Application of Requirements Analysis Methodology and Automatic Code Generation Using SysML and L&L with Assurance

Japan Aerospace Exploration Agency (JAXA)

Safety and Mission Assurance Department

Michihide NITTA, Hiroki UMEDA, Masafumi KATAHIRA

Outline

- This talk covers the following ;

  1 Requirement analysis using MBSE method focusing on constraints to prevent missing requirements

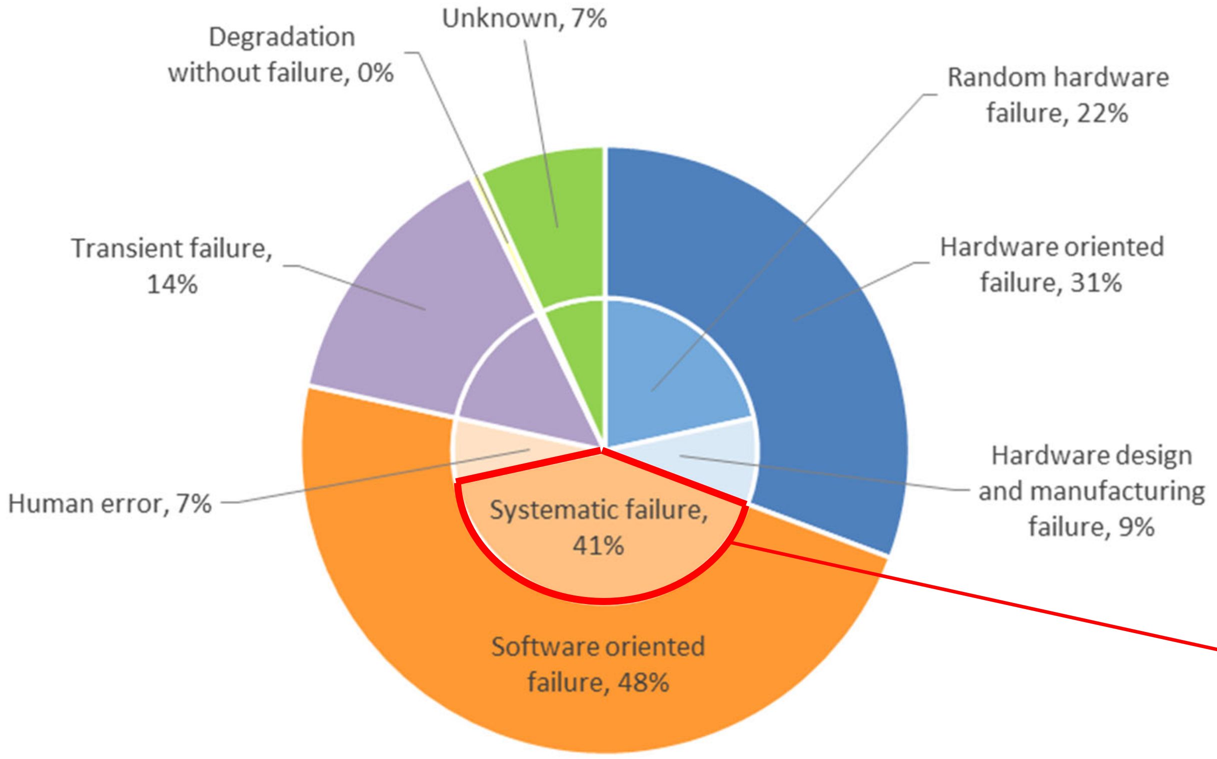
  2 Introduction of JAXA's Automatic Code Generation

1. REQUIREMENT ANALYSIS USING MBSE METHOD FOCUSING ON CONSTRAINTS TO PREVENT MISSING REQUIREMENTS

■Identification of issues (1)

Approximately 40% of on-orbit spacecraft failures are functional design-related failures, which are systematic failures caused by overlooking constraints.

JAXA's Spacecraft orbital failures from 2000 to 2019 (371 cases in total)



<Failure Inclusion Factors / Failure Remaining Factors>

(1) Design input error
(Example: Interpretation of calculation formulas between manufacturers)

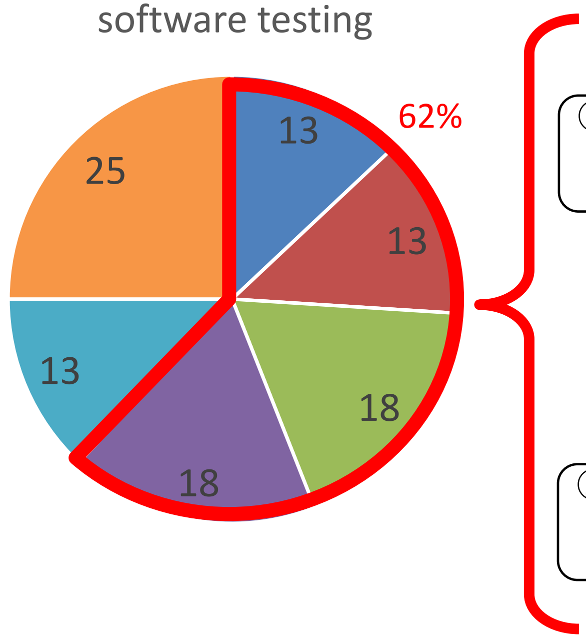
(2) Design error
(Example: contradiction between constraints implicitly placed in various parts of the system)

(3) Omission of verification
(Example: Unverified because the constraints were not clear)

■Identification of issues (2)

Software testing fails due to “blind spots by spatial and temporal distribution“.

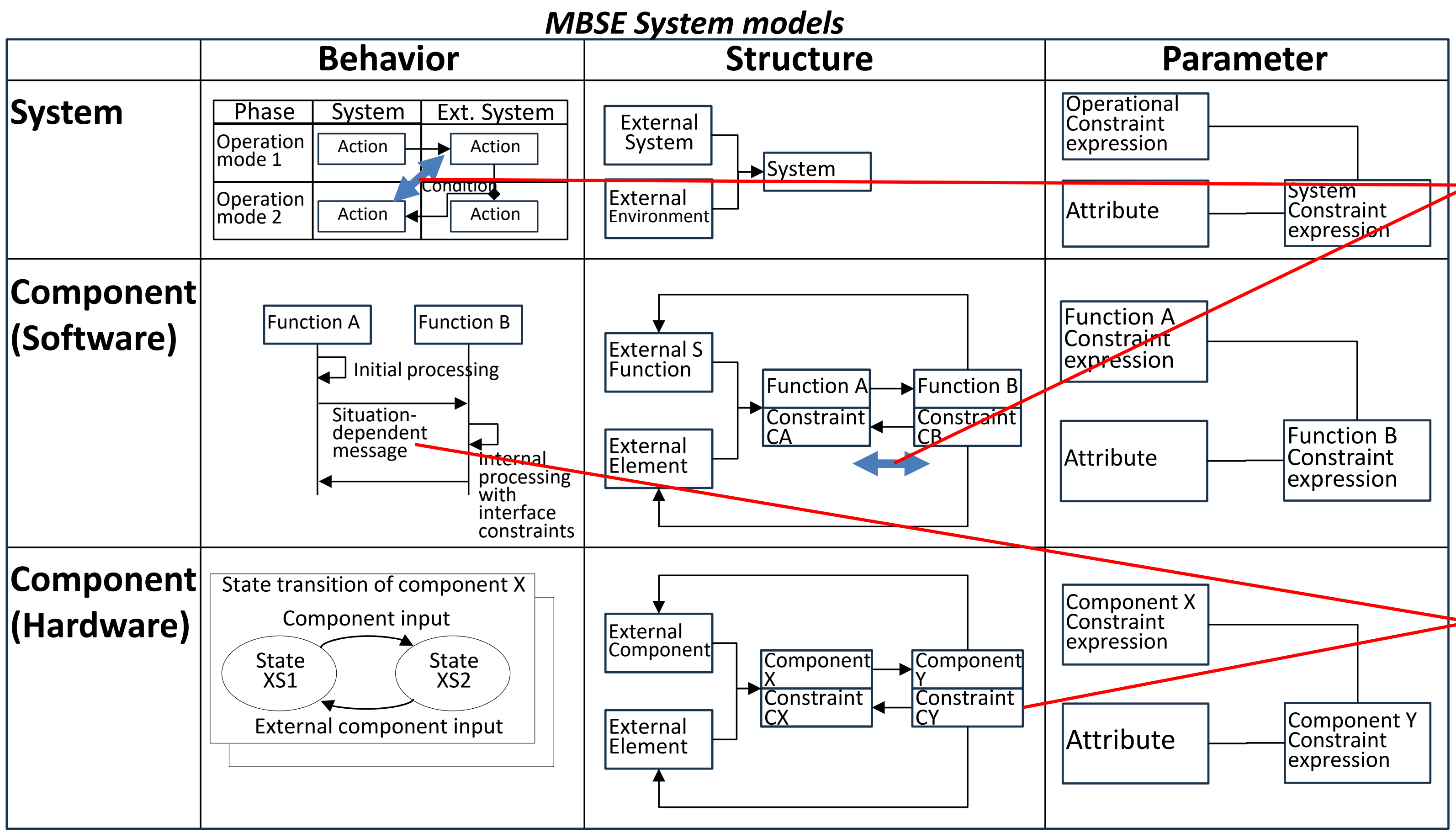
Failure factors after software testing



Classification of causes		Examples of failure factors
①: Spatial distribution of information	🟦 Missing constraint of relationships between upper and lower architectures	・Software does not take hardware constraints into account. ・Component constraints are not taken into account during operation.
	🟥 Large number of transmissions and missing of input/output relationships and processing at the same level	・Leakage of input-output relationships that depend on components. (It is necessary to consider elements with indirect input-output relationships.) ・If counterpart does not consider the constraints of the relevant element. (In a certain state, the operation is not accepted.)
②: Temporal distribution of information	🟩 Missing due to differences in temporal/sequential information	・The sequence between components and the state transitions of the components are not consistent.
	🟪 Time reference point missing due to differences	・Caused by the difference between starting and internal time ・Caused by periodic processing and interrupted processing
③: No distribution	🟦 Multiple internal elements	・Too many elements or parallel actions *Detectable only by direct input/output relationship
	🟧 A single internal element	・Occurs only on single internal elements

■Identification of issues (3)

To prevent overlooking due to temporal and spatial dispersion, it is necessary to guide viewpoints when using MBSE modeling and models that address the issues



- Example of missing constraint of relationships between upper and lower architectures

➢ << System level >>
Switching operation of primary and secondary systems.

➢ << Component level >>
Constraints on the system switching conditions were leaked, because the switching function at the time of switching differs for each component.

- Example of time reference point missing due to differences

➢ << Function level >>
The conditions for sending and receiving commands (reference time criteria) differ depending on the purpose and status of the function.

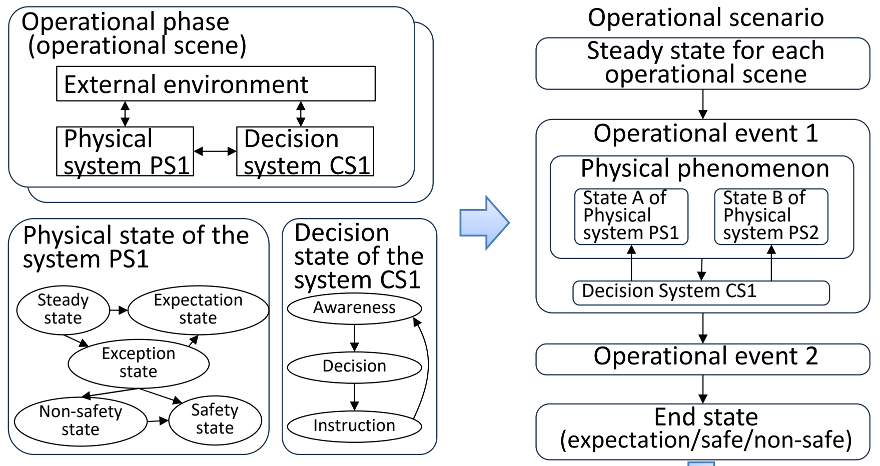
➢ << Hardware level >>
There are constraints on the cycle and update frequency of commands to be read depending on the component.

1.1 System functional design based on operational scenarios using constraint relations

■ Systematizing architectural longitudinal constraints and evaluating system functional design

the process of the method	the perspective of its analysis
Step1: Stakeholders Analysis	Definition the expectation state from Stakeholders
	Definition of Hazard Condition
Step2: Definition of external Systems and external environment	Identification of external systems for each phase of operation
	Extraction of physical actions with external environment and systems
Step3: Operational scenario generation	Analysis of the order of interaction between each system
	System state detailing (expectation, exception, non-safety, etc.)
Step4: Implementation of functional design	Functional input/output conditions by refinement
	Functional input-output conditions from system constraint expressions

Analysis Framework Overview (Output images)



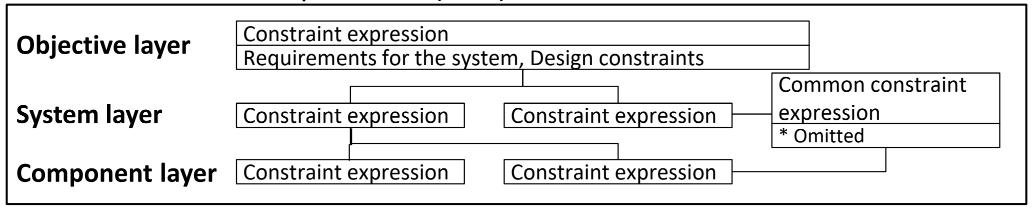
Generation

System function design results

Function			means of implementation	achievement level	non-functional properties
Execution process	Dest.	Execution condition			

Review

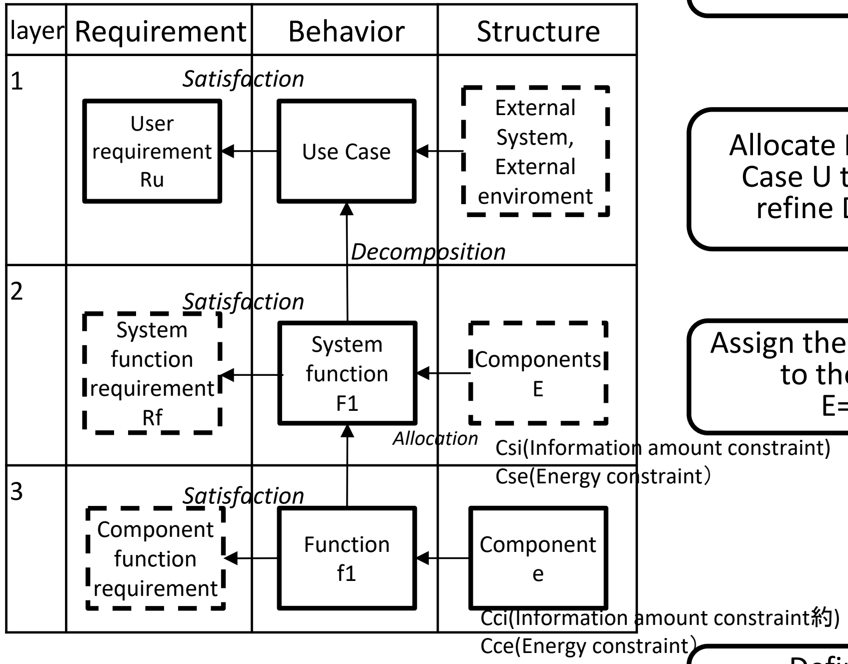
Constraint relational expressions (CRD)



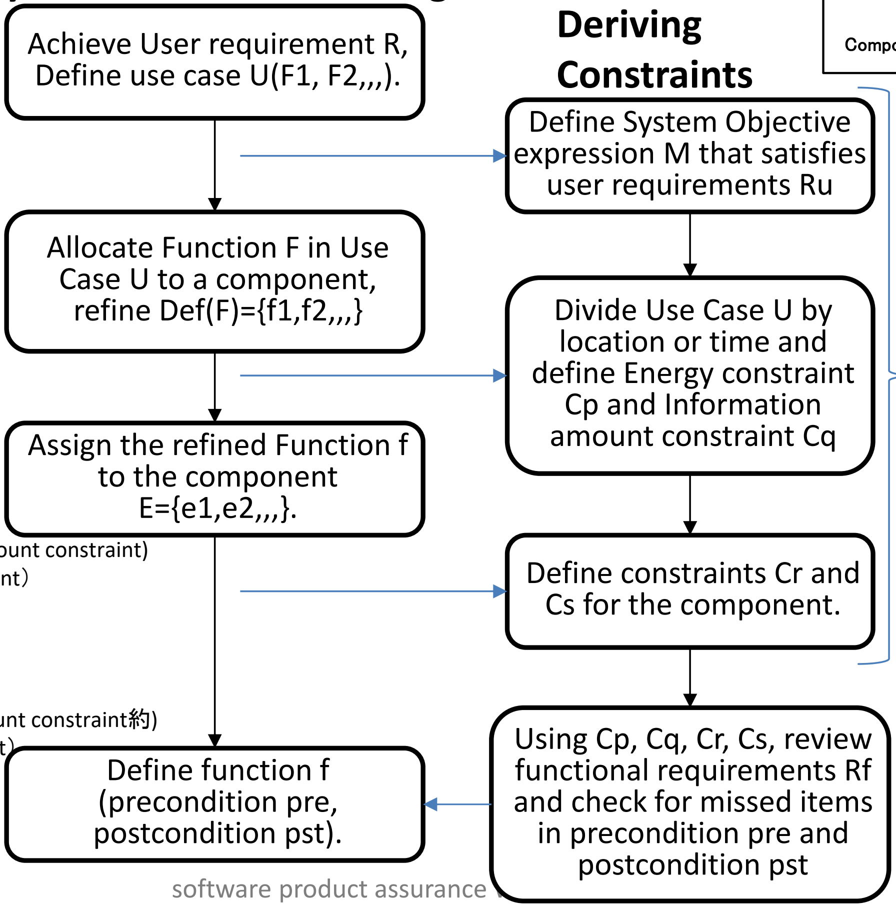
1.1 System functional design based on operational scenarios using constraint relations

- Description rule for CRD (1/2)

System Models（SysML）



System Architectural Design / **Method for Deriving Constraints**


Achieve User requirement R, Define use case U(F1, F2,,,).

Allocate Function F in Use Case U to a component, refine Def(F)={f1,f2,,,}

Assign the refined Function f to the component E={e1,e2,,,}.

Define function f (precondition pre, postcondition pst).

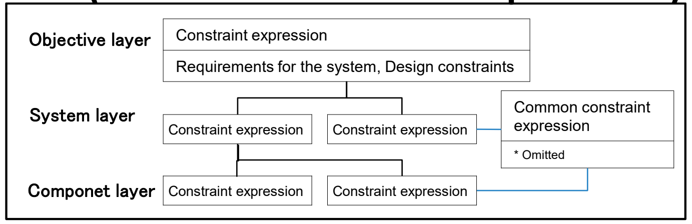
Define System Objective expression M that satisfies user requirements Ru

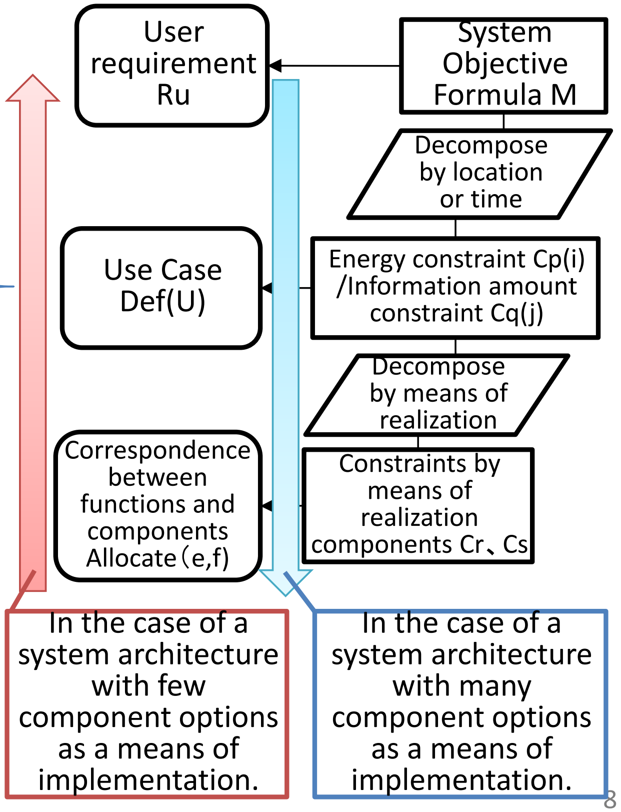
Divide Use Case U by location or time and define Energy constraint Cp and Information amount constraint Cq

Define constraints Cr and Cs for the component.

Using Cp, Cq, Cr, Cs, review functional requirements Rf and check for missed items in precondition pre and postcondition pst

CRD (Constraint relational expressions)



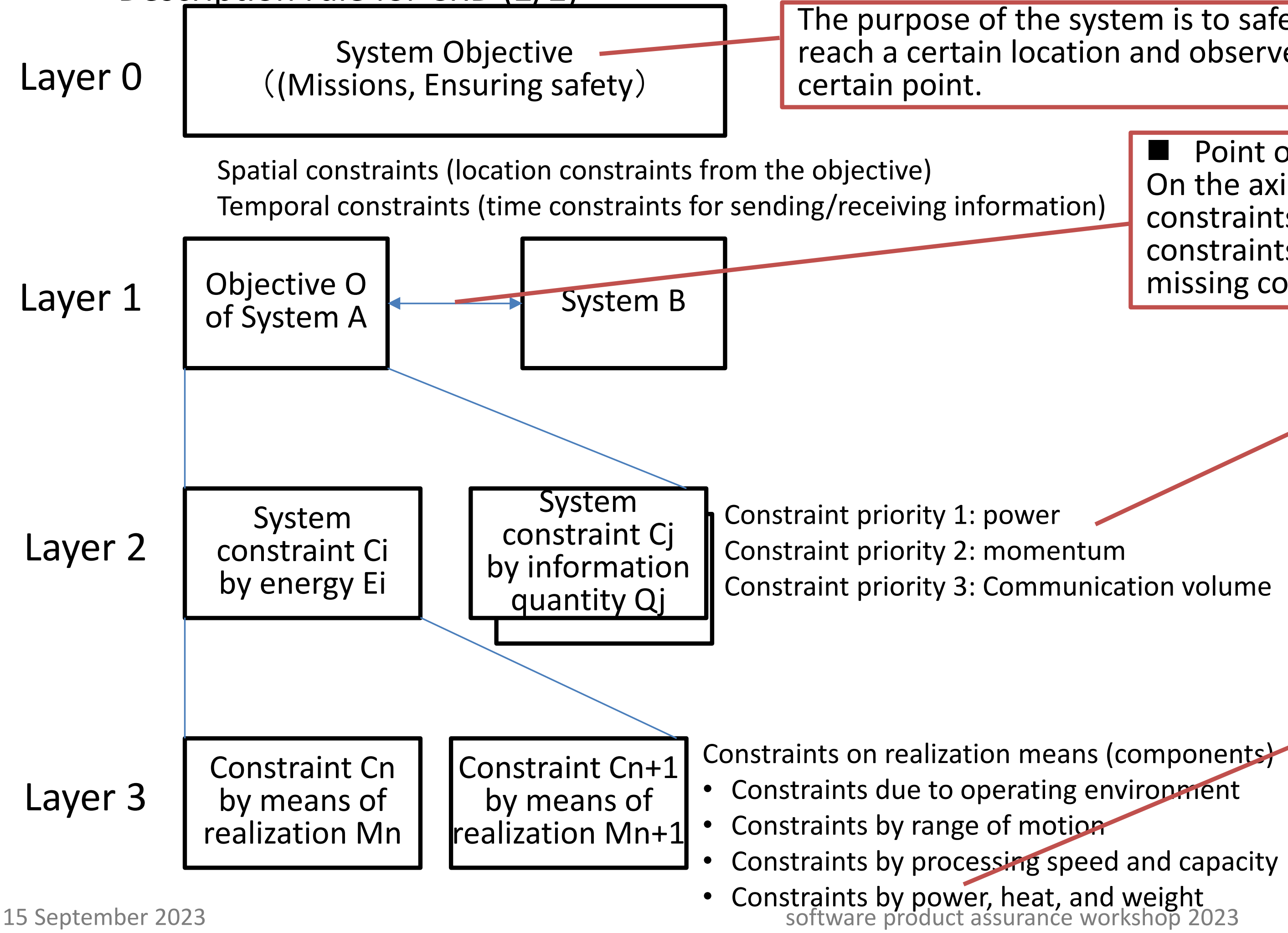

User requirement Ru
System Objective Formula M
Decompose by location or time
Use Case Def(U)
Energy constraint Cp(i) /Information amount constraint Cq(j)
Decompose by means of realization
Correspondence between functions and components Allocate（e,f)
Constraints by means of realization components Cr、Cs

In the case of a system architecture with few component options as a means of implementation.

In the case of a system architecture with many component options as a means of implementation.

1.1 System functional design based on operational scenarios using constraint relations

- Description rule for CRD (2/2)


Layer 0
System Objective
((Missions, Ensuring safety)
Layer 1
Objective O of System A
System B
Layer 2
System constraint Ci by energy Ei
System constraint Cj by information quantity Qj
Layer 3
Constraint Cn by means of realization Mn
Constraint Cn+1 by means of realization Mn+1

Spatial constraints (location constraints from the objective)
Temporal constraints (time constraints for sending/receiving information)

Constraint priority 1: power
Constraint priority 2: momentum
Constraint priority 3: Communication volume

Constraints on realization means (components)
- Constraints due to operating environment
- Constraints by range of motion
- Constraints by processing speed and capacity
- Constraints by power, heat, and weight

The purpose of the system is to safely reach a certain location and observe a certain point.

■ Point of Method
On the axis of time and space, clarify the relationship of constraints for each operational phase/phase set from the constraints of the means of realization, and check for missing conditions for function execution.

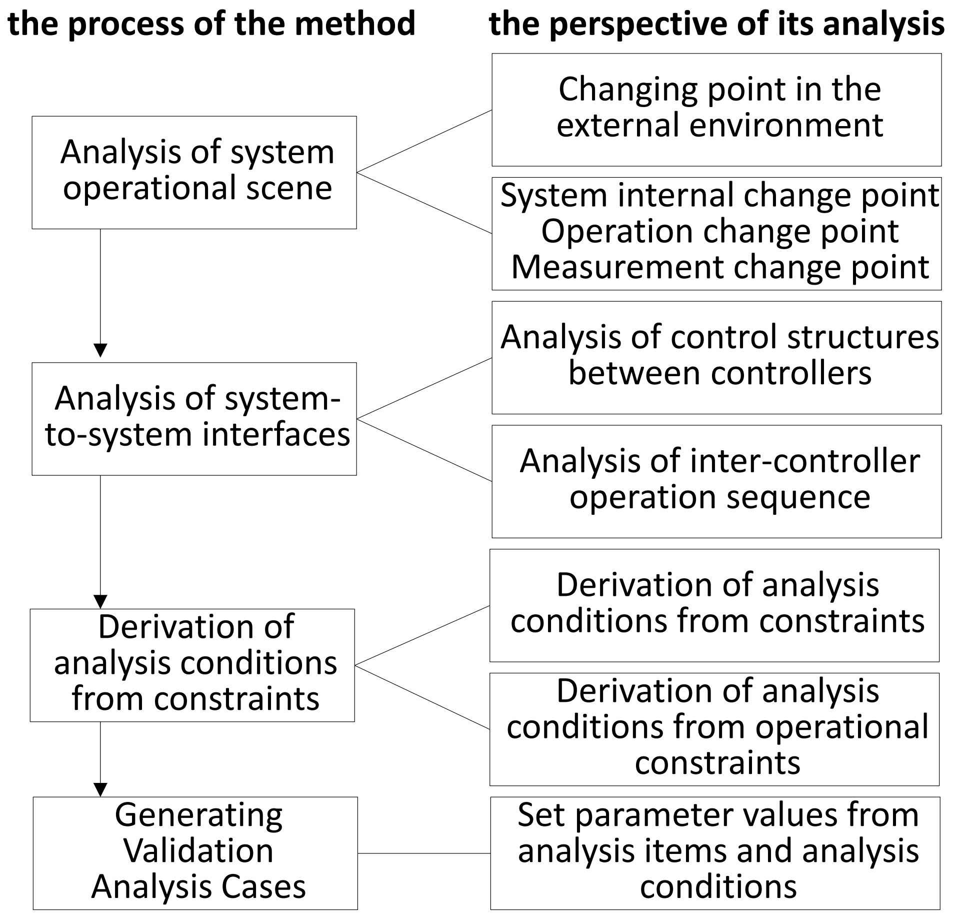
■ System features
The actions performed by the system such as observation, communication, charging, and heat dissipation are determined by the location of the system, each of which has a time constraint.

By determining each constraint and its distribution, the component selection criteria are clarified.

■ Target applying the method
The execution conditions of the function are overlooked when designing the function of the component.
why?
- For the constraints, which are the means of implementation, related constraints exist in two layers, but the effect is not considered.

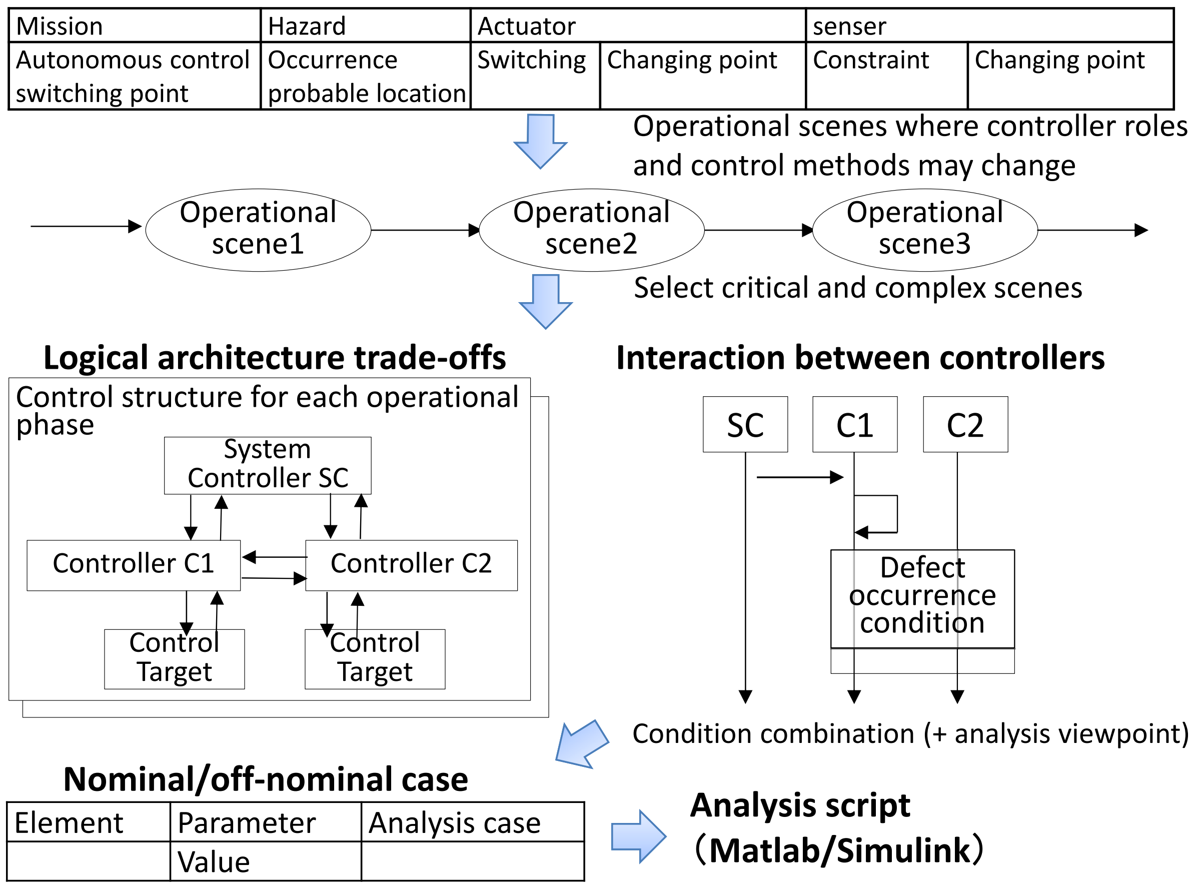
1.2 Software state transition analysis

The operation mode includes an artificially set mode such as an operation plan, and a physically set mode such as the operating state of the actuator.



the process of the method
the perspective of its analysis
Analysis of system operational scene
Changing point in the external environment
System internal change point Operation change point Measurement change point
Analysis of system-to-system interfaces
Analysis of control structures between controllers
Analysis of inter-controller operation sequence
Derivation of analysis conditions from constraints
Derivation of analysis conditions from constraints
Derivation of analysis conditions from operational constraints
Generating Validation Analysis Cases
Set parameter values from analysis items and analysis conditions

Analysis Framework Overview (Output images)



Mission	Hazard	Actuator		senser	
Autonomous control switching point	Occurrence probable location	Switching	Changing point	Constraint	Changing point

Operational scenes where controller roles and control methods may change
Operational scene1
Operational scene2
Operational scene3
Select critical and complex scenes
Logical architecture trade-offs
Control structure for each operational phase
System Controller SC
Controller C1
Controller C2
Control Target
Control Target
Interaction between controllers
SC
C1
C2
Defect occurrence condition
Condition combination (+ analysis viewpoint)
Nominal/off-nominal case
Analysis script (Matlab/Simulink)

Element	Parameter	Analysis case
	Value	

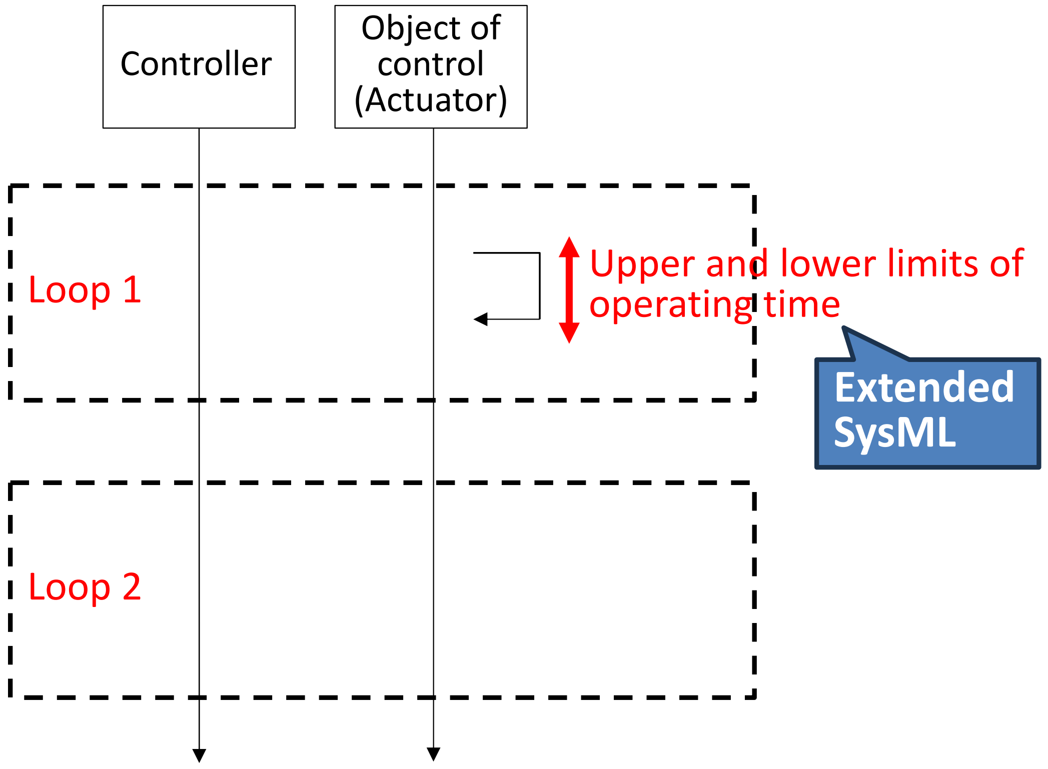
1.2 Software state transition analysis

Define the description rules in the SysML sequence diagram according to the characteristics of the system.

Description rule

Sequence	・2 loops ・Range of time constraints
1 message	・Sensor measurement and periodic processing ・Controller's judgment ・Actuator operation status
Multiple messages	・Events and order of operation ・Time constraints between events
Component	・Hazard Causes Actuator ・Controllers that prevent hazards

Description

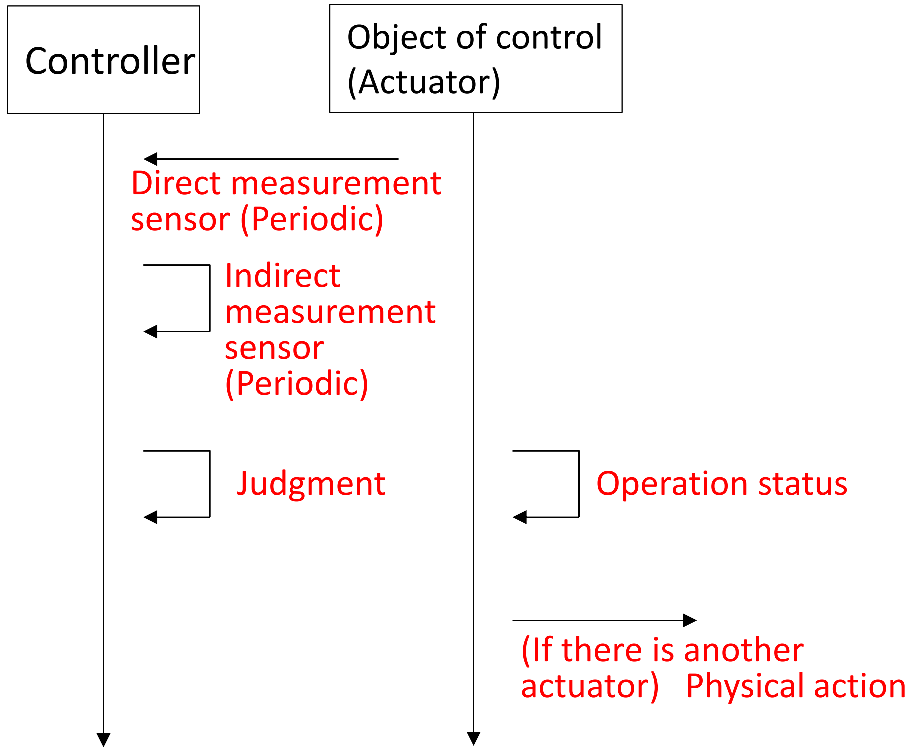

Controller
Object of control (Actuator)
Loop 1
Upper and lower limits of operating time
Extended SysML
Loop 2

1.2 Software state transition analysis

Description rule

Sequence	・2 loops ・Range of time constraints
1 message	・Sensor measurement and periodic processing ・Controller's judgment ・Actuator operation status
Multiple messages	・Events and order of operation ・Time constraints between events
Component	・Hazard Causes Actuator ・Controllers that prevent hazards

Description

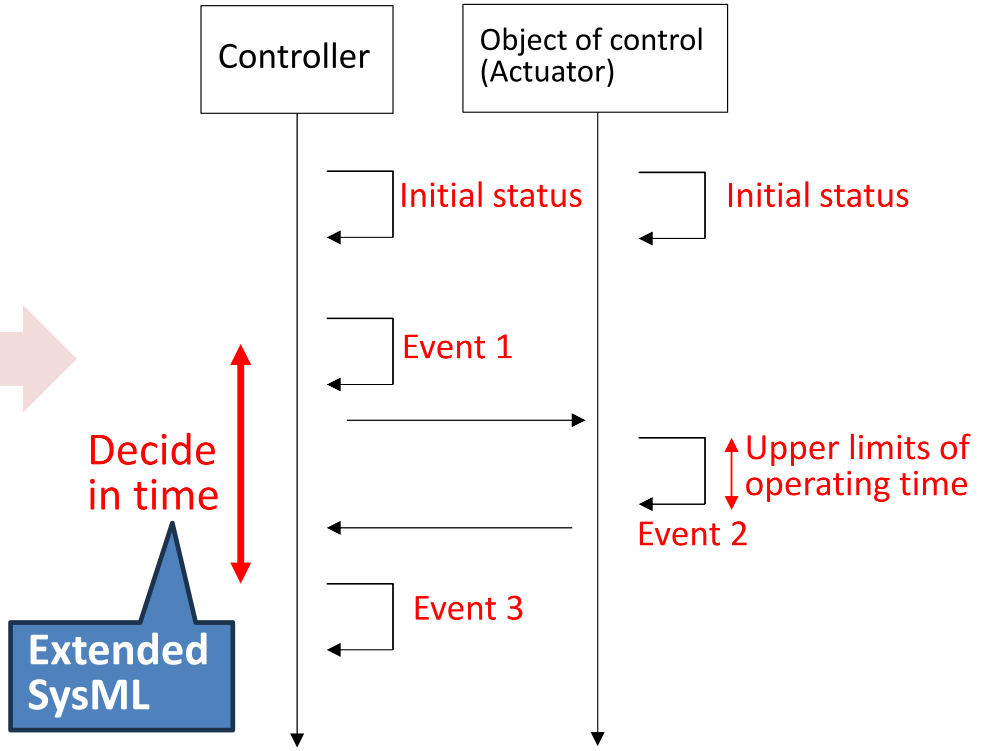

Controller
Object of control (Actuator)
Direct measurement sensor (Periodic)
Indirect measurement sensor (Periodic)
Judgment
Operation status
(If there is another actuator) Physical action

1.2 Software state transition analysis

Description rule

Sequence	･2 loops ･Range of time constraints
1 message	･Sensor measurement and periodic processing ･Controller's judgment ･Actuator operation status
Multiple messages	･Events and order of operation ･Time constraints between events
Component	･Hazard Causes Actuator ･Controllers that prevent hazards

Description

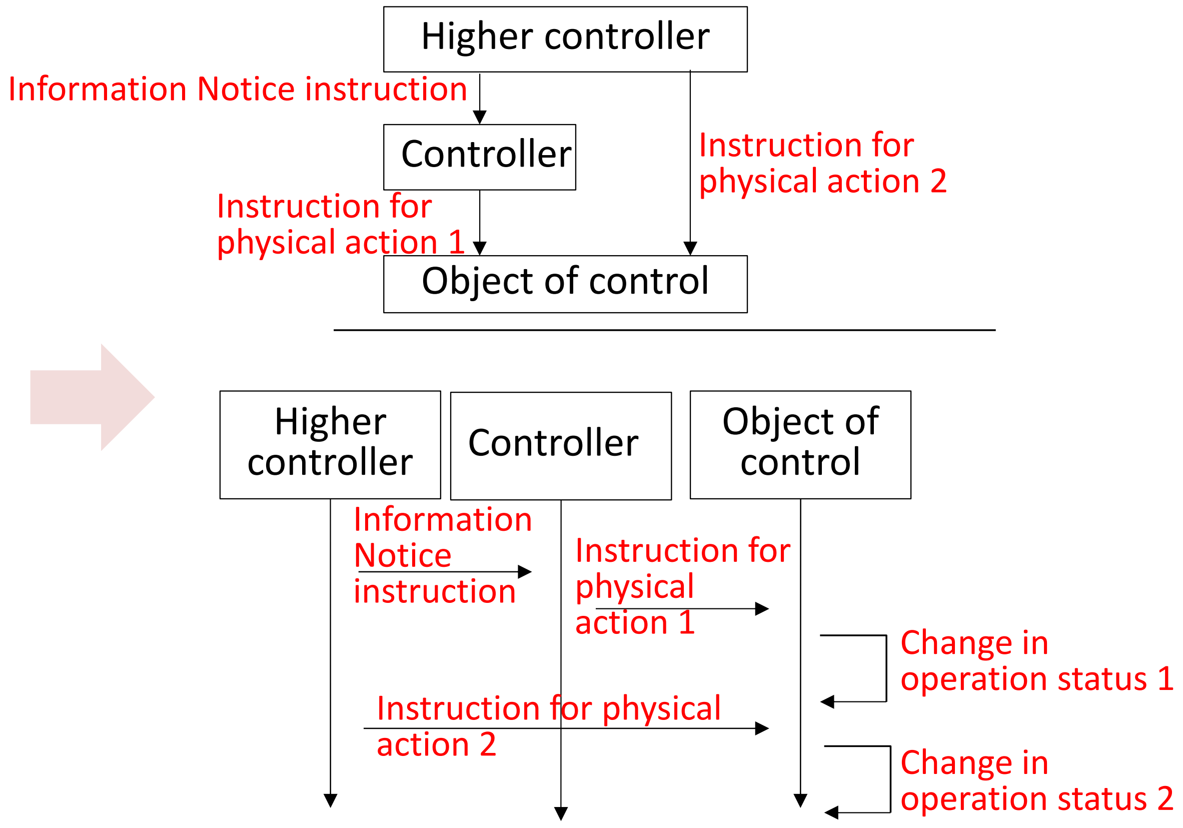


1.2 Software state transition analysis

Description rule

Sequence	・2 loops ・Range of time constraints
1 message	・Sensor measurement and periodic processing ・Controller's judgment ・Actuator operation status
Multiple messages	・Events and order of operation ・Time constraints between events
Component	・Hazard Causes Actuator ・Controllers that prevent hazards

Description


Higher controller
Information Notice instruction
Controller
Instruction for physical action 2
Instruction for physical action 1
Object of control
Higher controller
Controller
Object of control
Information Notice instruction
Instruction for physical action 1
Change in operation status 1
Instruction for physical action 2
Change in operation status 2

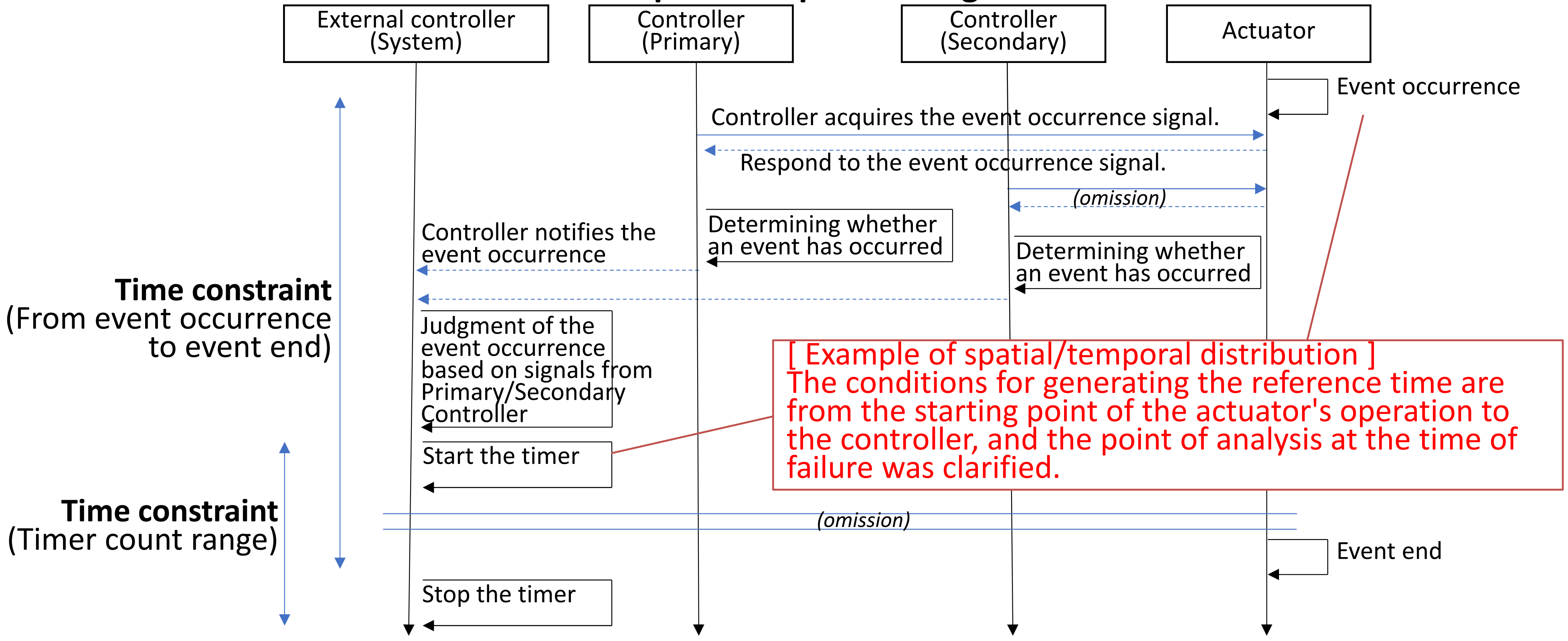
1.2 Software state transition analysis

■ Modeling rules set for verification purposes

In order to perform safe operation (actuator operation) after a certain time has elapsed since the event occurred,

- The conditions for starting and stopping the timer (including and/or) must be specified.
- The information about the timer must be specified in the message between each controller.
- The logic for safety such as "detection and judgment" shall be specified individually.

Example of Sequence Diagram



External controller (System)
Controller (Primary)
Controller (Secondary)
Actuator
Event occurrence
Controller acquires the event occurrence signal.
Respond to the event occurrence signal.
(omission)
Determining whether an event has occurred
Determining whether an event has occurred
Controller notifies the event occurrence
Time constraint
(From event occurrence to event end)
Judgment of the event occurrence based on signals from Primary/Secondary Controller
Start the timer
Time constraint
(Timer count range)
(omission)
Event end
Stop the timer

[Example of spatial/temporal distribution]
The conditions for generating the reference time are from the starting point of the actuator's operation to the controller, and the point of analysis at the time of failure was clarified.

2. INTRODUCTION OF JAXA’S AUTOMATIC CODE GENERATION

2 Introduction of JAXA's Automatic Code Generation

The software development standard for spacecraft, JERG-2-610, was revised in 2021.
The requirements shown in the table below are shared.

- Requirements for the automatic code generation (1/4)

Description related to automatic code generation of JERG-2-610B	
Process	Requirement
5.3.1 Process implementation	5.3.1.1(2) Based on the development strategy, a software development plan including the following information shall be established to cover:
	5.3.1.1(2)(f) Definition of activities in accordance with automatic code generation tools, scope (∗3), verification policy (∗4), management policy, the tools to be used and selection criteria and rationale applied to the selection of the tools, and the automatic code generation handbook when automatic code generation is implemented
	5.3.1.1(2)(m) Environment to be used for the software development and verification (enabling systems or services such as simulators, real hardware, test environment, source code analysis tools, and automatic code generation tools) shall be identified and made available
	∗3: Functions and modules to which automatic code generation is applied.
	∗4: Verification policy defines source code review, verification methods (model based simulation and model unit testing) of source data (e.g., models) for automatic code generation, unit testing, integration tests, equivalence evaluation of models and code, and comprehensive verification strategy combining them.
5.3.4 Computer system architectural design	5.3.4.1(9) Evaluation criteria for the computer system architectural design shall be defined, and the results of the computer system architectural design shall be evaluated in accordance with these criteria. Additionally, rationale for the selection of the computer system architectural design shall be recorded. When model based development is implemented, the validity of the design shall be evaluated by applying model based simulation with the level of the computer system architectural design specifications.
5.3.5 Software requirements analysis	5.3.5.1(14) When automatic code generation is implemented, the interface between manually-developed source code and automatically-generated code shall be analyzed in accordance with the scope decided in the establishment of the (software devvelopment) plan.
	5.3.5.1(19) When model based development is implemented, the validity of the requirements shall be evaluated by applying model based simulation with the level of the software requirements specifications.

2 Introduction of JAXA's Automatic Code Generation

- Requirements for the automatic code generation (2/4)

Description related to automatic code generation of JERG-2-610B	
Process	Requirement
5.3.6 Software design	5.3.6.1 <u>Software architectural design</u> (1) Functional decomposition and module partitioning shall be performed based on the software requirements specifications and module structures and the structures between the modules comprising the functions shall be clarified, so that an appropriate software architectural design is performed. When automatic code generation is implemented, the scope of automatic code generation decided in the establishment of the (software devvelopment) plan shall be established, and interface specifications between manually-developed source code and automatically-generated code shall be established.
	5.3.6.1 <u>Common to software architectural and detailed designs</u> (8) When model based development is implemented, the validity of the design shall be evaluated by applying model based simulation.
	(11) When automatic code generation is implemented, source data (e.g., models) for automatic code generation shall be developed in accordance with the automatic code generation handbook.
	(16) When automatic code generation is implemented, model unit testing shall be performed so that the predetermined evaluation criteria for model testing coverage are met.
	Model unit testing : When automatic code generation is implemented, a test equivalent to unit testing is performed by using source data (e.g., models) for automatic code generation in the software design process or other processes. The test specifications (test cases) used in this testing are applied to the equivalence evaluation of unit testing in the software coding and testing process with the same specifications.
	Model testing coverage : The degree of coverage for a model when automatic code generation is implemented and model unit testing is performed. Criteria required for quality evaluation are required to be set in advance. A part of the evaluation criteria may be fulfilled by implementing model based simulation.

2 Introduction of JAXA's Automatic Code Generation

- Requirements for the automatic code generation (3/4)

Description related to automatic code generation of JERG-2-610B	
Process	Requirement
5.3.8 Software coding and testing	5.3.8.1(2) Coding and unit testing
	5.3.8.1(2)(b) When automatic code generation is implemented, the automatic code generation shall be performed in accordance with the automatic code generation handbook.
	5.3.8.1(2)(c) Source code shall be developed based on the defined coding standard. When automatic code generation is implemented, it shall be developed in accordance with the automatic code generation handbook.
	5.3.8.1(2)(d) The review of source code shall be performed according to the verification policy of the (software) development plan.
	5.3.8.1(2)(e) Unit testing specifications shall be developed in accordance with the software verification plan, the software test plan and the acceptance criteria defined in (1) (a) above. When automatic code generation is implemented, unit testing specifications including the test specifications (test cases) used for the model unit testing shall be developed.
	5.3.8.1(2)(f) Unit testing shall be performed in accordance with the unit testing specifications, and the test results shall be recorded in a format that it allows determination of pass or failure.
	5.3.8.1(2)(g) For unit testing, the test shall be performed so that the criteria of the test coverage for source code are satisfied. When automatic code generation is implemented, in addition, model unit testing is performed in the software design process (refer to 5.3.6.1 (16)). Then, unit testing for automatically-generated code is performed in accordance with the test specifications (test cases) used for the model unit testing, and the equivalence between source data (e.g., models) for automatic code generation and the automatically-generated code shall be evaluated.
	5.3.8.1(2)(h) Static analysis shall be performed with a code checking tool or equivalent and the source code quality shall be evaluated. (Automatically-generated code is included.)
5.3.10 Software integration	5.3.10.1(2) Implementation of Integration
	5.3.10.1(2)(b) When automatic code generation is implemented, software generated from manually-developed source code and one generated from automatically-generated code shall be integrated. Then, the applicability of the integrated one with the interface specifications clarified in the design shall be checked.

2 Introduction of JAXA's Automatic Code Generation

- Requirements for the automatic code generation (4/4)

Description related to automatic code generation of JERG-2-610B	
Process	Requirement
5.3.11 Software integration test	5.3.11(1) Test preparation
	5.3.11(1)(b) For software test, the following shall be considered:
	5.3.11(1)(b)(vii) Equivalence between source data (e.g., models) for automatic code generation and automatically-generated code when automatic code generation is implemented
	5.3.11(1)(b)(viii) When automatic code generation is implemented, software generated from manually-developed source code and that generated from automatically-generated code shall be integrated. Then, the validity of correct software behavior in an environment equivalent to the real target shall be checked.
5.5 Maintenance process	5.5.3 Modification implementation If an automatically-generated code part is modified manually, the same activities as for manually-developed source code shall be performed.
6.2 Configuration management process	6.2.2 Configuration identification
	6.2.2(2) For configuration items, the following shall be identified:
	(a) Version references (∗1)
	(b) Other identification details (∗2)
	∗1: When model based development and automatic code generation are implemented, the version of each of the elements such as models to be used, source data (e.g., models) for automatic code generation, tools, and simulators shall be included.
	∗2: The configuration management information of the following shall be able to be referred to: models, tools, simulators, parameters, and other matters used in model based simulation and model unit testing, and source data (e.g., models) for automatic code generation and tools for automatically-generated code.

2 Introduction of JAXA’s Automatic Code Generation

- The software assurance requirements for for the automatic code generation (1/2)

Description related to automatic code generation of JERG-2-610B	
Process	Requirement
6.3 Quality assurance process	6.3.1.3 Establishment of a quality assurance activities plan A quality assurance activity plan shall be established in accordance with the quality assurance strategy The quality assurance activity plan shall include the following: (10) Management in automatic code generation
	6.3.7 Management in automatic code generation When automatic code generation is implemented, situations of the implementation of the following management items shall be evaluated:
	(1) Selection of tools for automatic code generation Criteria are defined in terms of the evaluation viewpoints below, and the results of the evaluation of these criteria are documented when the tools required for automatic code generation, such as those for developing source data (e.g., models) for automatic code generation; generating code automatically from source data (e.g., models) for automatic code generation; collecting the metrics of source data (e.g., models) for automatic code generation, and managing the configuration of source data (e.g., models) for automatic code generation and parameters used for these source data (e.g., models); and other functions. The results can be replaced with the following perspectives based on the evaluation of achievements of other products having equal quality requirements. (a) Compliance with the identified automatic code generation handbook (b) Measurement environment of model testing coverage in model unit testing (c) Compatibility in cooperation with tools used in model based simulation and model unit testing (d) Compatibility with other tools (e.g., compilers and code management systems) using automatically-generated code as one of inputs (e) Whether tools required for the project can be customized or not (f) Configuration change control of tools including parameters (g) Whether the quality assurance information of tools (e.g., versions and upgrading information) is available or not (h) Performance of automatically-generated code (size and speed)
	(2) Modification of automatically-generated code (a) Automatically-generated code shall not be modified in the code itself but modified through the source data (e.g., models) for automatic code generation. (b) If automatically-generated code is modified manually from necessity, activities of the development and configuration management processes, such as compliance with coding standards, the review of code, and unit testing, equivalent to those for manually-developed source code shall be performed.

2 Introduction of JAXA's Automatic Code Generation

- The software assurance requirements for the automatic code generation (2/2)

Description related to automatic code generation of JERG-2-610B	
Process	Requirement
6.4 Verification process	6.4.2.3 Design verification The following viewpoint shall be considered:
	(6) When automatic code generation is implemented, model unit testing is performed.
	6.4.2.4 Source code verification The following viewpoint shall be considered:
	(5) When automatic code generation is implemented, equivalence between source data (e.g., models) for automatic code generation and automatically-generated code is evaluated.

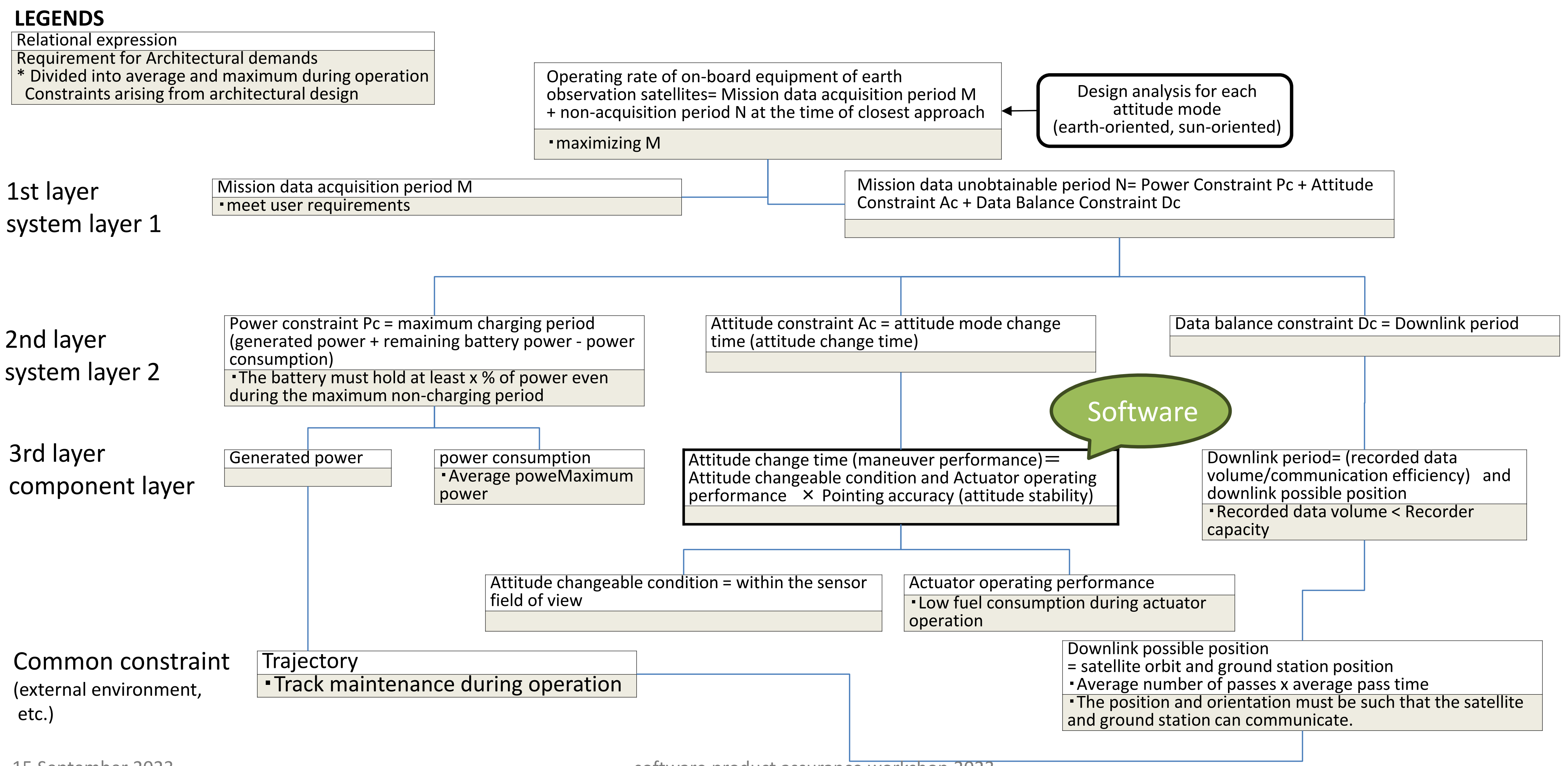
Conclusion

- MBSE method focusing on constraints
  - Number of Demonstrated Projects
    - 5 projects
  - Applied subsystem
    - Attitude and Orbit Control Subsystem
  - Effectiveness and Outlook
    - The above two Methods are moving to the stage of application to actual projects because they are able to solve the issues through demonstration.
- Automatic Code Generation
  - Number of projects applied
    - 3 projects
  - Applied subsystem
    - Attitude and Orbit Control Subsystem
  - Application Policy
    - Apply to newly developed software
  - Effectiveness and Outlook
    - Since the software development standard were established two years ago, automatic code generation has been completed. No particular issues have been found in this phase so far, and the process is going well.

Thank you for listening!

Any questions & comments

Examples of Constraint relational expressions (CRD) according to architectural characteristics



LEGENDS

Relational expression

Requirement for Architectural demands

* Divided into average and maximum during operation
Constraints arising from architectural design

Operating rate of on-board equipment of earth observation satellites= Mission data acquisition period M + non-acquisition period N at the time of closest approach

・maximizing M

Design analysis for each attitude mode (earth-oriented, sun-oriented)

1st layer
system layer 1

Mission data acquisition period M

・meet user requirements

Mission data unobtainable period N= Power Constraint Pc + Attitude Constraint Ac + Data Balance Constraint Dc

2nd layer
system layer 2

Power constraint Pc = maximum charging period (generated power + remaining battery power - power consumption)

・The battery must hold at least x % of power even during the maximum non-charging period

Attitude constraint Ac = attitude mode change time (attitude change time)

Data balance constraint Dc = Downlink period

Software

3rd layer
component layer

Generated power

power consumption

・Average poweMaximum power

Attitude change time (maneuver performance)＝ Attitude changeable condition and Actuator operating performance × Pointing accuracy (attitude stability)

Downlink period= (recorded data volume/communication efficiency) and downlink possible position

・Recorded data volume < Recorder capacity

Attitude changeable condition = within the sensor field of view

Actuator operating performance

・Low fuel consumption during actuator operation

Common constraint
(external environment, etc.)

Trajectory

・Track maintenance during operation

Downlink possible position
= satellite orbit and ground station position
・Average number of passes x average pass time

・The position and orientation must be such that the satellite and ground station can communicate.