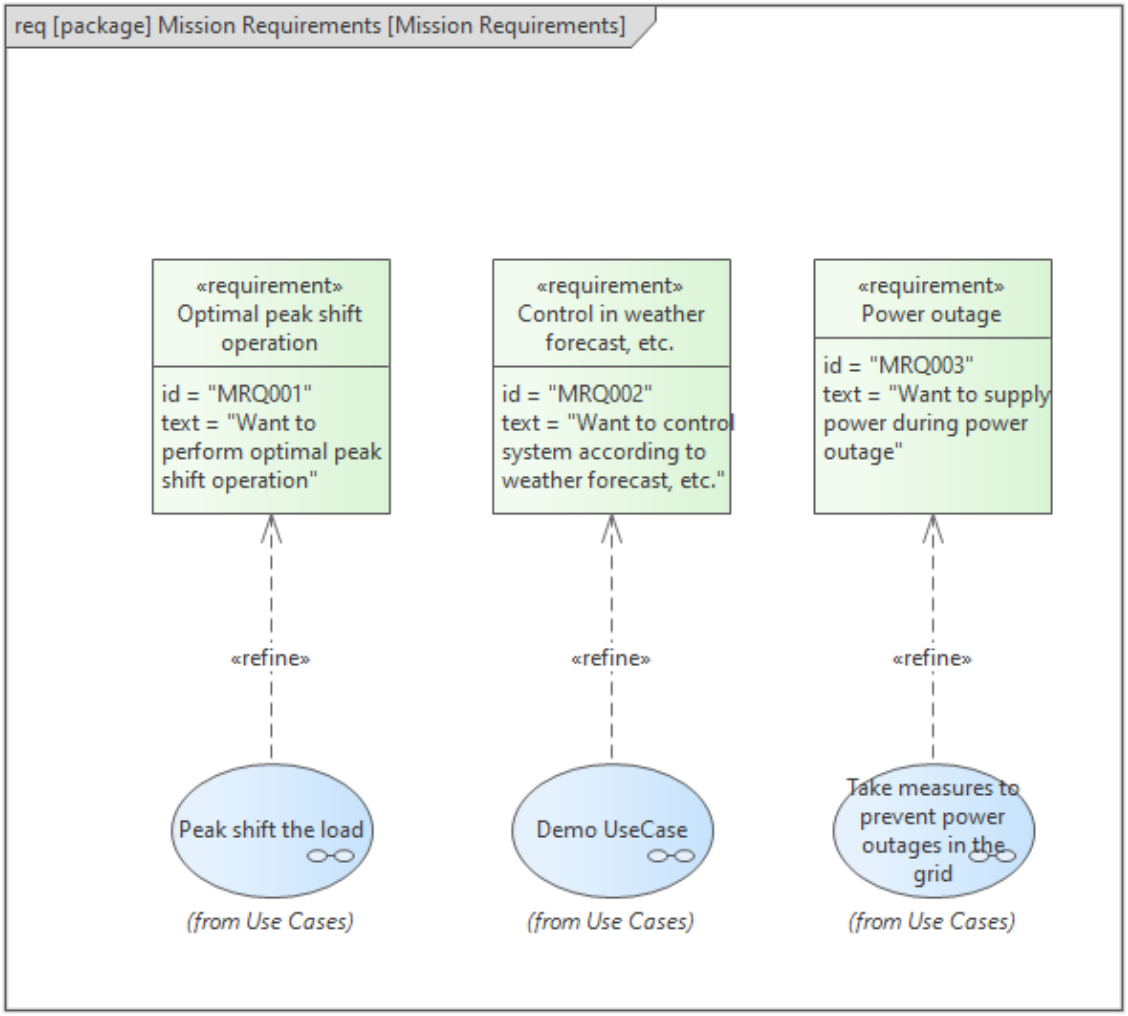



It is an excellent option to use images to objects in this diagram, mainly external elements.

## 3. Capture Use Cases

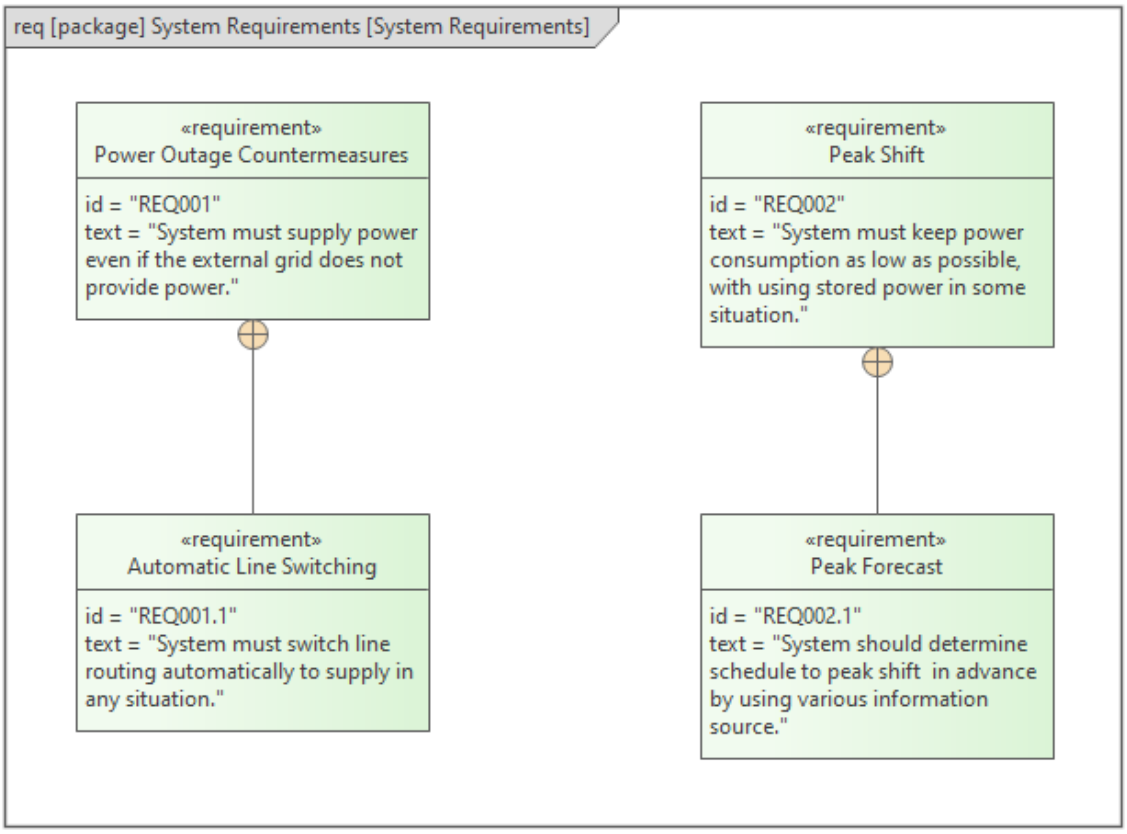
The third step is capturing use cases invoked by stakeholders to clarify how users and external elements use the target system. Usually, users of the target systems invoke use cases, but external elements might invoke some use cases.

Each use case refines one or more mission requirements. We need to connect between them by the SysML Refine connector. Using the matrix is easy to connect them.




## 4. Identify and Clarify System Requirements

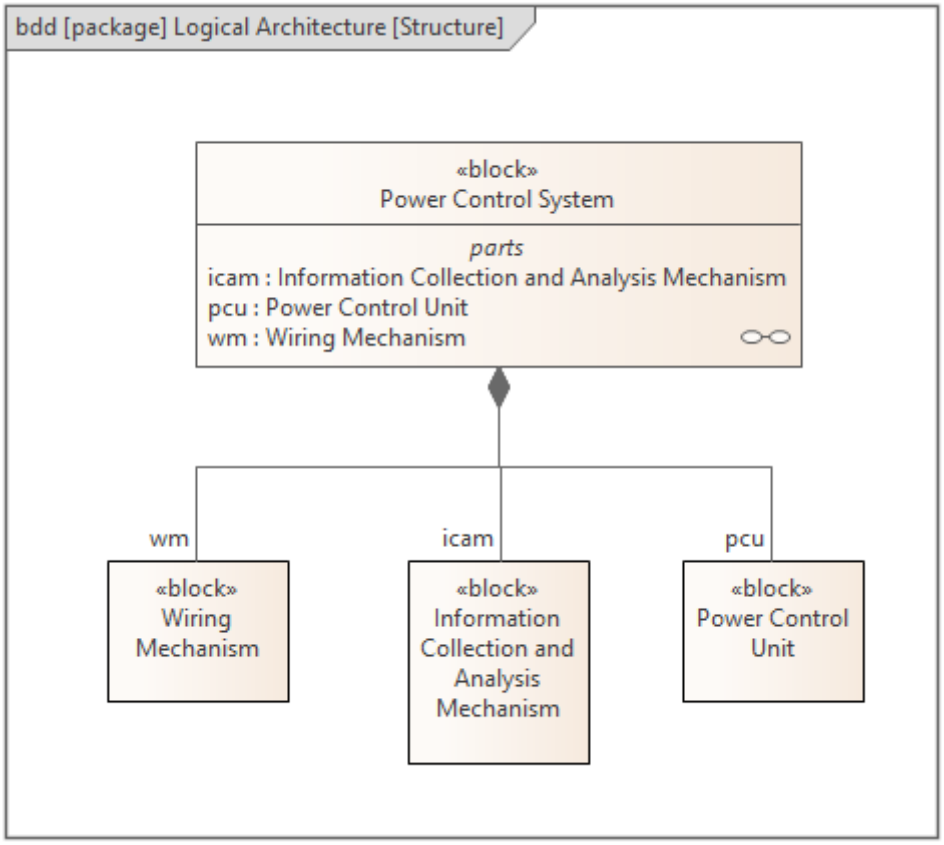
Next, we identify and clarify system requirements to determine the target system's feature. System requirements are defined by systems engineers from the viewpoint of systems engineers. System requirements meet and satisfy all mission requirements. Mission requirements are like '**Users want** to do something by using the system,' but system requirements are like '**The system must (or should) do** something.' System requirements contain non-functional requirements and constraints. We can decompose system requirements.




SysML DeriveReqt connects mission Requirements and System Requirements. Using the matrix is easy to connect between them.

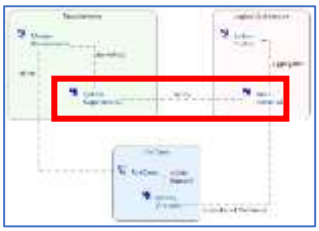
## 5. Define Logical Blocks

In this step, we define logical blocks as SysML Blocks to build a structure of the target system. There is no special way to list all necessary blocks in this method. This is why we need knowledge of the target domain or system.




We can confirm whether we can list all necessary blocks by further steps,

## 6. Confirm Consistency between System Requirements and Blocks



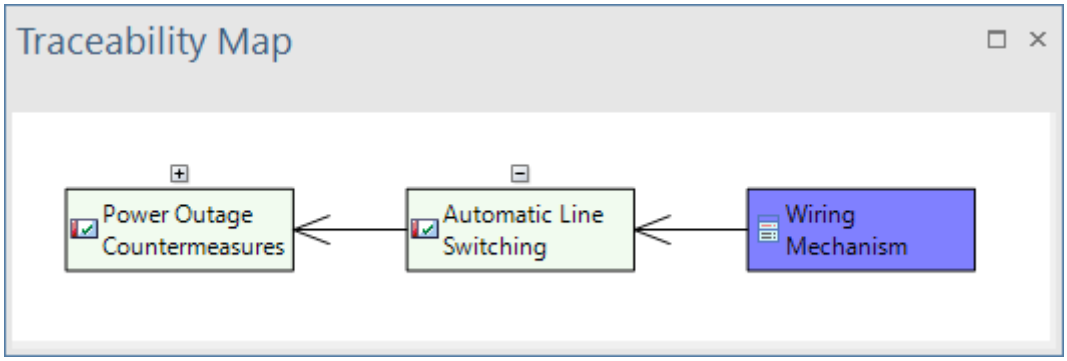
After defining logical blocks, we confirm relationships between system requirements and blocks by the matrix. If a requirement is not satisfied by any block, we need to add one or more blocks to satisfy the requirement. It might have no relationship if a requirement derives from other requirement(s).

It might be no problem that a block does not satisfy any requirement, mainly when we aim to enhance or modify the existing system.

| Source | Logical Architecture | Information Collection and Analysis M | Power Control Unit | Wiring Mechanism |
|---|---|---|---|---|
| System Requirements | 4 | 1 | 2 | 1 |
| Power Outage Countermeasures | 0 | | | |
| Automatic Line Switching | 2 | | ← | ← |
| Peak Shift | 0 | | | |
| Peak Forecast | 2 | ← | ← | |

(Tagged Value ‘id’ sorts requirements in this matrix.)

In this example, one of the relationships is like following in the traceability map.



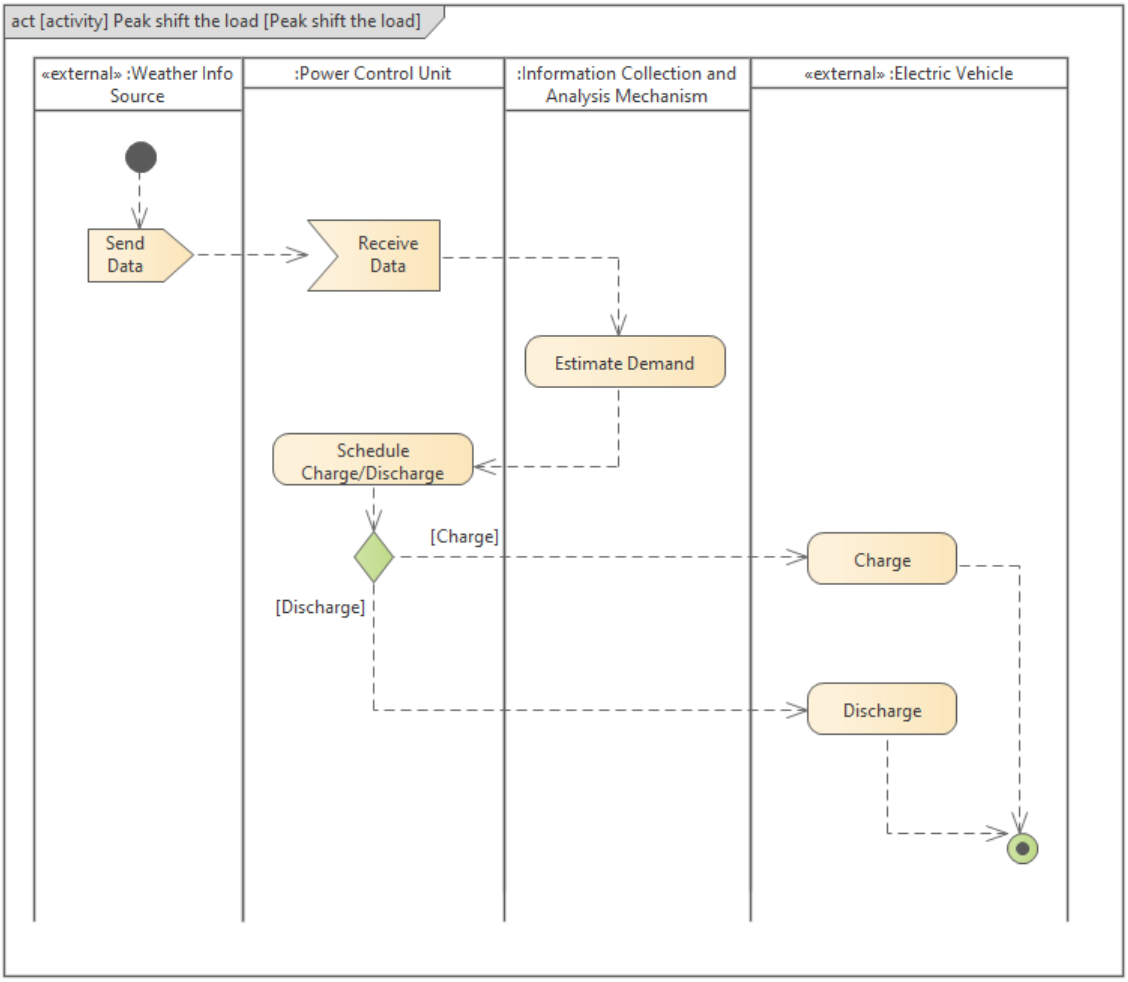

We can confirm the correctness of relationships by seeing a two-hop matrix between the top-level system requirements and logical blocks.

| Source | Logical Architecture | Information Collection and Analysis M | Power Control Unit | Wiring Mechanism |
|---|---|---|---|---|
| System Requirements | 4 | 1 | 2 | 1 |
| Power Outage Countermeasures | 2 | | ✓ | ✓ |
| Automatic Line Switching | 0 | | | |
| Peak Shift | 2 | ✓ | ✓ | |
| Peak Forecast | 0 | | | |

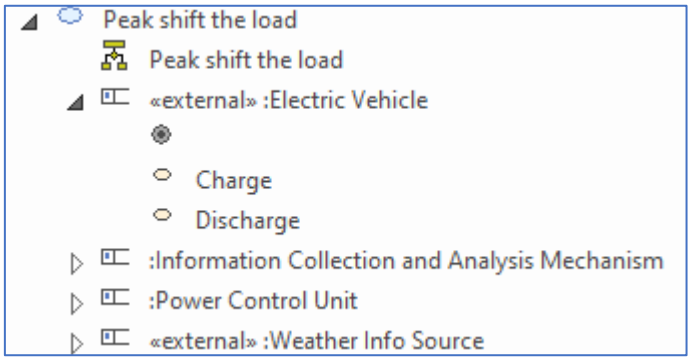
## 7. Create Use Case Scenarios

Then, we create Activity diagrams for use cases to describe scenarios. In each Activity diagram, logical blocks and external elements are placed as Partitions. Then, add actions to describe a scenario of each use case. Usually, we describe flows of both operation (control) and data (object) in Activity diagrams, but in this method, we describe only operational flows to simplify creating a model. We know that this cannot create a complete model, and we might miss some essential features, but we aim to understand the value of using Enterprise Architect and SysML.

There is another problem: it is difficult to define the granularity of actions, in other words, how many actions we use in each diagram. We could describe scenarios in detail using many actions, decisions, and merges. I recommend a policy that we do not place consecutive actions in the same lane. The following action should be on the different partition. In some cases, we need to use two or more successive actions on the same partition to describe the scenario correctly, but we should try to explain the actions as one action. This policy can keep the model simple and avoid describing it in detail.

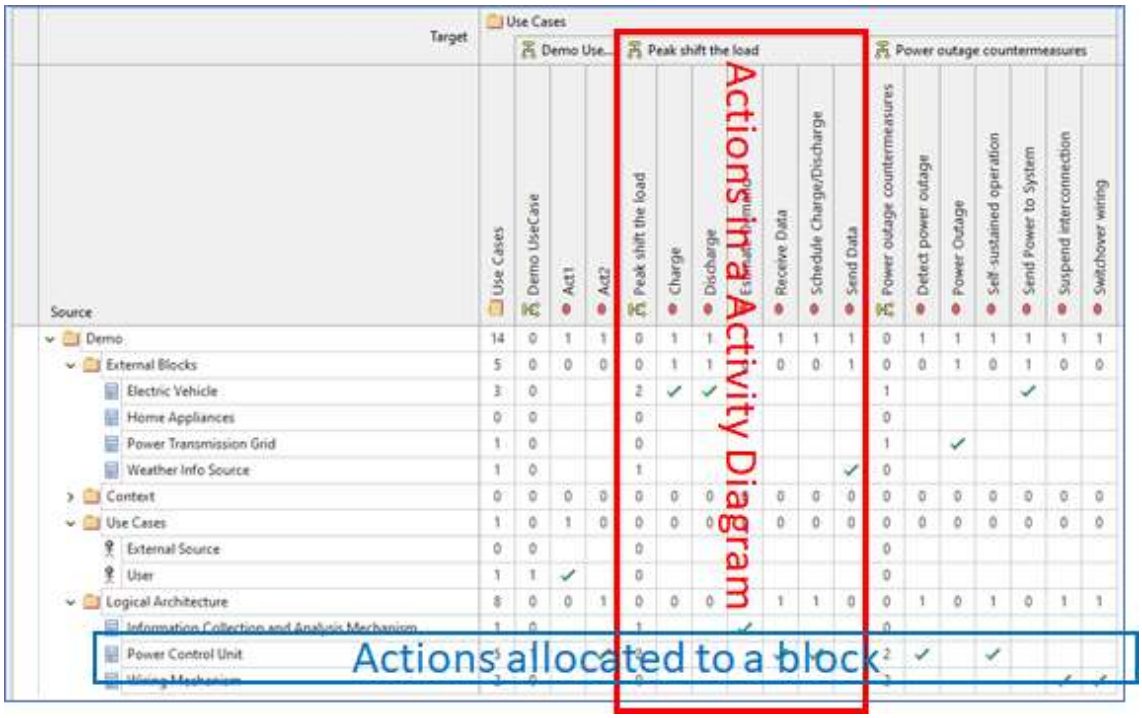



We do not need to add any connector among use cases, activities, partitions, and actions. We can trace parent-child relationships and type-usage relationships by the traceability map. We need to construct the Browser like below.




## 8. Confirm Consistency between Actions and Blocks

After creating all scenarios, we clarify the responsibility of each block by confirming whether we allocate all actions to blocks correctly. Actions on a partition will be features of a related block. By using the matrix, we can confirm relationships between blocks (including external blocks and actors) and actions in all scenarios at a glance.




We need to confirm the following:

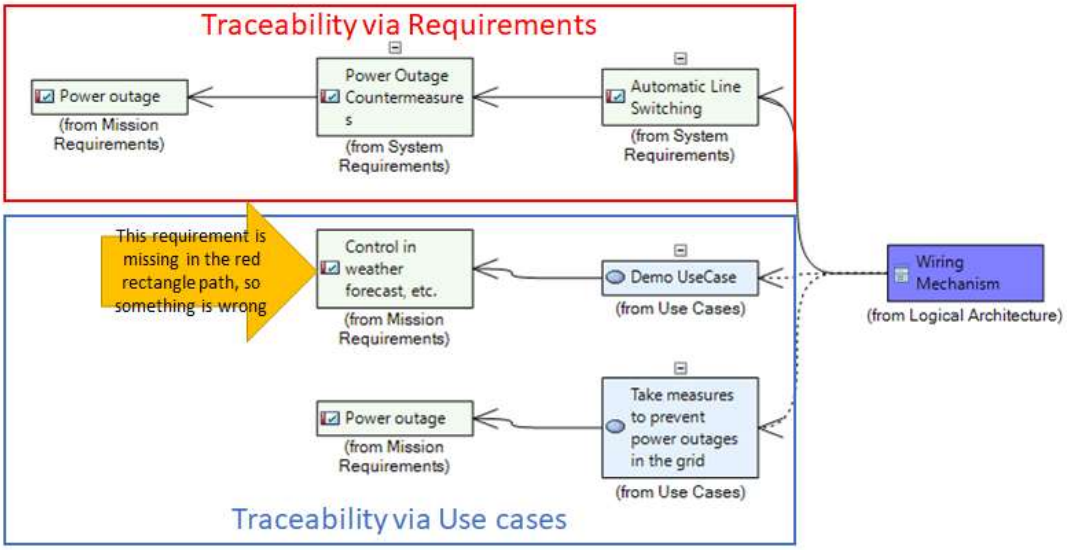
- Identical or similar actions are NOT allocated to different blocks.
- If a block has many actions compared with other blocks, it might be abstract and decomposed into more concrete blocks.
- If a block has no action, we need to check if we miss some use cases (e.g., miss some mission requirements)

We need to modify or update existing blocks, use cases, requirements, and actions to be a better model. Return to each step and confirm by the matrix again.

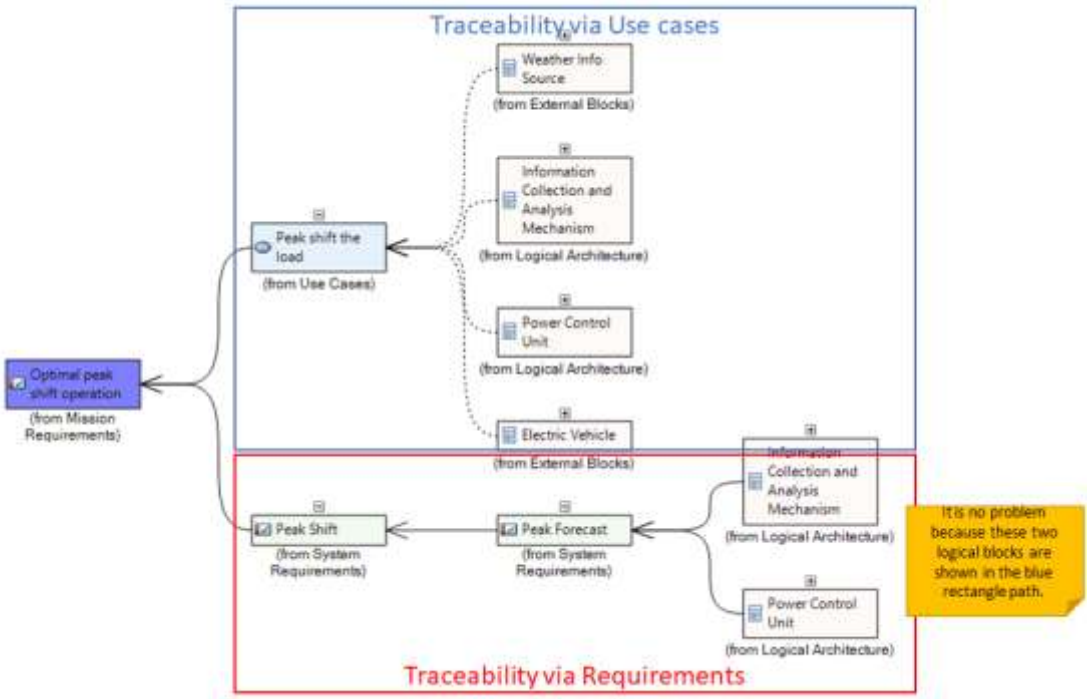
## 9. Find Inconsistency, Omission, and Missing

Finally, we confirm model consistency and find omission and missing in the model to create a consistent model. Please check the first figure in this article again. The traceability is looped, so we can confirm consistency by tracing these two

paths. For example, the following map confirms relationships by checking two paths. In this case, the starting point is a logical block.



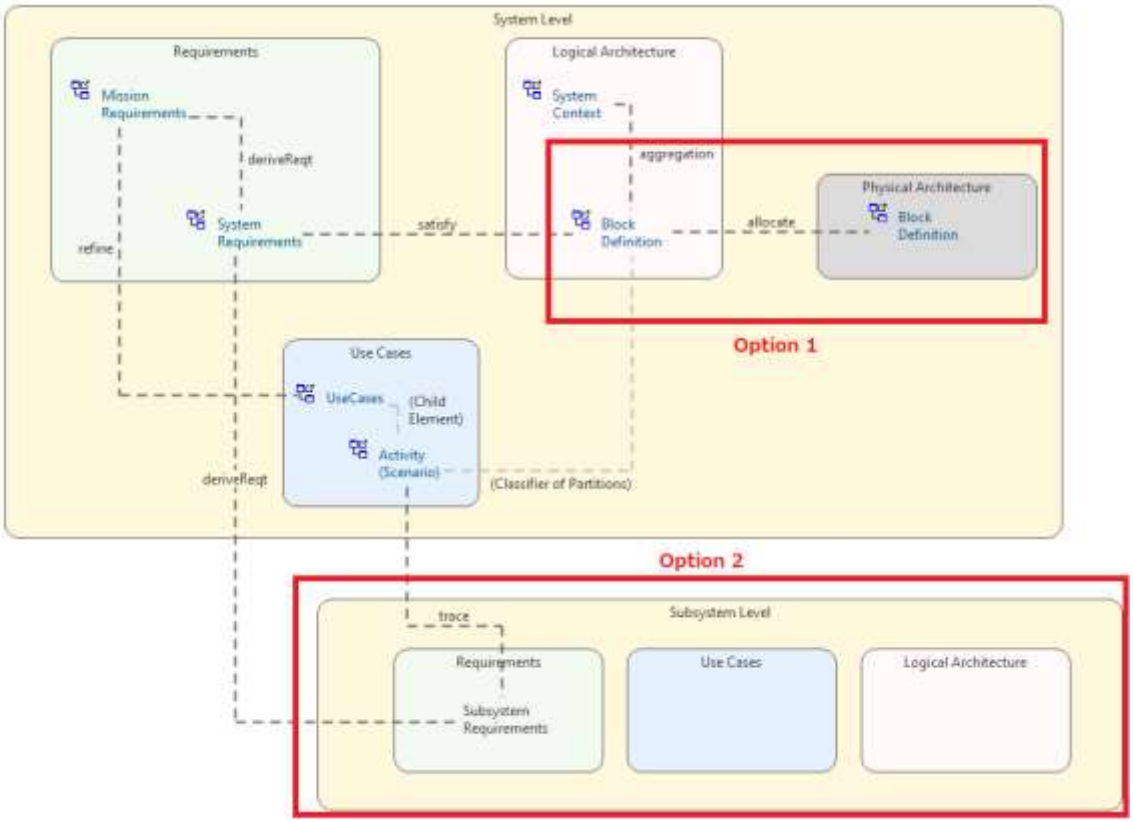

In the same manner, we can check completeness from a mission requirement.



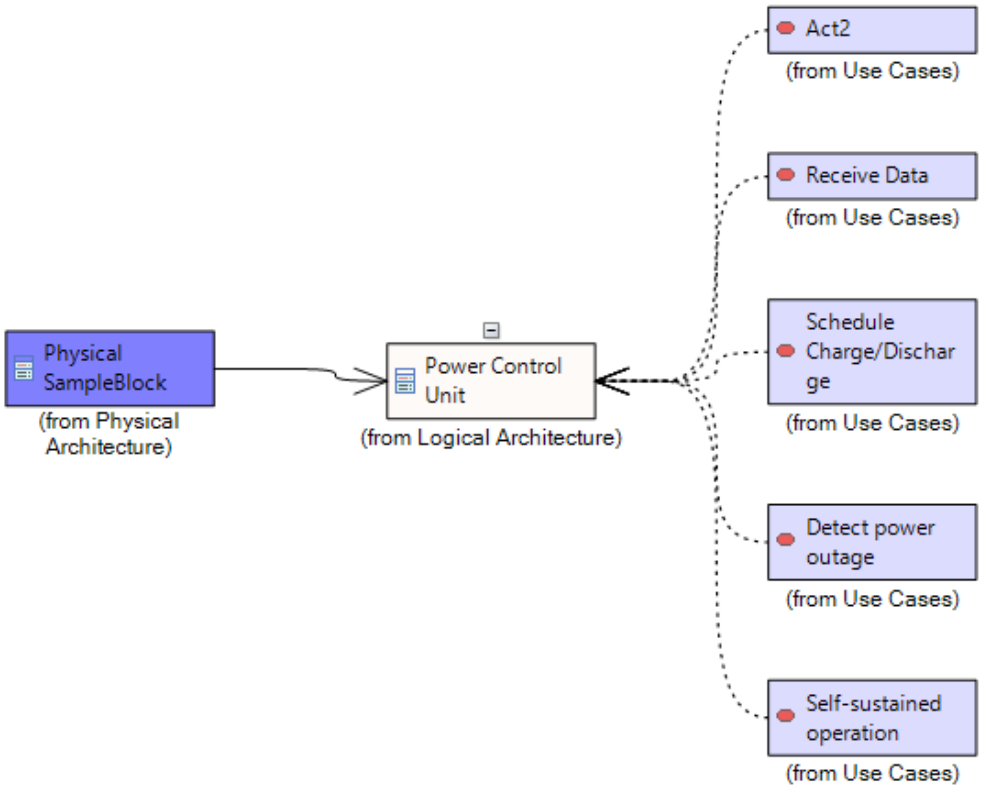
## Summary, and What's Next?

We can identify 'all' mission and system requirements by these steps. And we can find 'all' allocated features (responsibilities) for each logical block. We can confirm 'all'-ness by traceability. This Minimum MBSE method is straightforward, and it omits many essential things to create a detailed model for system design and development. But I think it is still helpful for some purpose and situation.

There are two further options if we want to continue.




The first option is defining physical blocks, then allocating logical blocks to physical blocks. While allocating, we might need to consider some requirements and constraints. For example, we might merge two or more logical blocks to a (one) physical block to save cost. We might separate (or duplicate) a logical block into two or more physical blocks to continue working the system even if some physical block is broken. After allocating logical blocks to physical blocks, we can see actions that belong to a physical block below the map by using traceability.




Creating models for subsystems is another option. Each logical block will be a subsystem. Requirements of a subsystem are derived from system requirements. All actions allocated to each block are related to the requirements of a subsystem. Here, there are two traceability paths so we can confirm whether we can identify all subsystems' requirements.

I know this method is not perfect, but this is an excellent option to try MBSE with Enterprise Architect and SysML quickly.

If you have any questions, ideas, or anything else, please send an email to me at tkouno@sparxsystems.jp.