SysML Modeling Guide for Target System

Table of Contents

Change History

Modification Date	Modifications
2014/01/10	Created

1 Scope

This document guides the notation and definition of SysML in order to enable the collaboration of D-Case and SysML.

2 Overview of D-Case and SysML Modeling Guide

2.1 Background and Purpose

Recently, embedded systems are used by users in many fields. They have become complicated to satisfy various demands. The demands consist not only of functional demands from users but also of non-functional demands related to dependability. Dependability includes attributes of safety, reliability, availability, integrity, maintainability.

This guide shows an approach by D-Case which consistently realizes the dependability of the target system from upper process to lower process. Table 2-1 shows what is asked for developing a dependable system.

Table 2-1 What is asked for developing a dependable system

Development Phase	What is asked
Requirements Definition	System demands should satisfy dependability
System Design	Design specifications should reflect the demands correctly
System Verification	Verification results should account for satisfying dependability

In requirements definition phase, system demands should be derived by removing all the factors which inhibit dependability so that system demands satisfy dependability. D-Case is utilized for just enough derivation of system demands. D-Case decomposes dependability based on defining threat to dependability, scene of threat, cause, and provision. This decomposition extracts all the factors which inhibit dependability and marshals their provisions as system demands.

In system design phase, system design should be performed by utilizing the design information which is included in the demands based on dependability so that design specifications reflect the demands correctly. Derived design specifications should also be verified just enough by checking with the demands based on dependability. To realize this, design information like functional or non-functional demand, system element, restriction to element, and verification condition is extracted from the demands derived from dependability, and design specifications are correctly derived. Next, design specifications are analyzed by correspondence check with the demands based on dependability.

In system verification phase, verification results should be associated to the demands or design specifications, and their positions should be clarified so that verification results satisfy dependability.
The flow of the method is shown in Figure 2-1.

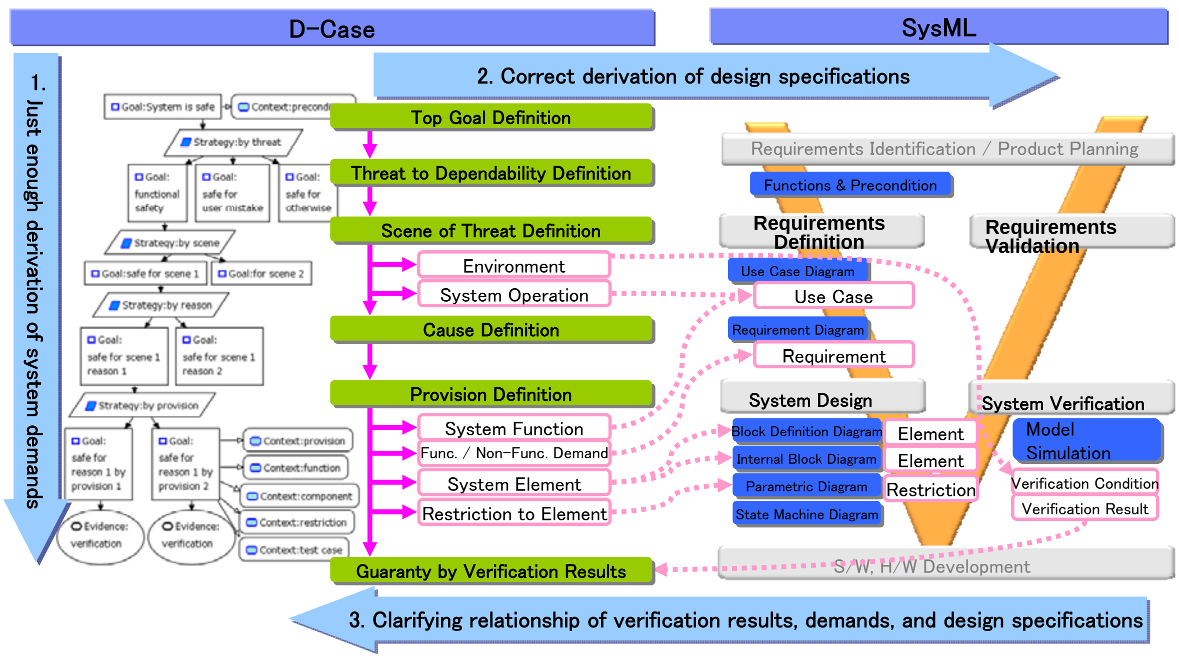


Figure 2-1 Dependable System Development Method by D-Case SysML Collaboration

In this method, D-Case improves quality by reflecting development intents from upper process to lower process. SysML models can be made by reflecting the system demands derived from D-Case decomposition. As the sub-goals derived from the D-Case decomposition contain activities needed by the development, the accuracy of development plan can be enhanced by reflecting the activities to the plan.
This method is guided in the following documents:

- D-Case Modeling Guide for Target System
- SysML Modeling Guide for Target System
- D-Case Template
- SysML Template

2.2 Target System of Modeling Guide

Target of this guide is the in-vehicle system complying with ISO26262, the global standard of functional safety for vehicles. The derivational development is assumed in that the intents such as safety demands or reliability demands are reflected to previous model already developed to adapt functional safety.

2.3 Constitution of Modeling Guide

Relationship of modeling flow by this method and ISO26262 safety lifecycle is shown in Figure 2-2. The constitution of D-Case decomposition corresponds to ISO26262 part 3 concept phase and part 4 product development: system level.

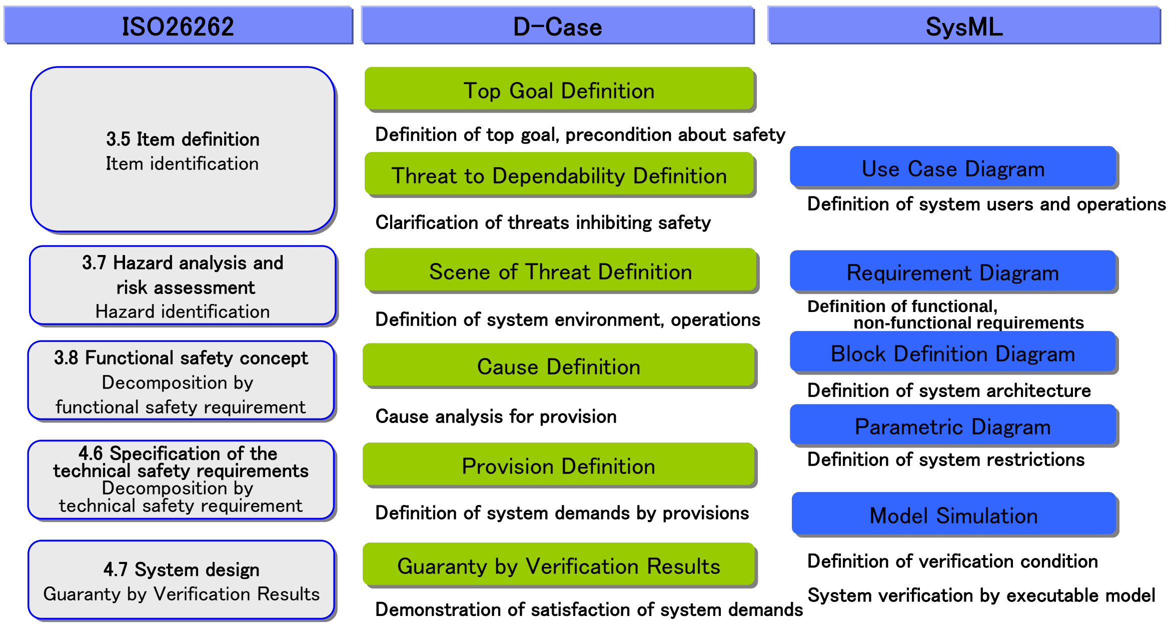

ISO26262
D-Case
SysML
3.5 Item definition
Item identification
3.7 Hazard analysis and risk assessment
Hazard identification
3.8 Functional safety concept
Decomposition by functional safety requirement
4.6 Specification of the technical safety requirements
Decomposition by technical safety requirement
4.7 System design
Guaranty by Verification Results
Top Goal Definition
Definition of top goal, precondition about safety
Threat to Dependability Definition
Clarification of threats inhibiting safety
Scene of Threat Definition
Definition of system environment, operations
Cause Definition
Cause analysis for provision
Provision Definition
Definition of system demands by provisions
Guaranty by Verification Results
Demonstration of satisfaction of system demands
Use Case Diagram
Definition of system users and operations
Requirement Diagram
Definition of functional, non-functional requirements
Block Definition Diagram
Definition of system architecture
Parametric Diagram
Definition of system restrictions
Model Simulation
Definition of verification condition
System verification by executable model

Figure 2-2 Relationship of Modeling Flow and ISO26262

Constitution of D-Case, SysML modeling guide is shown in Table2-2.

Table2-2. Constitution of Modeling Guides

Target	D-Case		SysML	
Category	D-Case Structure	Node Notation	Association from D-Case	Association to D-Case
Item	Item Definition	Goal to Achieve, Environment and Restriction	-	-
	Identification of Hazards	Environment and Operation of System	Environment and Operation of System	Use Case Verification condition
	Decomposition by Functional Safety Requirements	Detailed Cause to Take Actions	-	-
	Decomposition by Technical Safety Requirements	System Requirement	Func. / non-func. requirement, System elements, Restriction	Use Case, Requirement, Component, and Restriction
	Guaranty by Verification Results	Information required to Verification	Condition and Processing of Control	Verification Result

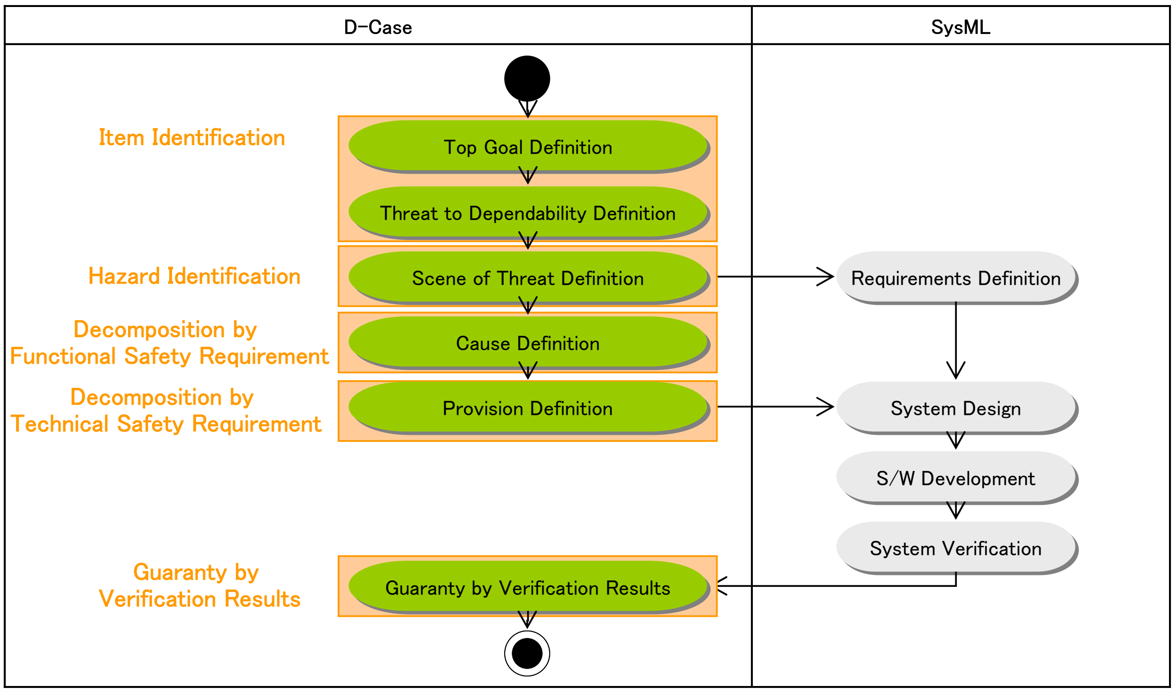
This guide mainly guides as follows:

1. **D-Case structure based on ISO26262 safety lifecycle**
   The property which the system for development should satisfy is described as D-Case top goal in the form of proposition. The whole structure of D-Case decomposition is considered to accomplish the D-Case. The top goal is divided into ISO26262 part and other part based on ISO26262. The D-Case of ISO26262 is divided by utilizing work products made in the activities of safety lifecycle. The decomposition flow of D-Case is explained in "D-Case Modeling Guide for Target System".

2. **Notation of D-Case nodes providing information needed by SysML model**
   The information needed by SysML model is described in the process of D-Case decomposition. The notation which is suitable for SysML collaboration is explained in "D-Case Modeling Guide for Target System".

3. **Writing procedure of SysML model based on the information of D-Case**
SysML models are created or updated by extracting information needed for SysML model. Procedures are explained in chapter 4, 5, and 6.

Figure 2-3 shows the modeling flow of D-Case and SysML.



D-Case
SysML
Item Identification
Top Goal Definition
Threat to Dependability Definition
Hazard Identification
Scene of Threat Definition
Requirements Definition
Decomposition by Functional Safety Requirement
Cause Definition
Decomposition by Technical Safety Requirement
Provision Definition
System Design
S/W Development
System Verification
Guaranty by Verification Results
Guaranty by Verification Results

Figure 2-3 Modeling Flow of D-Case and SysML

3 Development Process

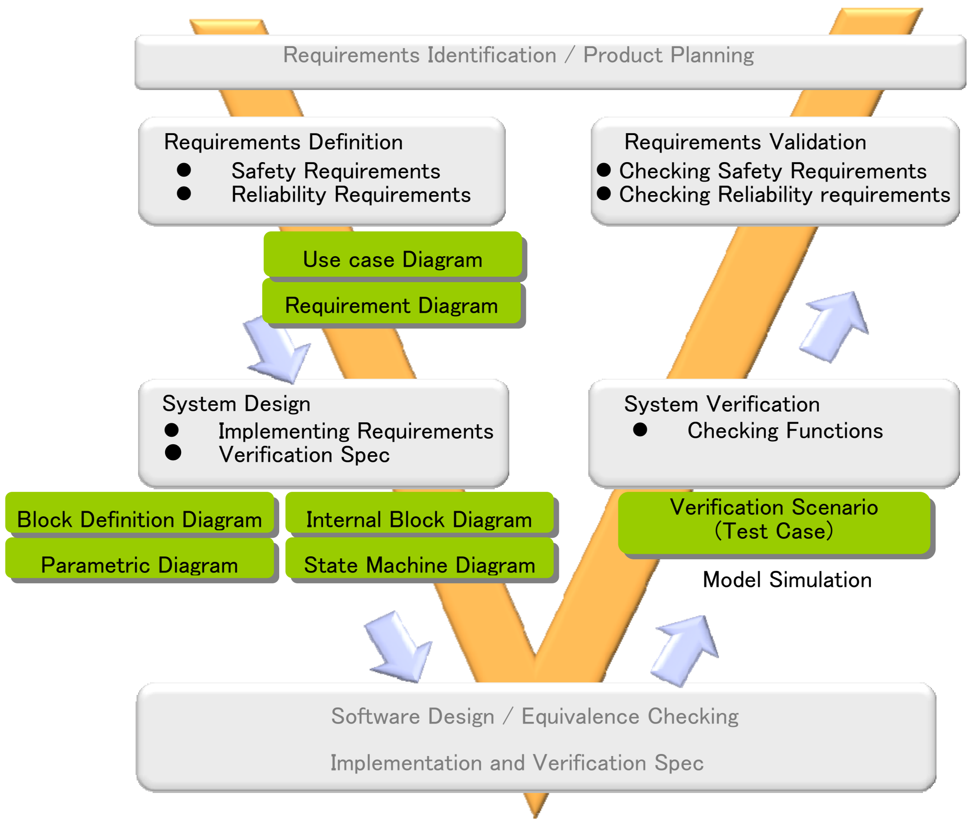


Figure 3-1 Development Process Applying SysML

This document premises that in-vehicle systems are developed following the development process shown in Figure 5-1. Also premises that derivational development is applied where new models are developed by deriving former models utilizing former work products such as specifications and models.

(Requirements Definition Phase)

In requirements definition phase, use case diagrams and requirement diagrams are updated. Use case diagrams define relationships between users and operations of a system, and boundaries of external systems. Requirement diagrams define detailed functional requirements and non-functional requirements of a system. The way of definition and example about use case diagram and requirement diagram is shown in chapter 4.

(System Design Phase)

In system design phase, block definition diagrams, parametric diagrams, internal block diagrams, and state machine diagrams are updated. Block definition diagrams define system architecture. Parametric diagrams define restrictions, related values and mathematical expressions of a system. Internal block diagrams define internal design of blocks defined in the block definition diagram. State machine diagrams define dynamical behavior of a system. The way of definition and example about block definition diagram, parametric diagram, internal block diagram, and state machine diagram is shown in chapter 5.

(System Verification Phase)

In system verification phase, model simulations are executed based on verification scenarios defined in state machine diagrams and source codes automatically generated from state machine diagrams. The way of definition and example about verification scenario is shown in chapter 6.

4 Requirements Definition

4.1 Use Case Diagram

Use case diagram defines system users and external systems. Use case diagram consists of actor, use case, communication path, and subject. The way of definition and example about the elements is shown below.

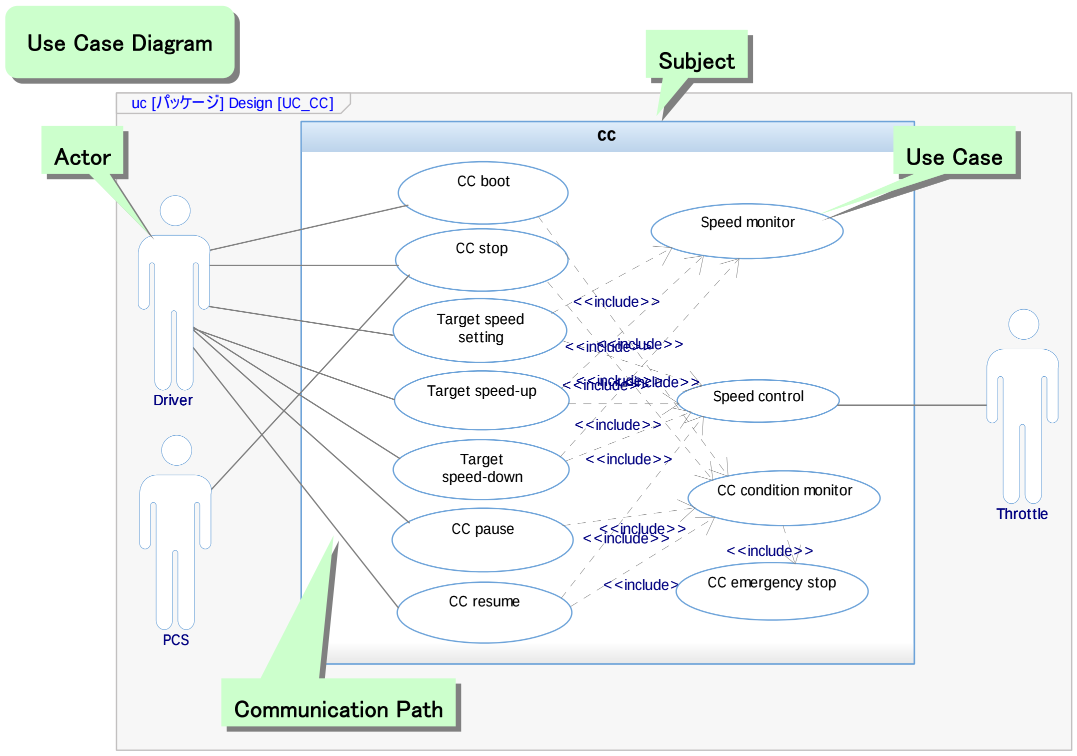


Figure 4-1 Example of a Use Case Diagram

(Actor)

An actor defines system user and external system. The way of defining actors is shown below.

- Basing on specifications and models of a former system, system users and external systems are defined as actors.
- In case where D-Case sub-goal "When ▼, ● system is safe for hazard X." is derived from D-Case strategy describing generation scene of threat inhibiting dependability "Argue about ● system's safety for every scene of threat.", extract systems users and external systems from the description corresponding to "When

▼" and define them as actors.

(Use Case)

A use case defines system function. The way of defining use cases is shown below.

- Extract system functions from specifications and models of a former model, and define them as use cases.
- In case where D-Case sub-goal "When ▼, ● system is safe for hazard X." is derived from D-Case strategy describing generation scene of threat inhibiting dependability "Argue about ● system's safety for every scene of threat.", extract system functions from the description corresponding to "When ▼" and define them as use cases.
- In case where D-Case sub-goal "When ▼, control which keeps ● value in tolerance level can be performed by ■ block even when a failure occurs at ▲ block." is derived from D-Case strategy describing provisions for causes deriving threats inhibiting dependability "Divide ● system's safety for every provision.", extract system functions from the description corresponding to "by ■ block" and define them as use cases if needed.
- In cases where D-Case is solved by each functional requirement based on "split by function pattern", extract system functions from them and define as use cases.

* Define use case from actor's point of view, not from system's point of view.

* Use case names should be defined, so that the sentence "actor name + use case name" makes sense.

* Use case description should be added, if needed.

(Communication Path)

Relationships between an actor and a use case used by that actor are defined as communication paths.

* Also define generalization, include, extend, etc, if needed.

(Subject)

Subsystems bundling multiple use cases configuring a single function are defined as Subjects.

(Association with D-Case)

If D-Case contains goal nodes associated with a use case defined in the use case diagram, traceability can be defined by associating that goal with the use case using context nodes of D-Case. At this time, the use case ID is written in “Desc” column or “Attachment” column of the context node. D-Case data collaboration function on SW development environment [1] is available.

(Correspondence check between D-Case and requirements)

Avoid lacks and mistakes of requirements definition activity by checking consistency between contents of requirement described in the D-Case goal nodes and contents of the use case diagrams.

(Example)

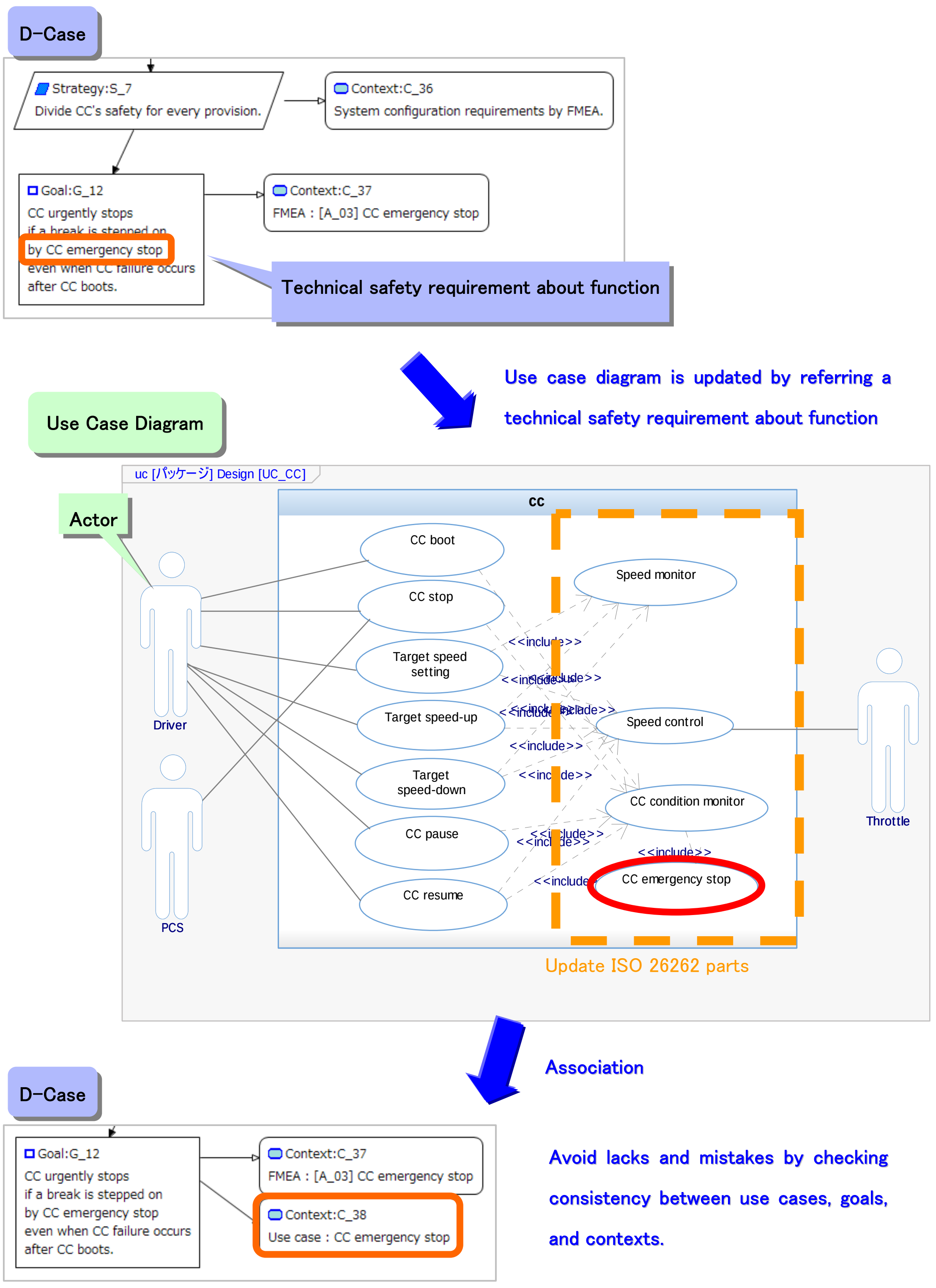

D-Case
Strategy:S_7
Divide CC's safety for every provision.
Context:C_36
System configuration requirements by FMEA.
Goal:G_12
CC urgently stops if a break is stepped on by CC emergency stop even when CC failure occurs after CC boots.
Context:C_37
FMEA : [A_03] CC emergency stop
Technical safety requirement about function
Use case diagram is updated by referring a technical safety requirement about function
Use Case Diagram
uc [パッケージ] Design [UC_CC]
Actor
cc
CC boot
CC stop
Target speed setting
Target speed-up
Target speed-down
CC pause
CC resume
Speed monitor
Speed control
CC condition monitor
CC emergency stop
<<include>>
Driver
PCS
Throttle
Update ISO 26262 parts
Association
D-Case
Goal:G_12
CC urgently stops if a break is stepped on by CC emergency stop even when CC failure occurs after CC boots.
Context:C_37
FMEA : [A_03] CC emergency stop
Context:C_38
Use case : CC emergency stop
Avoid lacks and mistakes by checking consistency between use cases, goals, and contexts.

Figure 4-2 Example of Updating a Use Case Diagram

An example of updating a use case diagram is shown in Figure 4-2. In this example, a description about functional safety is written in D-Case goal as a technical safety requirement (TSR) derived from FMEA results “CC urgently stops if a break is stepped on by CC emergency stop even when CC failure occurs after CC boots.” Based on this TSR, a function which is needed in point of functional safety “CC emergency stop” is extracted, and defined as a use case. This use case is associated as a D-Case context.
This use case has compatibility with an involved D-Case goal “CC urgently stops if a break is stepped on by CC emergency stop even when CC failure occurs after CC boots.” Thus requirement definition is proved to be correct and to have no lacks.

4.2 Requirement Diagram

Requirement diagram defines system functional requirements. Requirement diagram consists of requirement such as functional requirement and non-functional requirement, and path. The way of definition and example about the elements is shown below.

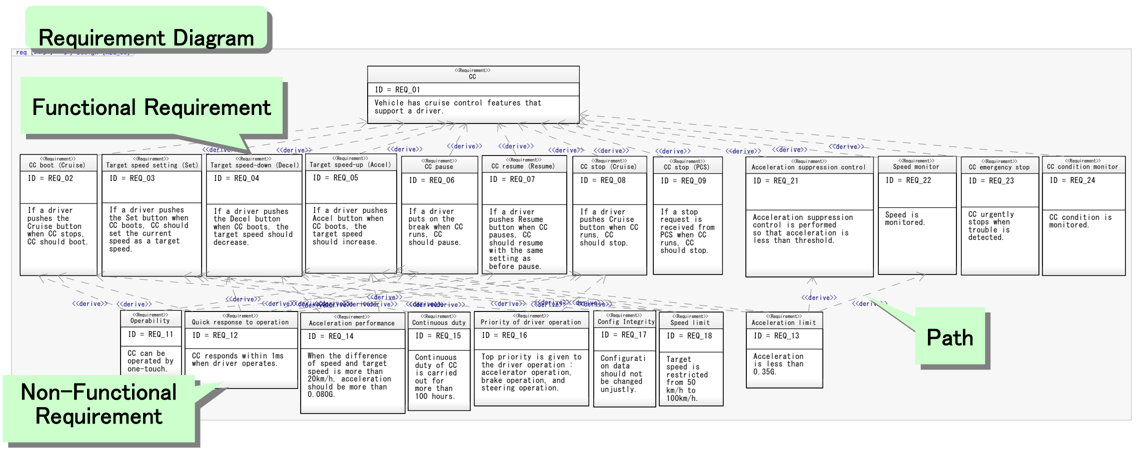

Requirement Diagram
Functional Requirement
Non-Functional Requirement
Path
<<Requirement>> CC ID = REQ_01 Vehicle has cruise control features that support a driver.
<<Requirement>> CC boot (Cruise) ID = REQ_02 If a driver pushes the Cruise button when CC stops, CC should boot.
<<Requirement>> Target speed setting (Set) ID = REQ_03 If a driver pushes the Set button when CC boots, CC should set the current speed as a target speed.
<<Requirement>> Target speed-down (Decel) ID = REQ_04 If a driver pushes the Decel button when CC boots, the target speed should decrease.
<<Requirement>> Target speed-up (Accel) ID = REQ_05 If a driver pushes Accel button when CC boots, the target speed should increase.
<<Requirement>> CC pause ID = REQ_06 If a driver puts on the break when CC runs, CC should pause.
<<Requirement>> CC resume (Resume) ID = REQ_07 If a driver pushes Resume button when CC pauses, CC should resume with the same setting as before pause.
<<Requirement>> CC stop (Cruise) ID = REQ_08 If a driver pushes Cruise button when CC runs, CC should stop.
<<Requirement>> CC stop (PCS) ID = REQ_09 If a stop request is received from PCS when CC runs, CC should stop.
<<Requirement>> Acceleration suppression control ID = REQ_21 Acceleration suppression control is performed so that acceleration is less than threshold.
<<Requirement>> Speed monitor ID = REQ_22 Speed is monitored.
<<Requirement>> CC emergency stop ID = REQ_23 CC urgently stops when trouble is detected.
<<Requirement>> CC condition monitor ID = REQ_24 CC condition is monitored.
<<Requirement>> Operability ID = REQ_11 CC can be operated by one-touch.
<<Requirement>> Quick response to operation ID = REQ_12 CC responds within 1ms when driver operates.
<<Requirement>> Acceleration performance ID = REQ_14 When the difference of speed and target speed is more than 20km/h, acceleration should be more than 0.080G.
<<Requirement>> Continuous duty ID = REQ_15 Continuous duty of CC is carried out for more than 100 hours.
<<Requirement>> Priority of driver operation ID = REQ_16 Top priority is given to the driver operation : accelerator operation, brake operation, and steering operation.
<<Requirement>> Config Integrity ID = REQ_17 Configuration data should not be changed unjustly.
<<Requirement>> Speed limit ID = REQ_18 Target speed is restricted from 50 km/h to 100km/h.
<<Requirement>> Acceleration limit ID = REQ_13 Acceleration is less than 0.35G.
<<derive>>

Figure 4-3 Example of a Requirement Diagram

(Functional Requirements)

A functional requirement defines functional viewpoint of demand for the system. The way of defining functional requirements is shown below.

- Extract functional requirements from specifications and models of a former model, and define them as requirement classes.
- In case where D-Case sub-goal "When ▼, control which keeps ● value in tolerance level can be performed by ■ block even when a failure occurs at ▲ block." is derived from D-Case strategy describing provisions for causes deriving threats inhibiting dependability "Divide ● system's safety for every provision.", extract functional requirements from the description corresponding to "by ■ block" and define them as requirement classes.
- In cases, where D-Case is solved by each functional requirement based on "split by function pattern", extract system functional requirements from them, and define as requirements.

* Functional requirement names must be detailed, unique and consistent.

(Non-Functional Requirements)

A non-functional requirement defines non-functional viewpoint of demand for the system. The way of defining non-functional requirements is shown below.

- Extract descriptions on dependability attributions such as safety, reliability, availability, integrity, and maintainability, and also, descriptions on other non-functional attributions shown in Table 1 such as functionality, usability, efficiency, portability, security, etc, from specifications and models of former model, and define them as requirement classes.
- In case where D-Case sub-goal "When ▼, control which keeps ● value in tolerance level can be performed by ■ block even when a failure occurs at ▲ block." is derived from D-Case strategy describing provisions for causes deriving threats inhibiting dependability "Divide ● system's safety for every provision.", extract non-functional requirements from the description corresponding to "control which keeps ● value in tolerance level" and define them as requirement classes.

Table 1 Attribution of Non-Functional Requirement

Attribution	Description
Safety	No destructive influences to users or environment. No injury to users.
Reliability	Continuity of proper service. Little chance of trouble.
Availability	Conformity nature of proper service. Immediate response to users' service request.
Integrity	No inadequate system changes.
Maintainability	Acceptable changes or repairs.
Functionality	Provide functions meeting explicit or implicit needs.
Usability	Understandable, acquirable, usable and attractive to users.
Efficiency	Reasonable performance against the resource consumed.
Portability	Attribution to transfer from one environment to another. Enable to operate in different environment without modification.
Security	Only allow authorized users to access restricted information.

(Path)

Define relations between requirement classes or relations with other SysML model elements as requirement containment relationship, copy dependency, derive dependency, satisfy dependency, verify dependency, refine dependency or trace dependency, etc.

（Association with D-Case）

If D-Case contains goal nodes associated with functional or non-functional requirements defined in the requirement diagram, traceability can be kept by associating those functional or non-functional requirements with the goals using D-Case context nodes. At this time, the requirement ID is written in “Desc” column or “Attachment” column of the context node. D-Case data collaboration function on SW development environment [1] is available.

（Correspondence check between D-Case and requirements）

Avoid lacks and mistakes in requirement definition by checking consistency between requirement contents defined in D-Case goal nodes and requirement diagrams.
Avoid lacks in requirements definition by checking if each of the goals divided by D-Case is associated to one or more functional or non-functional requirements.

(Example)

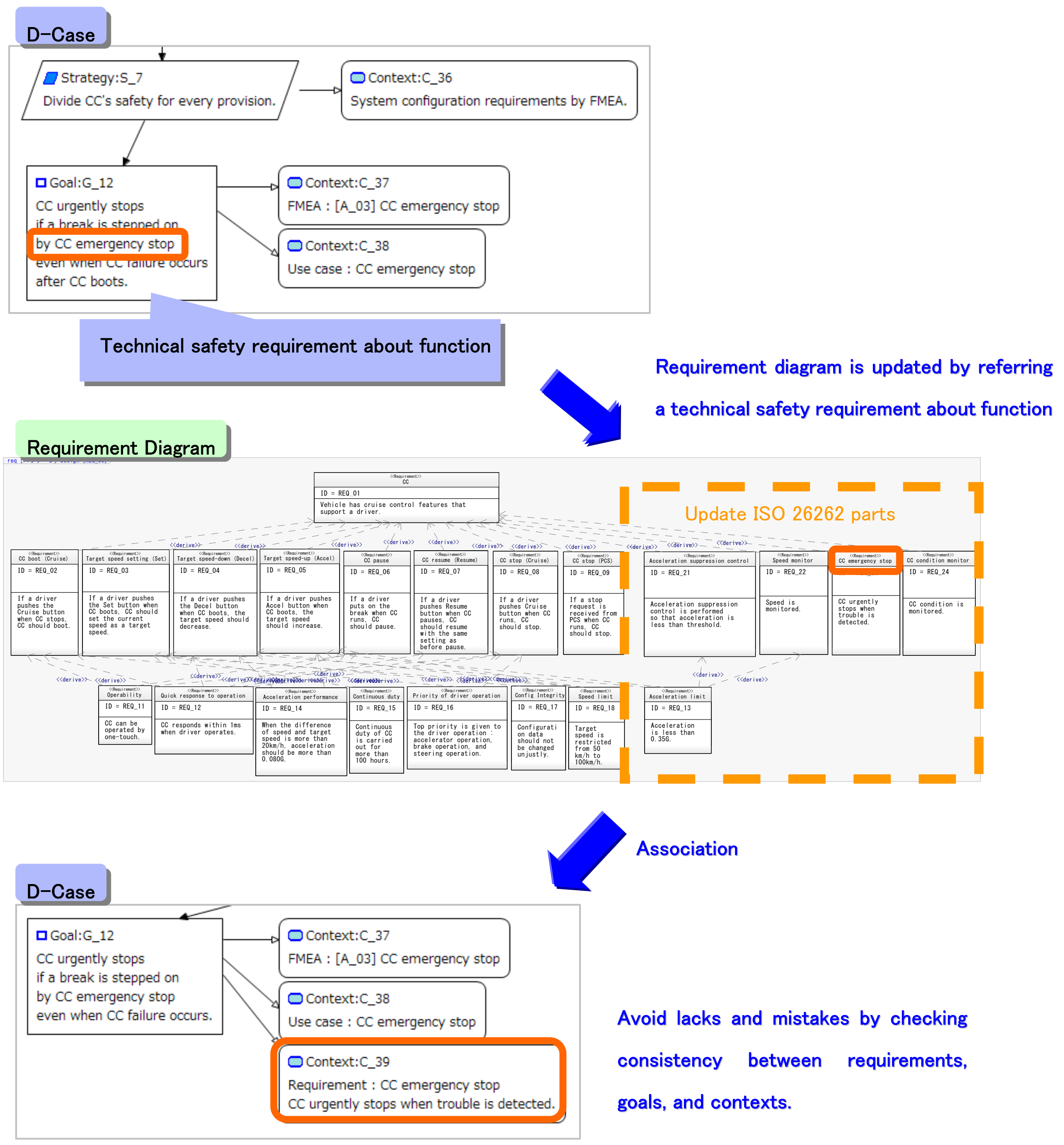

D-Case
Strategy:S_7
Divide CC's safety for every provision.
Context:C_36
System configuration requirements by FMEA.
Goal:G_12
CC urgently stops
if a break is stepped on
by CC emergency stop
even when CC failure occurs
after CC boots.
Context:C_37
FMEA : [A_03] CC emergency stop
Context:C_38
Use case : CC emergency stop
Technical safety requirement about function
Requirement diagram is updated by referring
a technical safety requirement about function
Requirement Diagram
Update ISO 26262 parts
CC emergency stop
CC urgently stops when trouble is detected.
Association
D-Case
Goal:G_12
CC urgently stops
if a break is stepped on
by CC emergency stop
even when CC failure occurs.
Context:C_37
FMEA : [A_03] CC emergency stop
Context:C_38
Use case : CC emergency stop
Context:C_39
Requirement : CC emergency stop
CC urgently stops when trouble is detected.
Avoid lacks and mistakes by checking
consistency between requirements,
goals, and contexts.

Figure 4-4 Example of Updating a Requirement Diagram

An example of updating a requirement diagram is shown in Figure 4-4. In this example, a description about functional safety is written in D-Case goal as a technical safety requirement (TSR) derived from FMEA results "CC urgently stops if a break is stepped on by CC emergency stop even when CC failure occurs after CC boots." Based on this

TSR, a function which is needed in point of functional safety “CC emergency stop” is extracted, and defined as a functional requirement. This functional requirement is associated as a D-Case context.

This functional requirement has compatibility with an involved D-Case goal “CC urgently stops if a break is stepped on by CC emergency stop even when CC failure occurs after CC boots.” Thus requirement definition is proved to be correct and to have no lacks.

5 System Design

5.1 Block Definition Diagram

Block definition diagram defines system architecture. Block definition diagram consists of block and path. The way of definition and example about the elements is shown below.

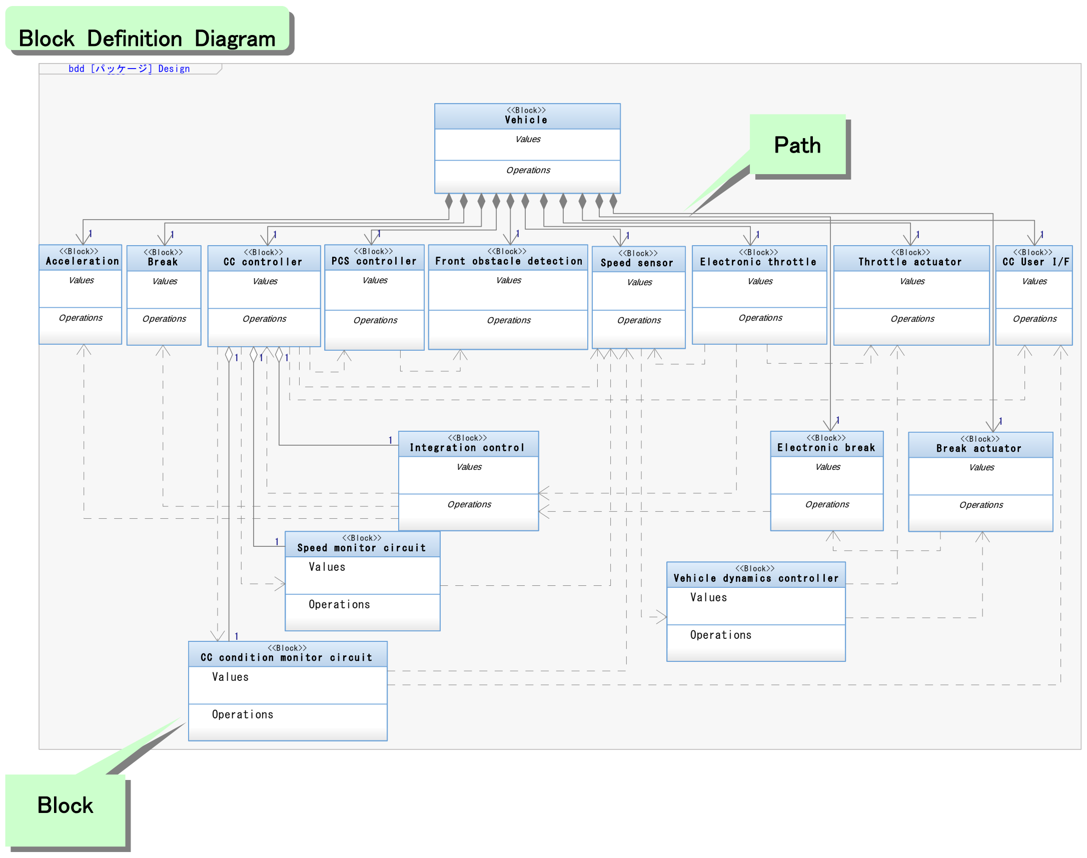

Block Definition Diagram
bdd [パッケージ] Design
Path
Block
<<Block>> Vehicle
<<Block>> Acceleration
<<Block>> Break
<<Block>> CC controller
<<Block>> PCS controller
<<Block>> Front obstacle detection
<<Block>> Speed sensor
<<Block>> Electronic throttle
<<Block>> Throttle actuator
<<Block>> CC User I/F
<<Block>> Integration control
<<Block>> Electronic break
<<Block>> Break actuator
<<Block>> Speed monitor circuit
<<Block>> Vehicle dynamics controller
<<Block>> CC condition monitor circuit
Values
Operations

Figure 5-1 Example of a Block Definition Diagram

（Block）

A block defines system configuration element. The way of defining blocks is shown below.

- Extract system configuration elements from specifications and models of former model, and define them as blocks.
- In case where D-Case sub-goal "When ▼, control which keeps ● value in tolerance level can be performed by ■ block even when a failure occurs at ▲ block." is derived from D-Case strategy describing provisions for causes deriving

threats inhibiting dependability “Divide ● system's safety for every provision.”, extract system elements from the description corresponding to “by ■ block” and define them as blocks.

- If D-Case is solved by each system component based on “split by architecture pattern”, extract system configuration elements from them, and define these elements as blocks.

* Define compartments or properties if needed.

(Path)

Define meaningful relationships between multiple blocks as dependency, part association (composition), shared association (composition), reference association, generalization, and etc.

* Define path ends, multiplicity and etc, if needed.

(Association with D-Case)

If D-Case contains goal nodes associated to blocks defined in block diagrams, traceability should be kept by associating the blocks to corresponding goals by using context nodes. At this time, the block ID is written in “Desc” column or “Attachment” column of the context node. D-Case data collaboration function on SW development environment [1] is available.

(Correspondence check between D-Case and design specifications)

Avoid lacks and mistakes in system design by checking the consistency between the contents of requirements defined in D-Case goal nodes and contents of block definition diagrams.
Avoid lacks in system design by checking if each of the goals divided by D-Case is associated to one or more blocks.

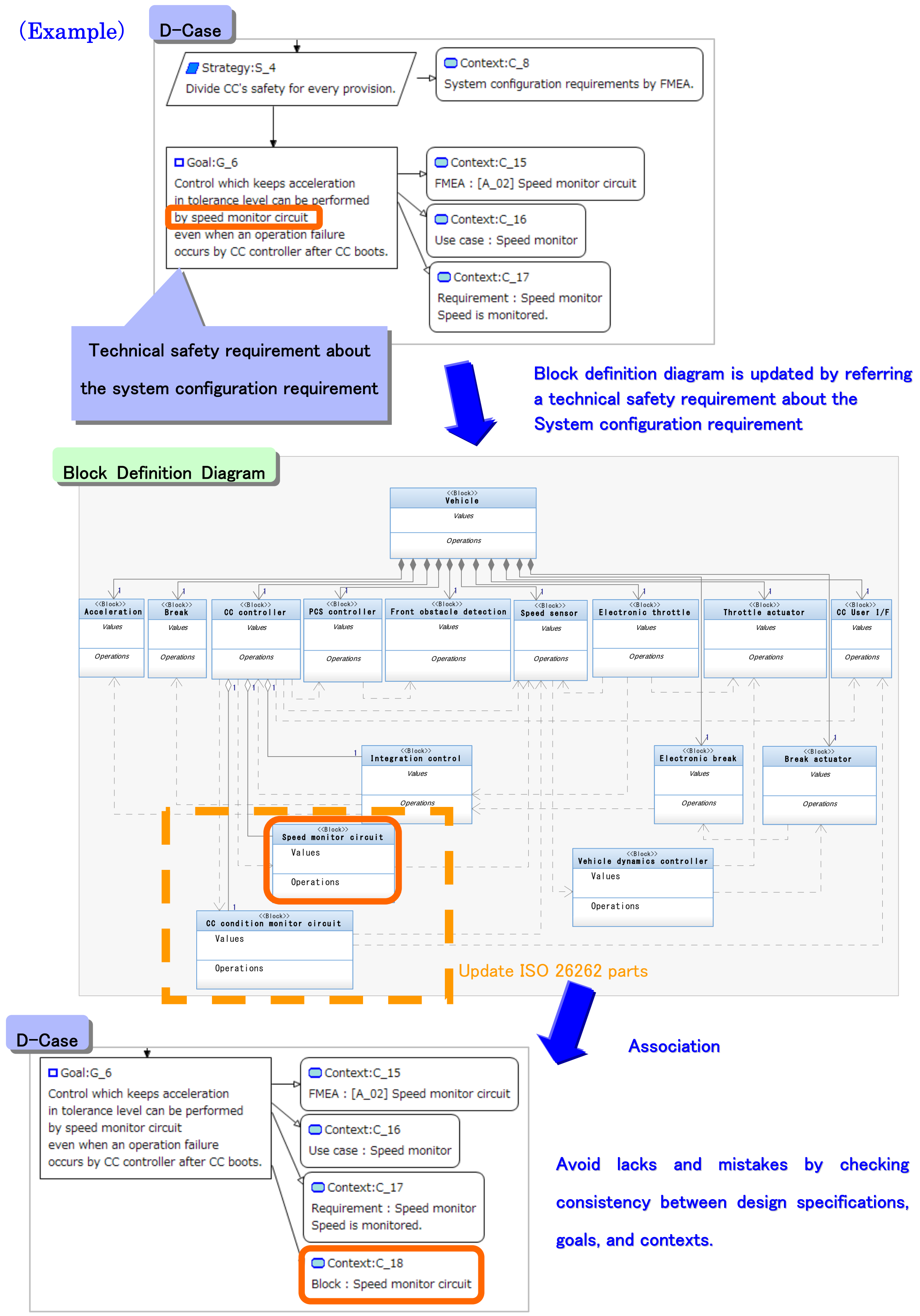

(Example)
D-Case
Strategy:S_4
Divide CC's safety for every provision.
Context:C_8
System configuration requirements by FMEA.
Goal:G_6
Control which keeps acceleration in tolerance level can be performed by speed monitor circuit even when an operation failure occurs by CC controller after CC boots.
Context:C_15
FMEA : [A_02] Speed monitor circuit
Context:C_16
Use case : Speed monitor
Context:C_17
Requirement : Speed monitor Speed is monitored.
Technical safety requirement about the system configuration requirement
Block definition diagram is updated by referring a technical safety requirement about the System configuration requirement
Block Definition Diagram
Vehicle
Acceleration
Break
CC controller
PCS controller
Front obstacle detection
Speed sensor
Electronic throttle
Throttle actuator
CC User I/F
Integration control
Electronic break
Break actuator
Speed monitor circuit
Vehicle dynamics controller
CC condition monitor circuit
Update ISO 26262 parts
Association
D-Case
Goal:G_6
Control which keeps acceleration in tolerance level can be performed by speed monitor circuit even when an operation failure occurs by CC controller after CC boots.
Context:C_15
FMEA : [A_02] Speed monitor circuit
Context:C_16
Use case : Speed monitor
Context:C_17
Requirement : Speed monitor Speed is monitored.
Context:C_18
Block : Speed monitor circuit
Avoid lacks and mistakes by checking consistency between design specifications, goals, and contexts.

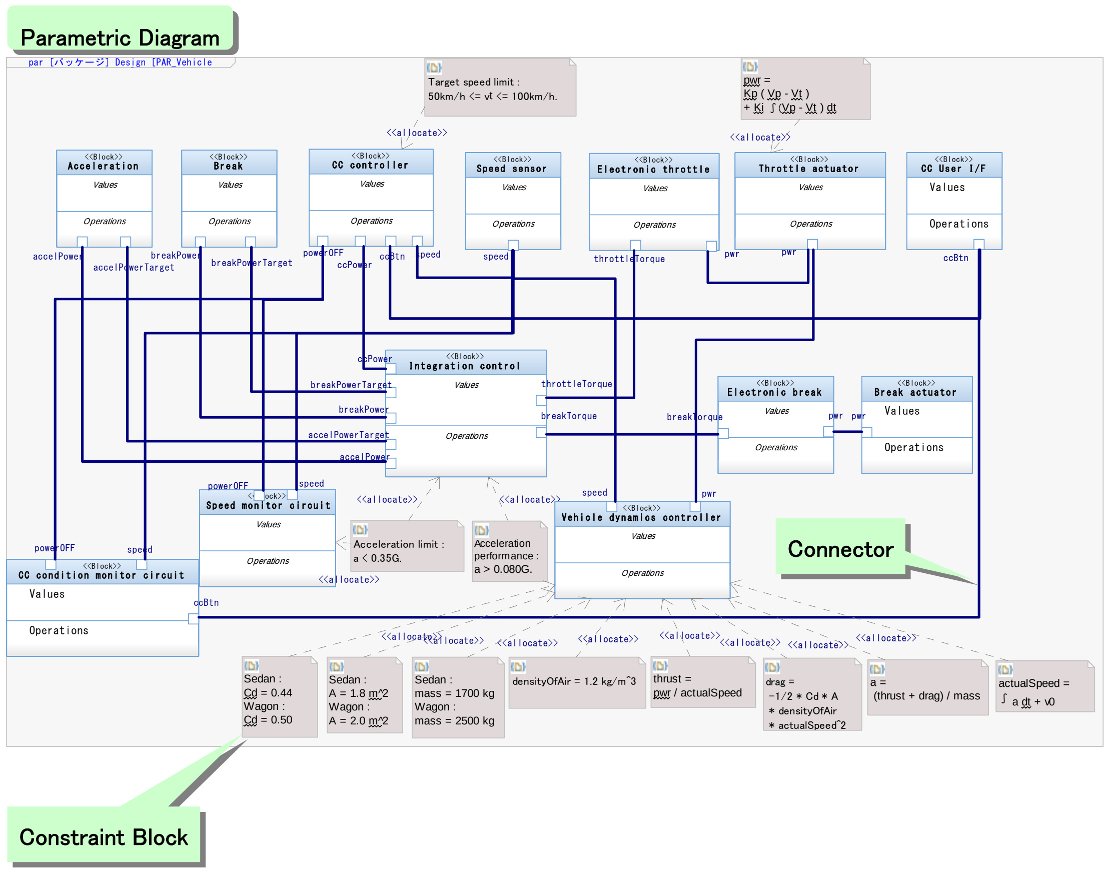
Figure 5-2 Example of Updating a Block Definition Diagram

An example of updating a block definition diagram is shown in Figure 5-2. In this example, a description about functional safety is written in D-Case goal as a technical safety requirement (TSR) derived from FMEA results “Control which keeps acceleration in tolerance level can be performed by speed monitor circuit even when an operation failure occurs by CC controller after CC boots.” This TSR is an example of the system configuration requirements. Based on this TSR, a block which is needed in point of functional safety “Speed monitor circuit” is extracted, and defined as a block. This block is associated as a D-Case context.

This block has compatibility with an involved D-Case goal “Control which keeps acceleration in tolerance level can be performed by speed monitor circuit even when an operation failure occurs by CC controller after CC boots.”, with an involved D-Case context “Requirement : Speed monitor: Speed is monitored.”, or with other involved D-Case nodes. Thus system design is proved to be correct and to have no lacks.

5.2 Parametric Diagram

Parametric diagram defines system constraints, related values and mathematical formulas. Parametric diagram consists of constraint block and connector. The way of definition and example about the elements is shown below.



Parametric Diagram
par [パッケージ] Design [PAR_Vehicle
Target speed limit : 50km/h <= vt <= 100km/h.
pwr = Kp (Vp - Vt) + Ki ∫ (Vp - Vt) dt
<<allocate>>
<<Block>> Acceleration
<<Block>> Break
<<Block>> CC controller
<<Block>> Speed sensor
<<Block>> Electronic throttle
<<Block>> Throttle actuator
<<Block>> CC User I/F
Values
Operations
accelPower
accelPowerTarget
breakPower
breakPowerTarget
powerOFF
ccPower
ccBtn
speed
throttleTorque
pwr
<<Block>> Integration control
breakTorque
<<Block>> Electronic break
<<Block>> Break actuator
<<Block>> Speed monitor circuit
<<Block>> CC condition monitor circuit
Acceleration limit : a < 0.35G.
Acceleration performance : a > 0.080G.
<<Block>> Vehicle dynamics controller
Connector
Sedan : Cd = 0.44 Wagon : Cd = 0.50
Sedan : A = 1.8 m^2 Wagon : A = 2.0 m^2
Sedan : mass = 1700 kg Wagon : mass = 2500 kg
densityOfAir = 1.2 kg/m^3
thrust = pwr / actualSpeed
drag = −1/2 * Cd * A * densityOfAir * actualSpeed^2
a = (thrust + drag) / mass
actualSpeed = ∫ a dt + v0
Constraint Block

Figure 5-3 Example of a Parametric Diagram

(Constraint Block)

A constraint block defines block describing system constraints, related values and mathematical formulas. The way of defining constraint blocks is shown below.

- Extract system constraints, related values and mathematical formulas from specifications or models in the former model, and define these as constraint blocks.
- In case where D-Case sub-goal “When ▼, control which keeps ● value in tolerance level can be performed by ■ block even when a failure occurs at ▲ block.” is derived from D-Case strategy describing provisions for causes deriving threats inhibiting dependability “Divide ● system's safety for every provision.”,

extract system constraints, related values and mathematical formulas from the description corresponding to "control which keeps ● value in tolerance level" and define them as constraint blocks.

- Consider system constraints, related values and mathematical formulas based on D-Case goal nodes, and define them as constraint blocks.

(Connector)

Connect the associated constraint blocks using connectors by following the block structure designed in block definition diagrams.

(Association with D-Case)

If D-Case contains goal nodes associated with constraint blocks defined in the parametric diagram, traceability should be kept by associating these goals with corresponding blocks using D-Case context nodes. At this time, the constraint block ID is written in "Desc" column or "Attachment" column of the context node. D-Case data collaboration function on SW development environment [1] is available.

(Correspondence check between D-Case and design specifications)

Avoid lacks and mistakes in system design by checking the consistency between the contents of requirements defined in D-Case goal nodes and contents of parametric diagrams.

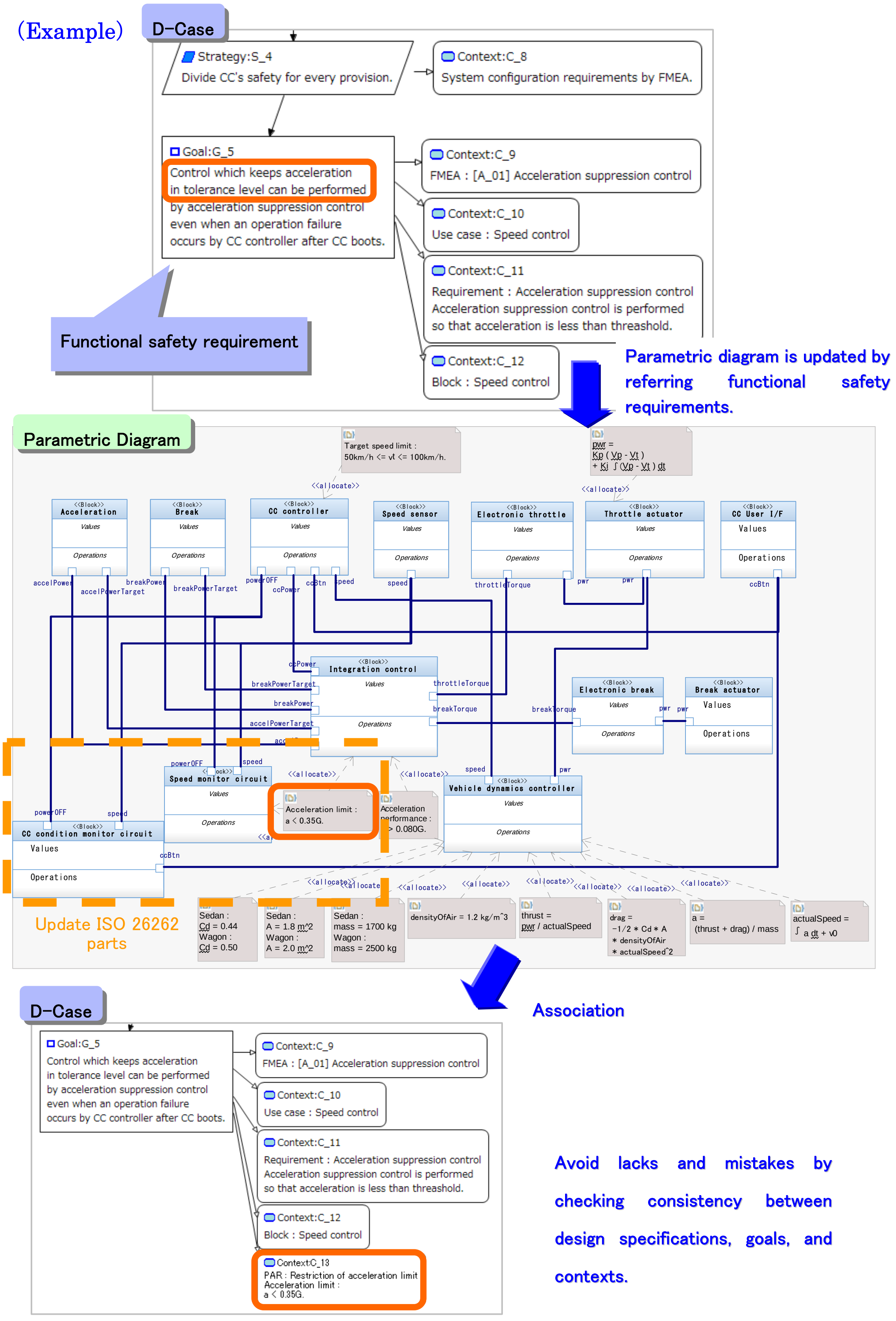

(Example)
D-Case
Strategy:S_4
Divide CC's safety for every provision.
Context:C_8
System configuration requirements by FMEA.
Goal:G_5
Control which keeps acceleration in tolerance level can be performed by acceleration suppression control even when an operation failure occurs by CC controller after CC boots.
Context:C_9
FMEA : [A_01] Acceleration suppression control
Context:C_10
Use case : Speed control
Context:C_11
Requirement : Acceleration suppression control Acceleration suppression control is performed so that acceleration is less than threashold.
Context:C_12
Block : Speed control
Functional safety requirement
Parametric diagram is updated by referring functional safety requirements.
Parametric Diagram
Target speed limit : 50km/h <= vt <= 100km/h.
pwr = Kp (Vp - Vt) + Ki ∫ (Vp - Vt) dt
<<allocate>>
Acceleration
Break
CC controller
Speed sensor
Electronic throttle
Throttle actuator
CC User I/F
Integration control
Electronic break
Break actuator
Speed monitor circuit
Vehicle dynamics controller
CC condition monitor circuit
Values
Operations
accelPower
accelPowerTarget
breakPower
breakPowerTarget
powerOFF
ccPower
ccBtn
speed
throttleTorque
breakTorque
pwr
Acceleration limit : a < 0.35G.
Acceleration performance : > 0.080G.
Sedan : Cd = 0.44 Wagon : Cd = 0.50
Sedan : A = 1.8 m^2 Wagon : A = 2.0 m^2
Sedan : mass = 1700 kg Wagon : mass = 2500 kg
densityOfAir = 1.2 kg/m^3
thrust = pwr / actualSpeed
drag = −1/2 ∗ Cd ∗ A ∗ densityOfAir ∗ actualSpeed^2
a = (thrust + drag) / mass
actualSpeed = ∫ a dt + v0
Update ISO 26262 parts
Association
D-Case
Goal:G_5
Control which keeps acceleration in tolerance level can be performed by acceleration suppression control even when an operation failure occurs by CC controller after CC boots.
Context:C_9
FMEA : [A_01] Acceleration suppression control
Context:C_10
Use case : Speed control
Context:C_11
Requirement : Acceleration suppression control Acceleration suppression control is performed so that acceleration is less than threashold.
Context:C_12
Block : Speed control
Context:C_13
PAR : Restriction of acceleration limit Acceleration limit : a < 0.35G.
Avoid lacks and mistakes by checking consistency between design specifications, goals, and contexts.

Figure 5-4 Example of Updating a Parametric Diagram

An example of updating a parametric diagram is shown in Figure 5-4. In this example, a description about functional safety is written in D-Case goal as a functional safety requirement (FSR) derived from hazard analysis “Control which keeps acceleration in tolerance level can be performed by acceleration suppression control even when an operation failure occurs by CC controller after CC boots.” Based on this FSR, a constraint block which is needed in point of functional safety is refined as “Acceleration limit: a < 0.35G.”, and defined as a constraint block. This constraint block is associated as a D-Case context.

This constraint block has compatibility with an involved D-Case goal “Control which keeps acceleration in tolerance level can be performed by acceleration suppression control even when an operation failure occurs by CC controller after CC boots.”, with an involved D-Case context “Requirement : Acceleration suppression control: Acceleration suppression control is performed so that acceleration is less than threshold.”, or with other involved D-Case nodes. Thus system design is proved to be correct and to have no lacks.

5.3 Internal Block Diagram

Internal block diagram defines internal structure of blocks clarified by block definition diagram. Internal block diagram consists of block and connector. The way of definition and example about the elements is shown below.

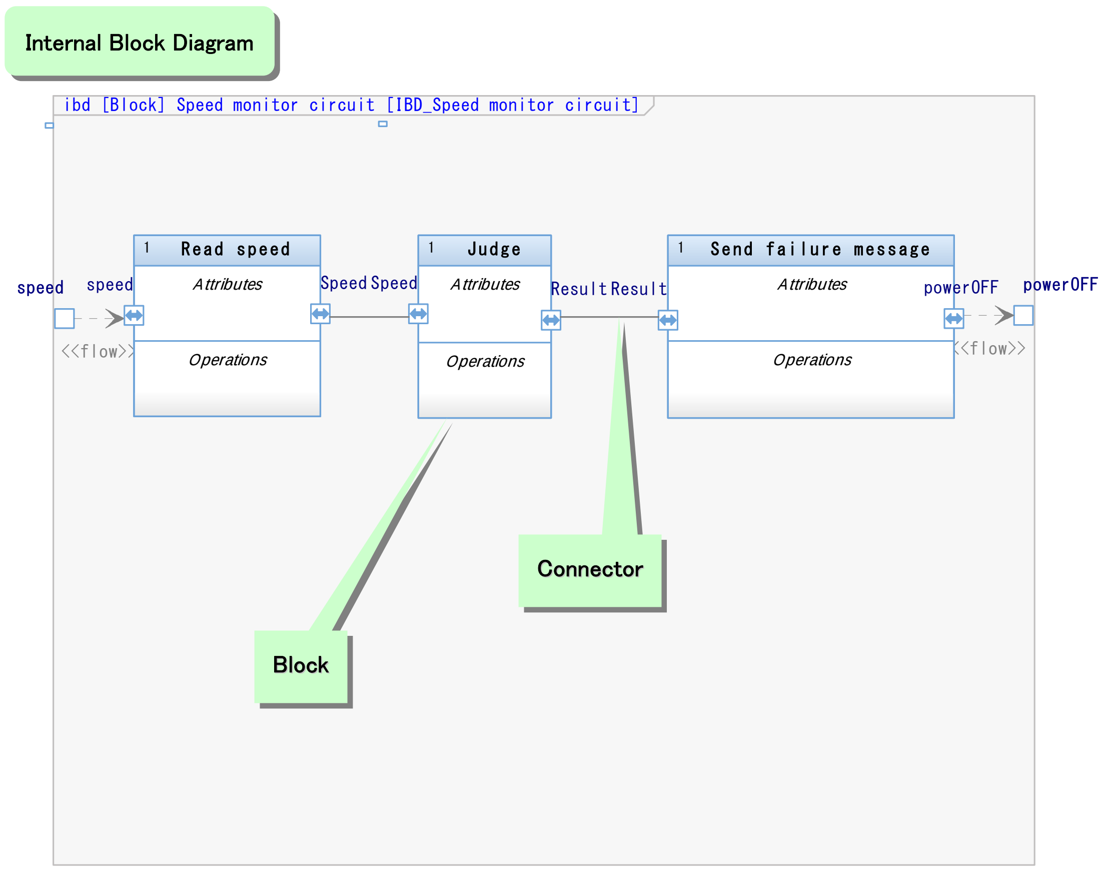


Figure 5-5 Example of an Internal Block Diagram

(Block)

A block defines system element of internal structure. The way of defining blocks is shown below.

- Extract blocks through detailing of internal structure from specifications and models of former model based on the blocks defined in the block definition diagram or constraint blocks defined in the parametric diagram.
- Define blocks by extracting system configuration elements from D-Case, similar to the block definition diagram shown in section 5.1.

（Connector）

Connect the associated blocks using connectors, based on the block structure designed in block definition diagrams or constraint blocks defined in the parametric diagram.

* The path defined in the block definition diagram, the connector defined in the parametric diagram, and the connector defined in the internal block diagram should be compatible.

（Association with D-Case）

Traceability should be kept if needed, by associating the blocks defined in internal block diagrams with corresponding goals by using D-Case context nodes. At this time, the block ID is written in “Desc” column or “Attachment” column of the context node. D-Case data collaboration function on SW development environment [1] is available.

（Correspondence check between D-Case and design specifications）

Avoid lacks and mistakes in system design by checking the consistency between the contents of requirements defined in D-Case goal nodes and contents of internal block diagrams.

(Example)

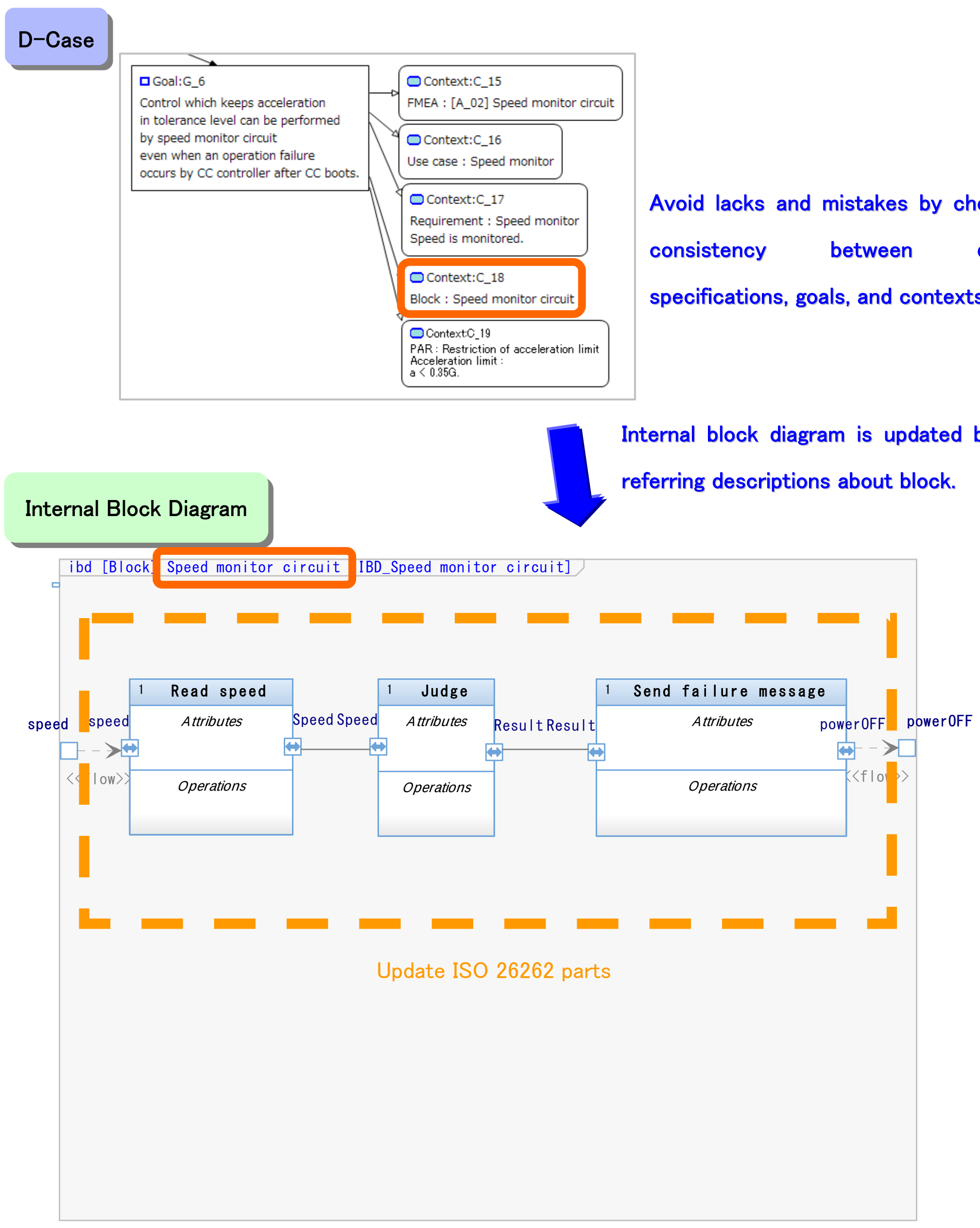

D-Case
Goal:G_6
Control which keeps acceleration in tolerance level can be performed by speed monitor circuit even when an operation failure occurs by CC controller after CC boots.
Context:C_15
FMEA : [A_02] Speed monitor circuit
Context:C_16
Use case : Speed monitor
Context:C_17
Requirement : Speed monitor
Speed is monitored.
Context:C_18
Block : Speed monitor circuit
Context:C_19
PAR : Restriction of acceleration limit
Acceleration limit :
a < 0.35G.
Avoid lacks and mistakes by checking consistency between design specifications, goals, and contexts.
Internal block diagram is updated by referring descriptions about block.
Internal Block Diagram
ibd [Block] Speed monitor circuit [IBD_Speed monitor circuit]
1 Read speed
1 Judge
1 Send failure message
Attributes
Operations
speed
speed
Speed
Speed
Result
Result
powerOFF
powerOFF
<<flow>>
<<flow>>
Update ISO 26262 parts

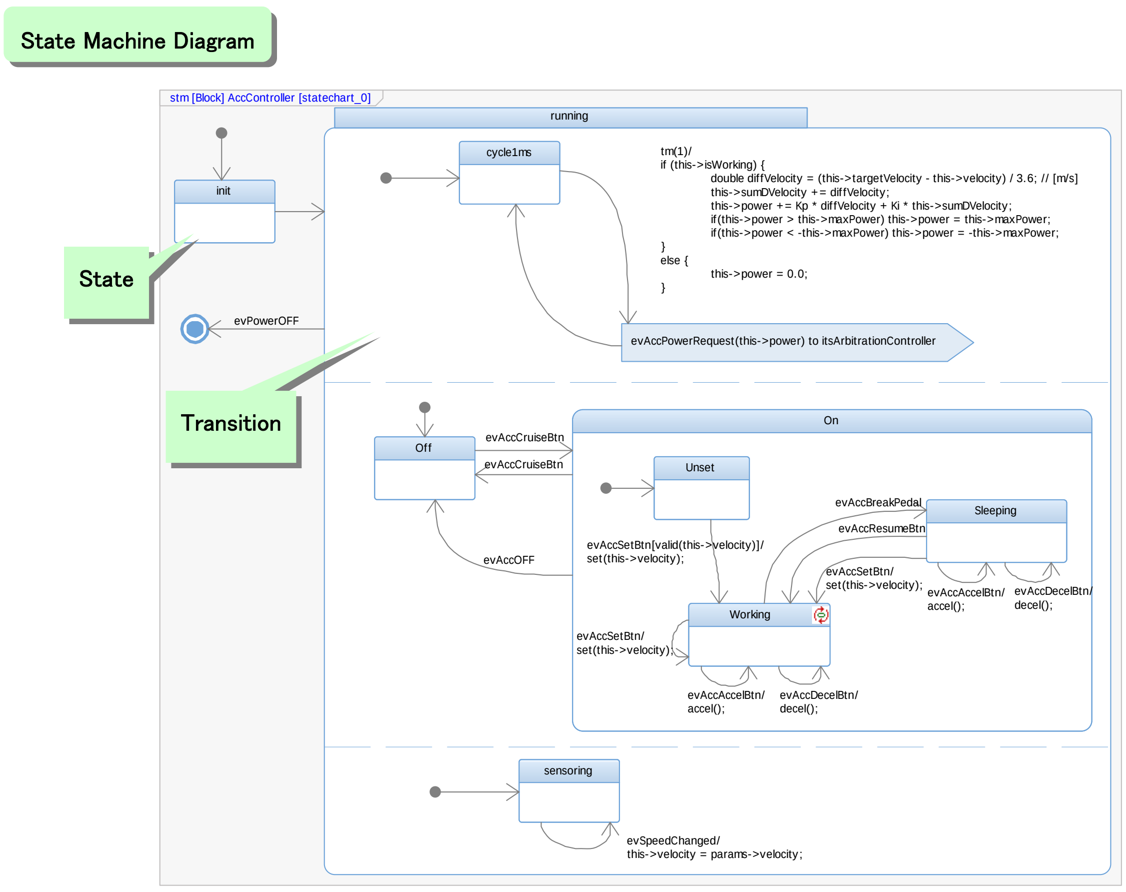
Figure 5-6 Example of Updating an Internal Block Diagram

An example of updating an internal block diagram is shown in Figure 5-6. In this example, a description about the block is written in D-Case context as “Block : Speed monitor circuit”. An internal block diagram is updated by clarifying the internal structure of the block.

This internal block diagram has compatibility with an involved D-Case goal “Control which keeps acceleration in tolerance level can be performed by speed monitor circuit even when an operation failure occurs by CC controller after CC boots.”, with an involved D-Case context “Requirement : Speed monitor : Speed is monitored.”, or with other involved D-Case nodes. Thus system design is proved to be correct and to have no lacks.

5.4 State Machine Diagram

State machine diagram defines dynamical behavior of a block. State machine diagram consists of state and transition. The way of definition and example about the elements is shown below.



```
tm(1)/
if (this->isWorking) {
            double diffVelocity = (this->targetVelocity - this->velocity) / 3.6; // [m/s]
            this->sumDVelocity += diffVelocity;
            this->power += Kp * diffVelocity + Ki * this->sumDVelocity;
            if(this->power > this->maxPower) this->power = this->maxPower;
            if(this->power < -this->maxPower) this->power = -this->maxPower;
}
else {
            this->power = 0.0;
}
```

Figure 5-7 Example of a State Machine Diagram

(State)

A state defines a particular status for certain time. The way of defining states is shown below.

- Define state to a block in block definition diagram, if the block behaves in particular status for certain time.
- If specifications or models for former model contain any state definition, these can be reused.

* Add start status or end status, if needed.

(Transition)

Define the following to each of the state; trigger for a state transition to different state, guard conditions to decide whether the state transition is valid, action to execute when the state transition occurs and continuous activities during particular state.
If specifications or models for former model contain any state transition, these can be reused.

* Organize the state transitions by creating state transition table, if needed.

(Association with D-Case)

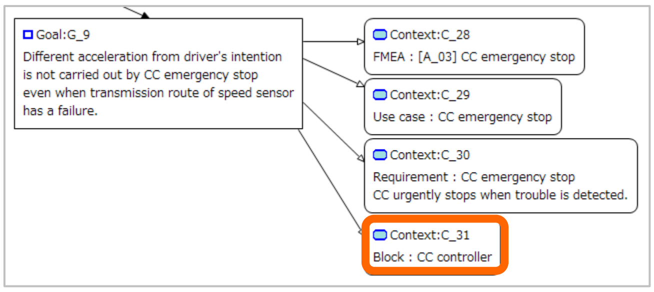
Traceability should be kept if needed, by associating states or transitions defined in state machine diagrams with corresponding goals by using D-Case context nodes. At this time, the state or transition is written in “Desc” column or “Attachment” column of the context node. D-Case data collaboration function on SW development environment [1] is available.

(Correspondence check between D-Case and design specifications)

Lacks and mistakes in system design should be avoided by checking the consistency between requirement contents defined in D-Case goal nodes and contents of state machine diagrams.

(Example)

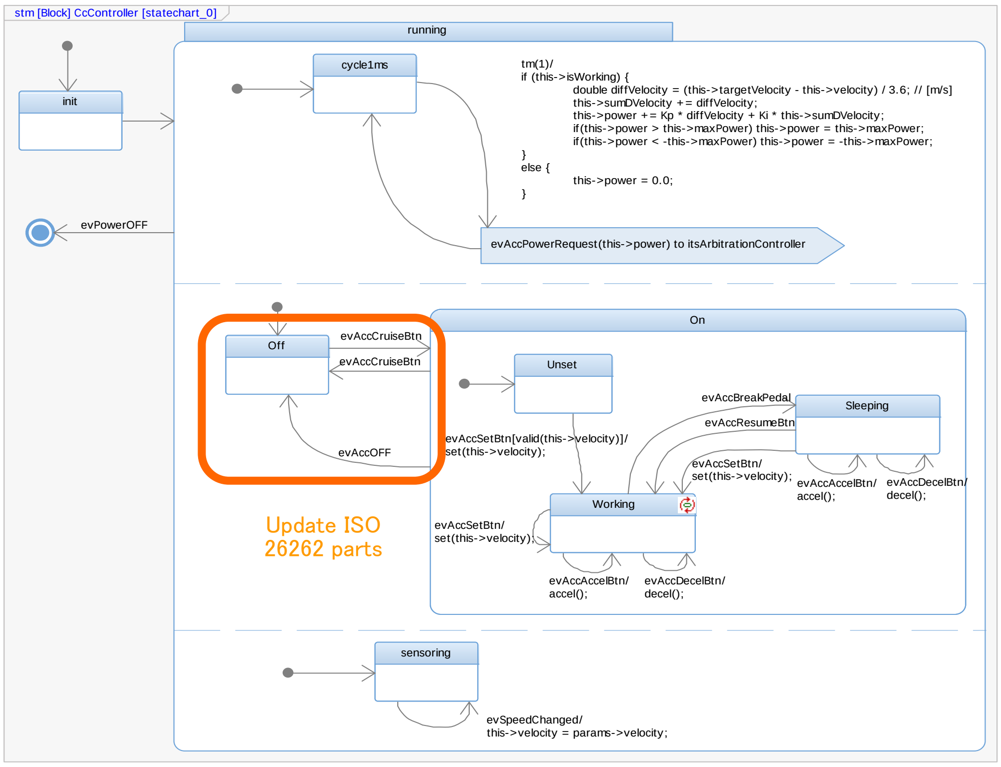
D-Case


Goal:G_9
Different acceleration from driver's intention is not carried out by CC emergency stop even when transmission route of speed sensor has a failure.
Context:C_28
FMEA : [A_03] CC emergency stop
Context:C_29
Use case : CC emergency stop
Context:C_30
Requirement : CC emergency stop
CC urgently stops when trouble is detected.
Context:C_31
Block : CC controller

Avoid lacks and mistakes by checking consistency between design specifications, goals, and contexts.

State machine diagram is updated by referring descriptions about block states.

State Machine Diagram



```
tm(1)/
if (this->isWorking) {
        double diffVelocity = (this->targetVelocity - this->velocity) / 3.6; // [m/s]
        this->sumDVelocity += diffVelocity;
        this->power += Kp * diffVelocity + Ki * this->sumDVelocity;
        if(this->power > this->maxPower) this->power = this->maxPower;
        if(this->power < -this->maxPower) this->power = -this->maxPower;
}
else {
        this->power = 0.0;
}
```

```
evAccPowerRequest(this->power) to itsArbitrationController
```

Figure 5-8 Example of Updating a State Machine Diagram

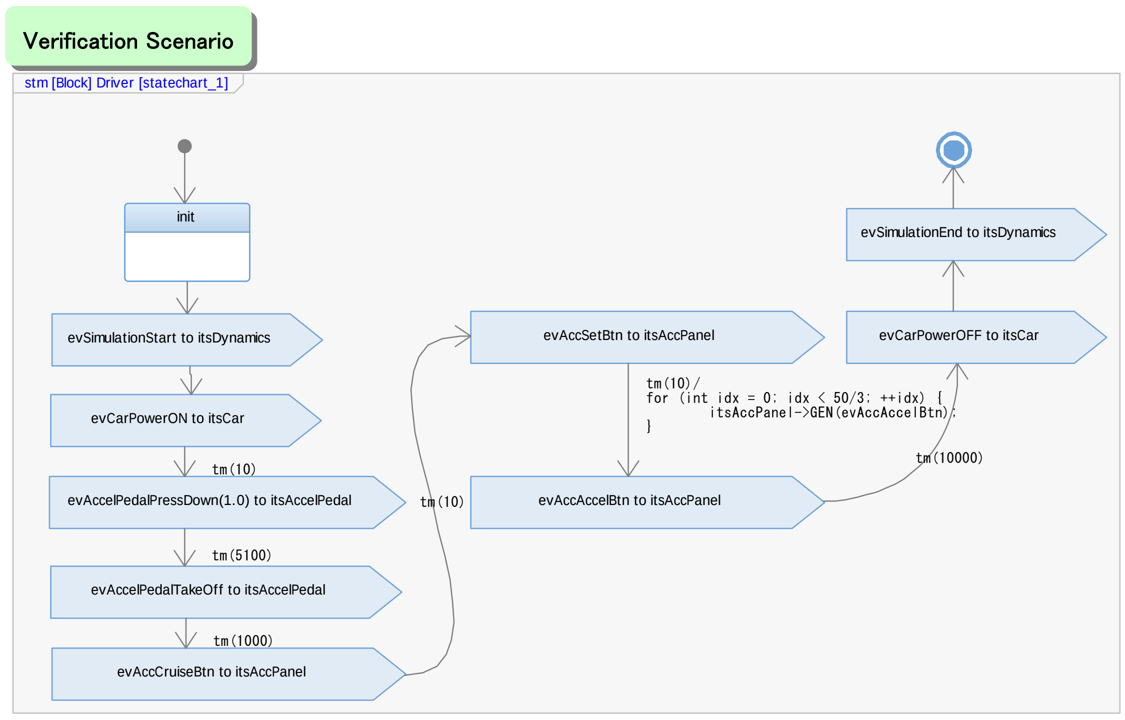
An example of updating a state machine diagram is shown in Figure 5-8. In this example, a description about a block is written in D-Case context as "Block : CC controller". The state machine diagram is updated by clarifying the dynamical behavior of the block.

This state machine diagram has compatibility with an involved D-Case goal “Different acceleration from driver's intention is not carried out by CC emergency stop even when transmission route of speed sensor has a failure.”, with an involved D-Case context “Requirement：CC emergency stop：CC urgently stops when trouble is detected.”, or with other involved D-Case nodes. Thus system design is proved to be correct and to have no lacks.

6 System Verification

6.1 Verification Scenario

Verification scenario defines the way of verification. In this guide, verification scenario is described in state machine diagram. The way of defining verification scenario is shown below.



Verification Scenario
stm [Block] Driver [statechart_1]
init
evSimulationStart to itsDynamics
evCarPowerON to itsCar
tm(10)
evAccelPedalPressDown(1.0) to itsAccelPedal
tm(5100)
evAccelPedalTakeOff to itsAccelPedal
tm(1000)
evAccCruiseBtn to itsAccPanel
tm(10)
evAccSetBtn to itsAccPanel
evAccAccelBtn to itsAccPanel
tm(10000)
evCarPowerOFF to itsCar
evSimulationEnd to itsDynamics

```
tm(10)/
for (int idx = 0; idx < 50/3; ++idx) {
          itsAccPanel->GEN(evAccAccelBtn);
}
```

Figure 6-1 Example of a Verification Scenario

(Verification Scenario)

A verification scenario defines verification procedure, and is described in state machine diagram and so on. The way of defining verification scenario is shown below.

- If specifications or models for the former model contain any verification scenarios related to the system verification, these scenarios can be reused,
- In case where D-Case sub-goal “When ▼, ● system is safe for hazard X.” is derived from D-Case strategy describing generation scene of threat inhibiting dependability “Argue about ● system's safety for every scene of threat.”, extract verification conditions from the description corresponding to “When ▼” and express these scenarios in state machine diagrams.

(System Verification)

Create source codes from state machine diagrams, and execute model simulation to verify the system.

(Association with D-Case)

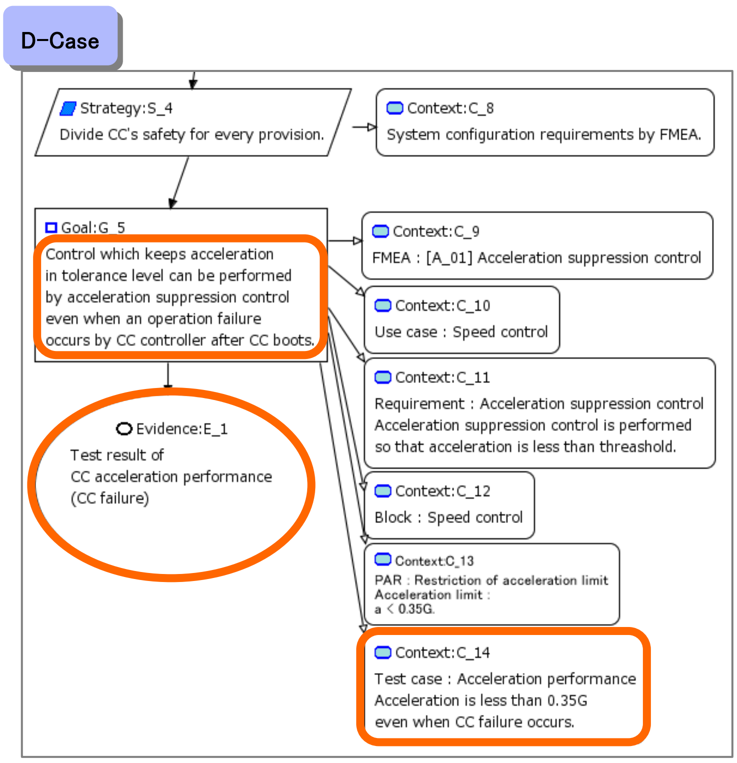
Traceability should be kept by associating system verification results with goals, by using D-Case context nodes or evidence nodes. At this time, the verification scenario ID is written in “Desc” column or “Attachment” column of the context node. D-Case OSLC add-on [2] is available.

(Correspondence check between D-Case and requirements)

Avoid lacks and mistakes in system verification, by checking consistency between requirement contents stated in D-Case goal nodes with the contents of system verification.

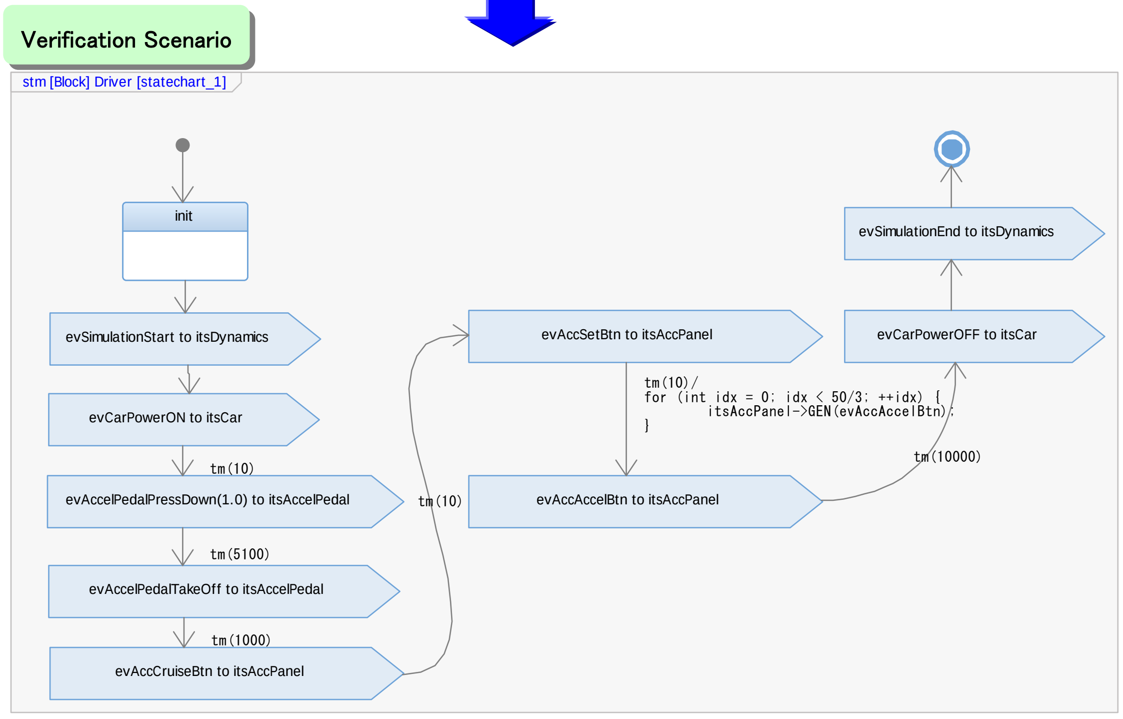
Avoid lacks in system verification by checking if each of the goals divided by D-Case is associated to one or more system verification results.

(Example)


D-Case
Strategy:S_4
Divide CC's safety for every provision.
Context:C_8
System configuration requirements by FMEA.
Goal:G_5
Control which keeps acceleration in tolerance level can be performed by acceleration suppression control even when an operation failure occurs by CC controller after CC boots.
Context:C_9
FMEA : [A_01] Acceleration suppression control
Context:C_10
Use case : Speed control
Context:C_11
Requirement : Acceleration suppression control
Acceleration suppression control is performed so that acceleration is less than threashold.
Evidence:E_1
Test result of CC acceleration performance (CC failure)
Context:C_12
Block : Speed control
Context:C_13
PAR : Restriction of acceleration limit
Acceleration limit :
a < 0.35G.
Context:C_14
Test case : Acceleration performance
Acceleration is less than 0.35G
even when CC failure occurs.

Avoid lacks and mistakes by checking consistency between verification results, goals, and contexts.

Verification scenario is created.


Verification Scenario
stm [Block] Driver [statechart_1]
init
evSimulationStart to itsDynamics
evCarPowerON to itsCar
tm(10)
evAccelPedalPressDown(1.0) to itsAccelPedal
tm(5100)
evAccelPedalTakeOff to itsAccelPedal
tm(1000)
evAccCruiseBtn to itsAccPanel
tm(10)
evAccSetBtn to itsAccPanel
evAccAccelBtn to itsAccPanel
tm(10000)
evCarPowerOFF to itsCar
evSimulationEnd to itsDynamics

```
tm(10)/
for (int idx = 0; idx < 50/3; ++idx) {
        itsAccPanel->GEN(evAccAccelBtn);
}
```

Figure 6-2 Example of Updates of Verification Scenario

An example of updates of verification scenario is shown in Figure 6-2. In this example, a description about verification scenario is written in D-Case goal as a functional safety requirement (FSR) derived from hazard analysis “Control which keeps acceleration in tolerance level can be performed by acceleration suppression control even when an operation failure occurs by CC controller after CC boots.” Based on this FSR, “Test case: Acceleration performance Acceleration is less than 0.35G even when CC failure occurs.” is defined and is associated as a D-Case context. Based on this verification scenario, state machine diagrams which are needed in model simulation are described, and the verification is carried out.

The verification result is written as a D-Case evidence “Test result of CC acceleration performance (CC failure)”. This evidence has compatibility with an involved D-Case context “Requirement: Acceleration suppression control: Acceleration suppression control is performed so that acceleration is less than threashold.”, or with other involved D-Case nodes. Thus system verification is proved to be correct and to have no lacks.

Reference

[1] D-Case data collaboration function on SW development environment, http://www.dependable-os.net/tech/D-Case-OSLC/index.html

[2] D-Case OSLC add-on, http://www.dependable-os.net/tech/D-Case-OSLC/index.html