



IEA-P – DEPARTAMENTO DE PROJETOS
(PROJECT DEPARTMENT)

ARCML BEHAVIORS

Session 07

Prepared by Prof. Dr. Christopher Shneider Cerqueira



Review



Last Lectures Review

1. Systems Engineering Basics

- Stk – Lifecycle – CONOPs – Functions – Architecture – V&V
- Classical Representations

2. Path to SE Digitalization

- Meta(meta)models
- Language
- Methodologies

- OPM

• ARCML Diagram

• Block Diagram

• Class Diagram



Motivation

- ARCML defines the use of two “diagram styles” to create functional/component architectures
 - Trees/layers
- Differently from OPM, even though functions are somehow behaviors. The diagrams do not show the component behavior over time. It is implemented three ways:
 - Reactive Behavior (Statemachines) descriptions
 - Use case (Capabilities) descriptions
 - Sequence (Scenarios) descriptions




SEMANA		TEORIA	INDIVIDUAL	PESO	GRUPO	PESO
1	1	Estrutura e Filosofia do Curso	AI-01 - Resumo Cap 1 - HB INCOSE	10%		
<i>05-Aug</i>	1	O que é Engenharia de Sistemas? INCOSE				
	1	Elementos da Eng Sis.				
	1	Introdução aos diagrams clássicos.				
2		* (Viagem ao EUA)	AI-02 - Leitura/Resumo paper sobre representações clássicas.	10%		
<i>12-Aug</i>						
3		* (Viagem ao EUA)	AI-03 - Exercício sobre arquitetura e escrita de requisitos.	10%		
<i>19-Aug</i>						
4	1	Metodologias de MBSE e uso de modelos.	AI-04 - Resumo Artigo de Metodologias	10%		
<i>26-Aug</i>	1	Revisão de UML-SysML.				
	1	OPM				
	1	Arcadia				
5	1	OPM	AI-05 - Lista de exercícios	10%		
<i>02-Sep</i>	1					
	1					
	1					
6	1	Blocos e Classes	AI-06 - Lista de Exercícios	20%		
<i>09-Sep</i>	1					
	1	Máquina de Estados				
7	1	Casos de Uso	AI-07 - Lista de Exercícios	20%		
<i>16-Sep</i>	1					
	1	Sequência				
8	1	Integração dos pontos de vistas em um	AI-08 - Resumo sobre Ciclo de Vida de Modelos	10%	AI-08 - Descrição e Contorno do Problema.	100%
<i>23-Sep</i>	1	Associação dos artefatos de SE com modelos				
	1	Análise Operacional				
	1					
				100%		100%
SEM						
<i>30-Sep</i>						






Summary




Reactive Behavior
(Statemachines)



Use Cases (Capabilities)



Sequence (Scenarios)



Final Considerations



Reactive Behavior (Statemachines)



Reactive system definition

- A reactive system is a system that, **when switched on, is able to create desired effects in its environment by enabling, enforcing, or preventing events in the environment.**
- Properties:
 - **Continuous interaction** (nonterminating)
 - System will respond to **external stimuli**, and
 - **the response depends of its current state**



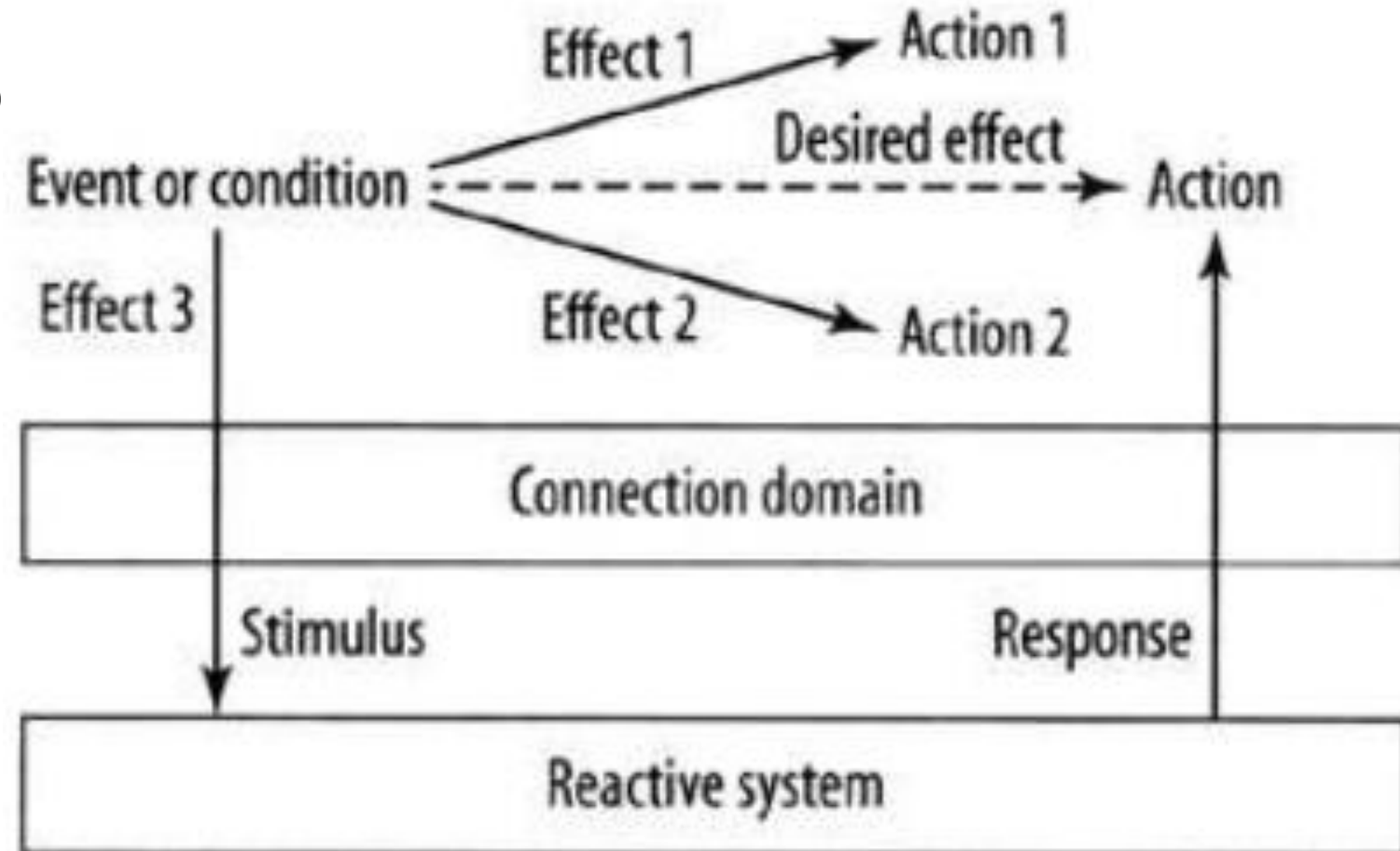
REACTIVE SYSTEMS EXAMPLES

- **REAL TIME SYSTEMS** – the answer depends on the time. Ex.: control software, elevator control, ATMs
- **SAFETY-CRITICAL SYSTEMS** – malfunctional behaviors can lead to lost of lives. Ex.: bio-physical systems, onboard computers (car/aircraft/spacecraft/ship)
- **EMBEDDED SYSTEMS** – implementation restrictions. Ex.: mobile, onboard software-firmware, IoT.
- Systems that manage critical infrastructures: air management, train, nuclear reactors, so on.



Cause and effect chains

- The function of a reactive system is to **respond to the occurrence of events or conditions in the environment by causing desirable changes in the environment.**





Events, conditions and actions

- **EVENT** – something that **happens** in the world.
 - **EXTERNAL EVENTS:** discrete change in the condition of the environment
 - **TEMPORAL EVENTS:** passage of a significant time to which the system is expected to respond
- **CONDITION** – **state of the world** that persists for some nonzero period of time.
- **ACTIONS** – events from the point of view of the initiator



Stimuli

- The stimulus of a system is an **event at the interface of the system caused by the environment.**
- External events occur somewhere in the environment; **stimuli occur at the interface of the system**

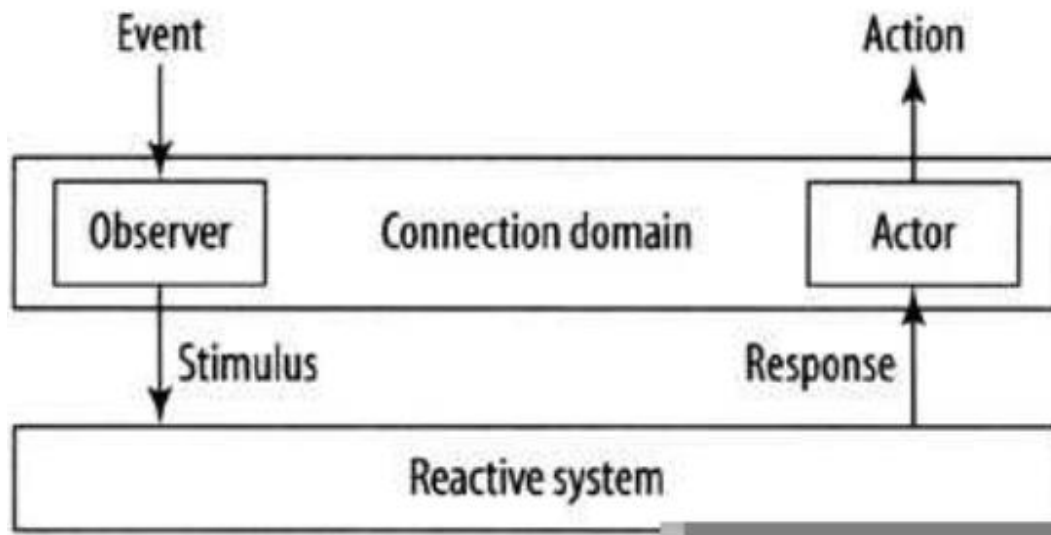


Response

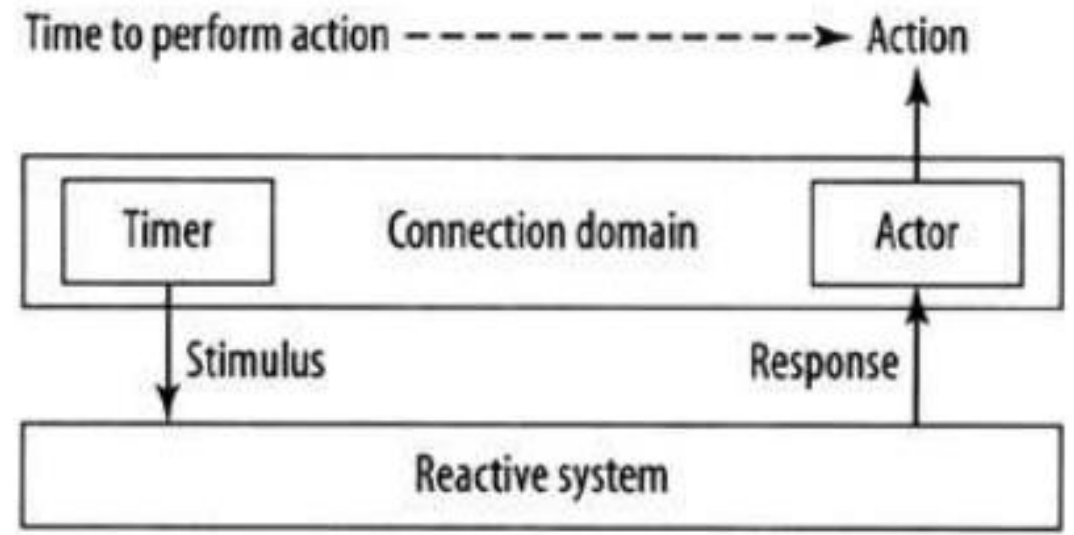
- The reactive system **responds to an external** or temporal event by updating its state or producing an output.
- An output response of a system is an **event at the interface** of the system, caused by the system.
- The desired effect of the stimulus may consist of **one or more desired actions**, to be caused by several responses.



Summary



Events-Stimuli



Temporal Events

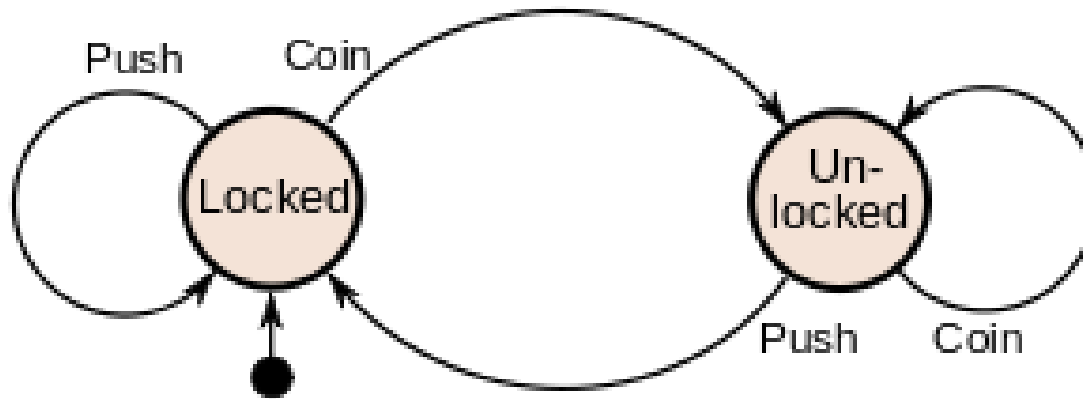


State Machines

- A finite-state machine (FSM) or finite-state automaton (FSA, plural: automata), finite automaton, or simply a state machine, is a mathematical model of computation.
- **It is an abstract machine that can be in exactly one of a finite number of states at any given time.**
- The FSM can **change from one state to another** in response to some external inputs; the change from one state to another is called a **transition**.
- An FSM is defined by a list of its states, its initial state, and the conditions for each transition.



Example



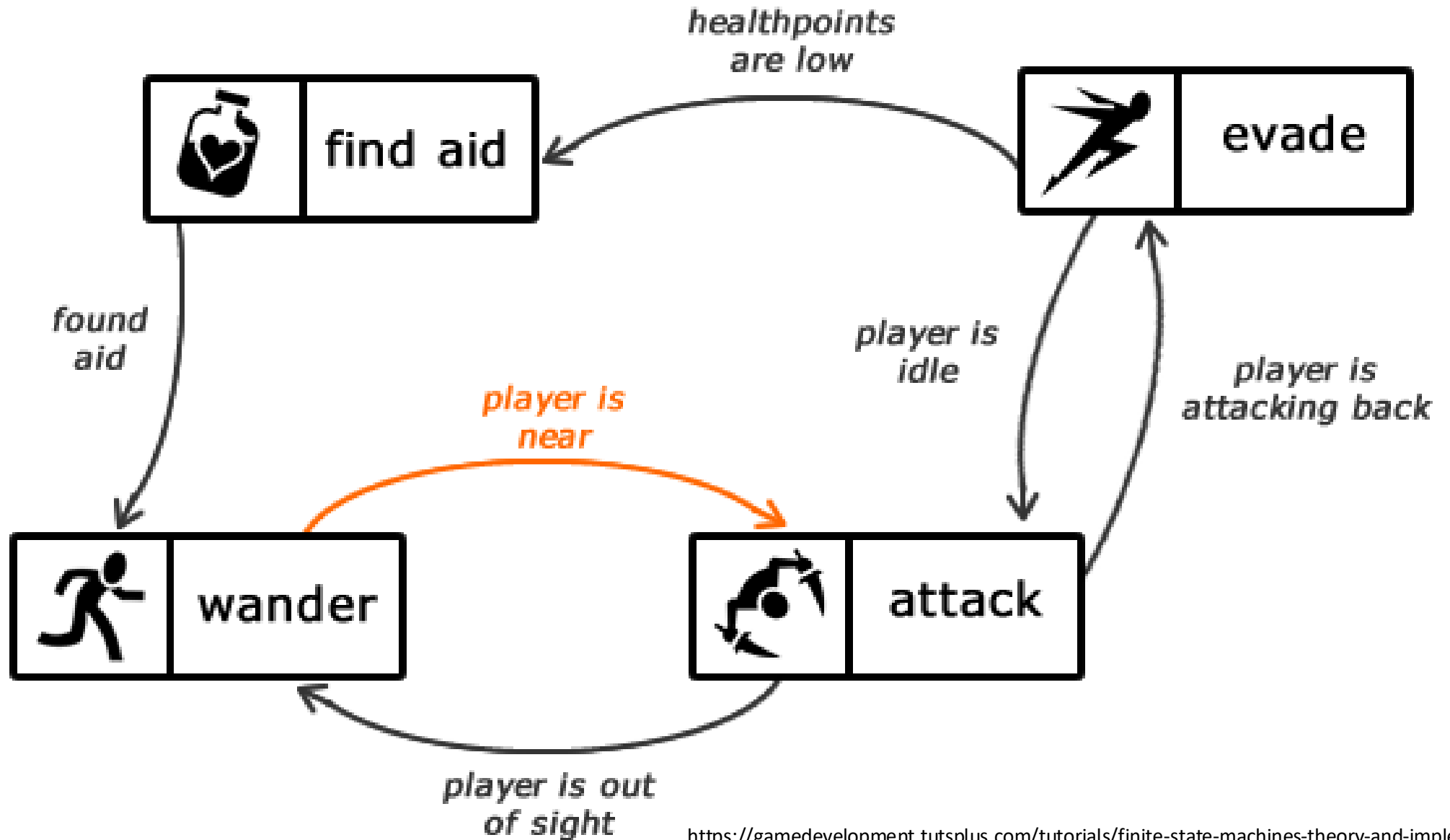
Current State	Input	Next State	Output
Locked	coin	Unlocked	Unlocks the turnstile so that the customer can push through.
	push	Locked	None
Unlocked	coin	Unlocked	None
	push	Locked	When the customer has pushed through, locks the turnstile.





A game example

FSM representing the brain of an enemy.



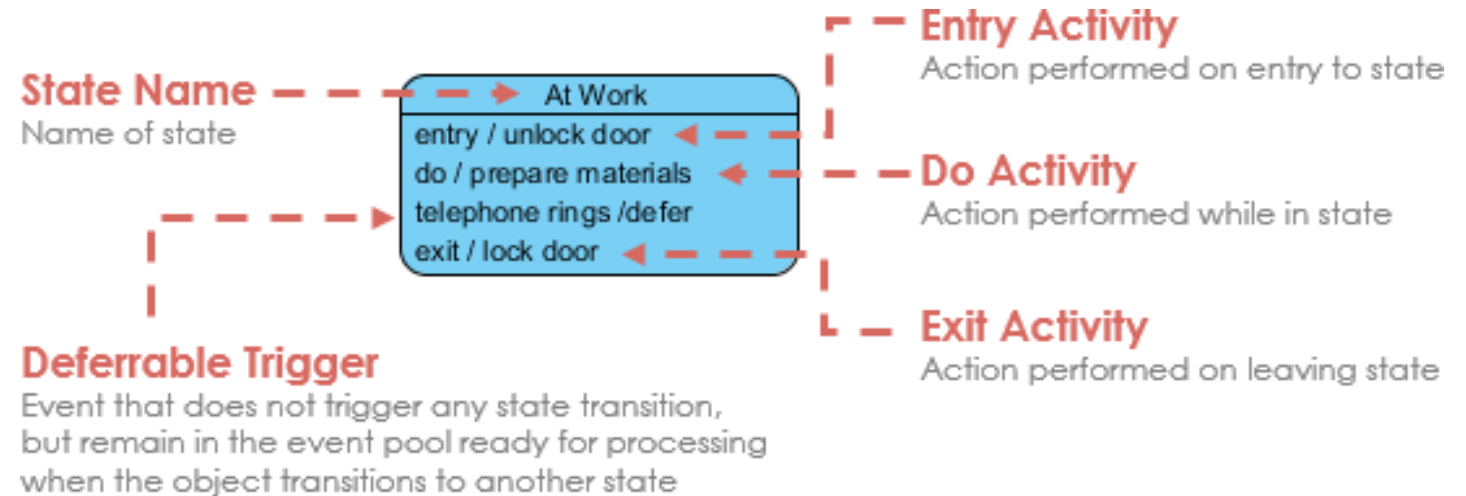
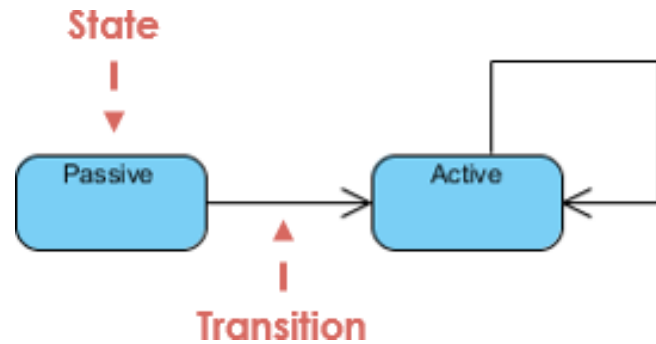


So

- State machines are typically used to **describe the state-dependent behavior of a block throughout its lifecycle**, which is defined in terms of its states and the transitions between them.
- The state machine defines **how the block's behavior changes as it transitions between different states** and while the block is in different states.
- State machines can be used to describe a wide range of **state-related behavior**, from the behavior of a simple lamp switch to the complex modes of an advanced aircraft



States

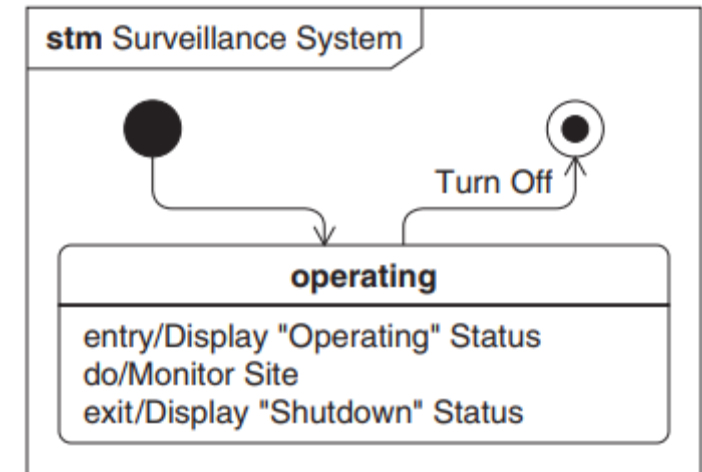


- A state is a constraint or a situation in the life cycle of an object, in which a constraint holds, the object executes an activity or waits for an event.
- There are several characteristics of states in general, regardless of their types:
 - State represent the conditions of objects at certain points in time.
 - A state is often associated with an abstraction of attribute values of an entity satisfying some condition(s).
 - An entity changes its state not only as a direct consequence of the current input, but it is also dependent on some past history of its inputs.



Specifying states

- **REGION:** describe the state-related behavior of the state machine.
- **STATE:** represents some significant condition in the life of a block, typically because it represents some change in how the block responds to events and what behaviors it performs.
 - Each state may contain **entry** and **exit** behaviors that are performed whenever the state is entered or exited, respectively. In addition, the state may contain a **do** behavior that executes once the entry behavior has completed





Transition

- Transition lines depict the movement from one state to another. Each transition line is labeled with the event that causes the transition.
- Understanding state transitions is part of system analysis and design
- Multiple transitions occur either when different events result in a state terminating or when there are guard conditions on the transitions
- A transition without an event and action is known as automatic transitions



Trigger

- A trigger identifies the possible stimuli that cause a transition to occur. SysML has four main kinds of triggering events.
 - A **signal event** indicates that a new asynchronous message corresponding to a signal has arrived.
 - A **signal event** may be accompanied by several arguments that can be used in the transition effect.
 - A **time event** indicates either that a given time interval has passed since the current state was entered (relative) or that a given instant in time has been reached (absolute).
 - A **change event** indicates that some condition has been satisfied (normally that some specific set of attribute values hold).
 - A **call event** indicates that an operation on the state machine's owning block has been requested. A call event may also be accompanied by several arguments



Guard

- The transition guard contains an **expression that must evaluate to true for the transition to occur**. The guard is specified using a constraint, which includes a textual expression to represent the guard condition.
- When an event satisfies a trigger, the guard on the transition is evaluated.
 - If the guard evaluates to true, the transition is triggered;
 - if the guard evaluates to false, then the event is consumed with no effect.
- Guards can test the state of the state machine using the operators in (state x) and not in (state x).



Effect

- The effect is a **behavior executed during the transition** from one state to another.
- The transition effect can be an arbitrarily complex behavior that may include send signal actions or operation calls used to interact with other blocks.
 - If the transition is triggered, **first the exit behavior of the current (source) state is executed**, then the **transition effect is executed**, and finally the **entry behavior of the target state is executed**.



Initialization and completion

- The initialization and completion of a region are described using an initial pseudostate and final state, respectively.
- An **initial pseudostate** is used to determine the initial state of a region.
- When the active state of a region is **the final state**, the region has completed, and no more transitions take place within it. Hence, a final state can have no outgoing transitions.
- The **terminate pseudostate** is always associated with the state of an entire state machine. If a terminate pseudostate is reached, then the behavior of the state machine terminates. A terminate pseudostate has the same effect as reaching the final states of all the state machine's regions.

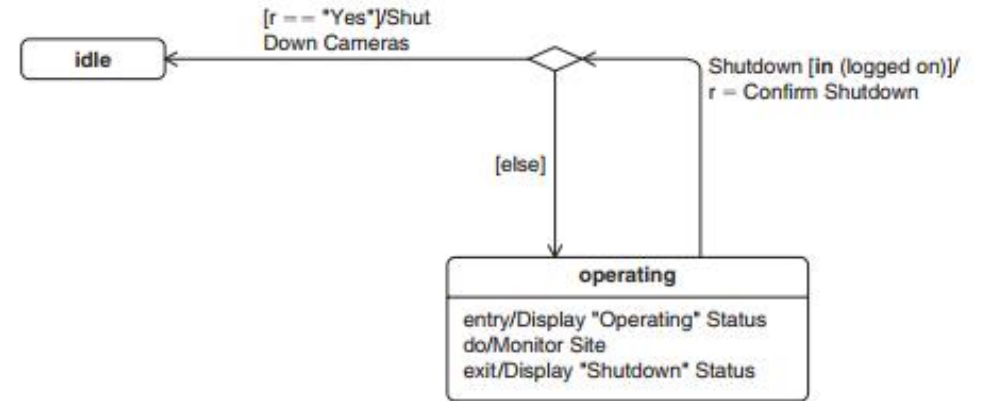
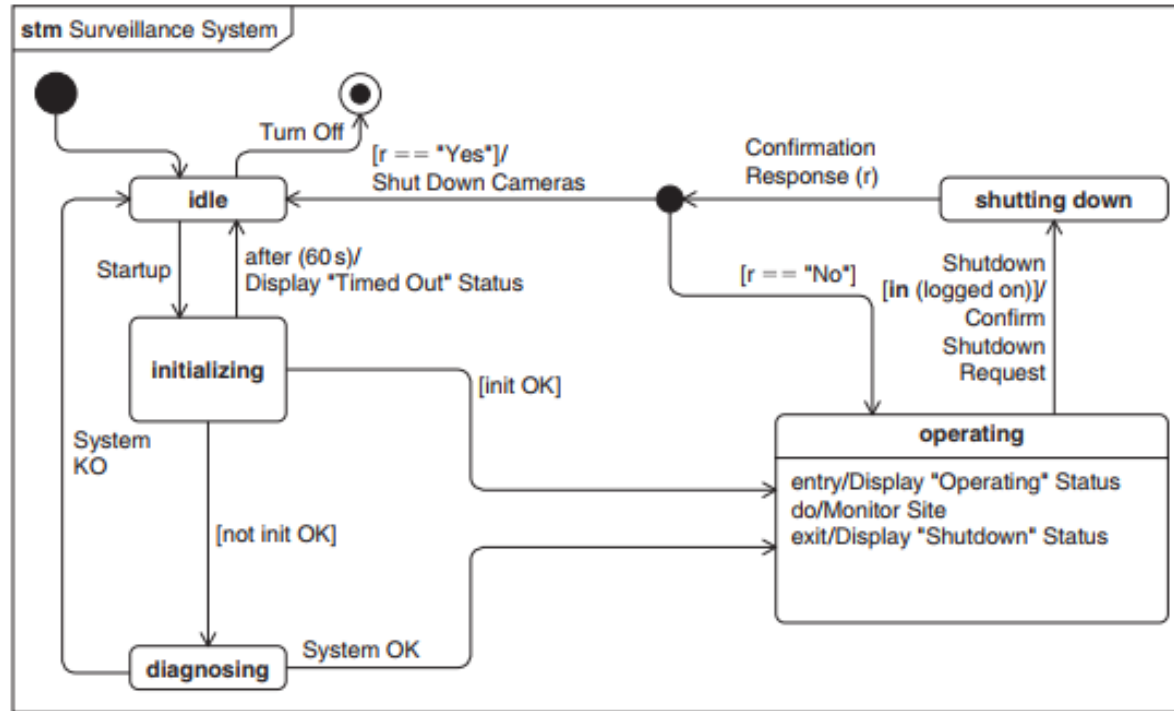


Routing transitions using pseudo states

- There are a variety of situations when a simple transition directly between two states is not sufficient to express the required semantics.
 - A **junction pseudostate** is used to construct a compound transition path between states. The compound transition allows more than one alternative transition path between states to be specified, although only one path can be taken in response to any single event.
 - The **choice pseudostate** also has multiple incoming transitions and outgoing transitions and, like the junction pseudostate, is part of a compound transition between states. The behavior of the choice pseudostate is distinct from that of a junction pseudostate in that the guards on its outgoing transitions are not evaluated until the choice pseudostate has been reached.



Routings



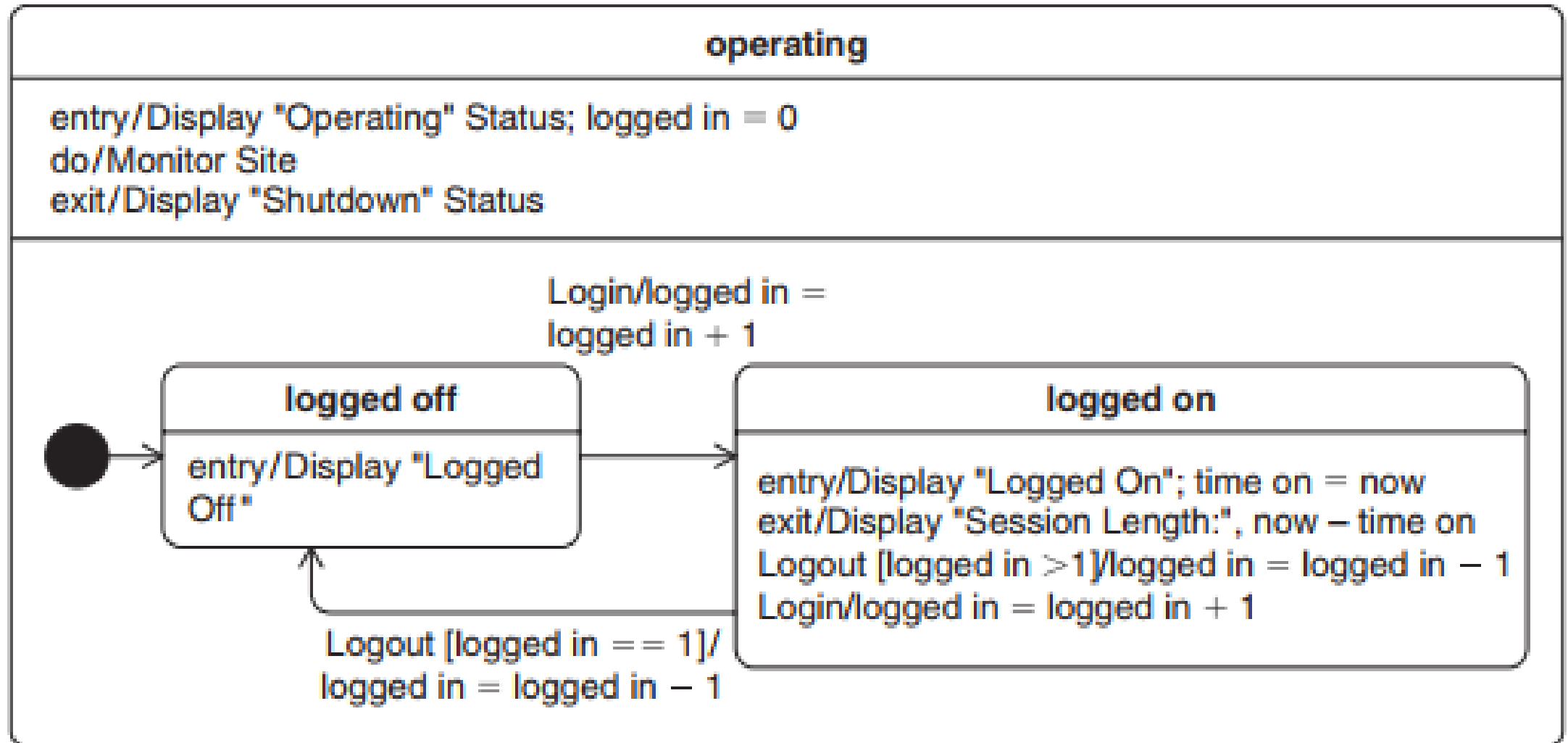


Composite states – single region

- **Arguably the most common situation is a composite state that has a single region.**
 - A region typically will contain **an initial pseudostate and a final state**, a set of pseudostates, and set of substates, which may themselves be composite states.
 - If the region has a final state, then a completion event is generated when that state is reached.
- A composite state may have many regions, which may each contain substates.



Composite states – single region (two substates)



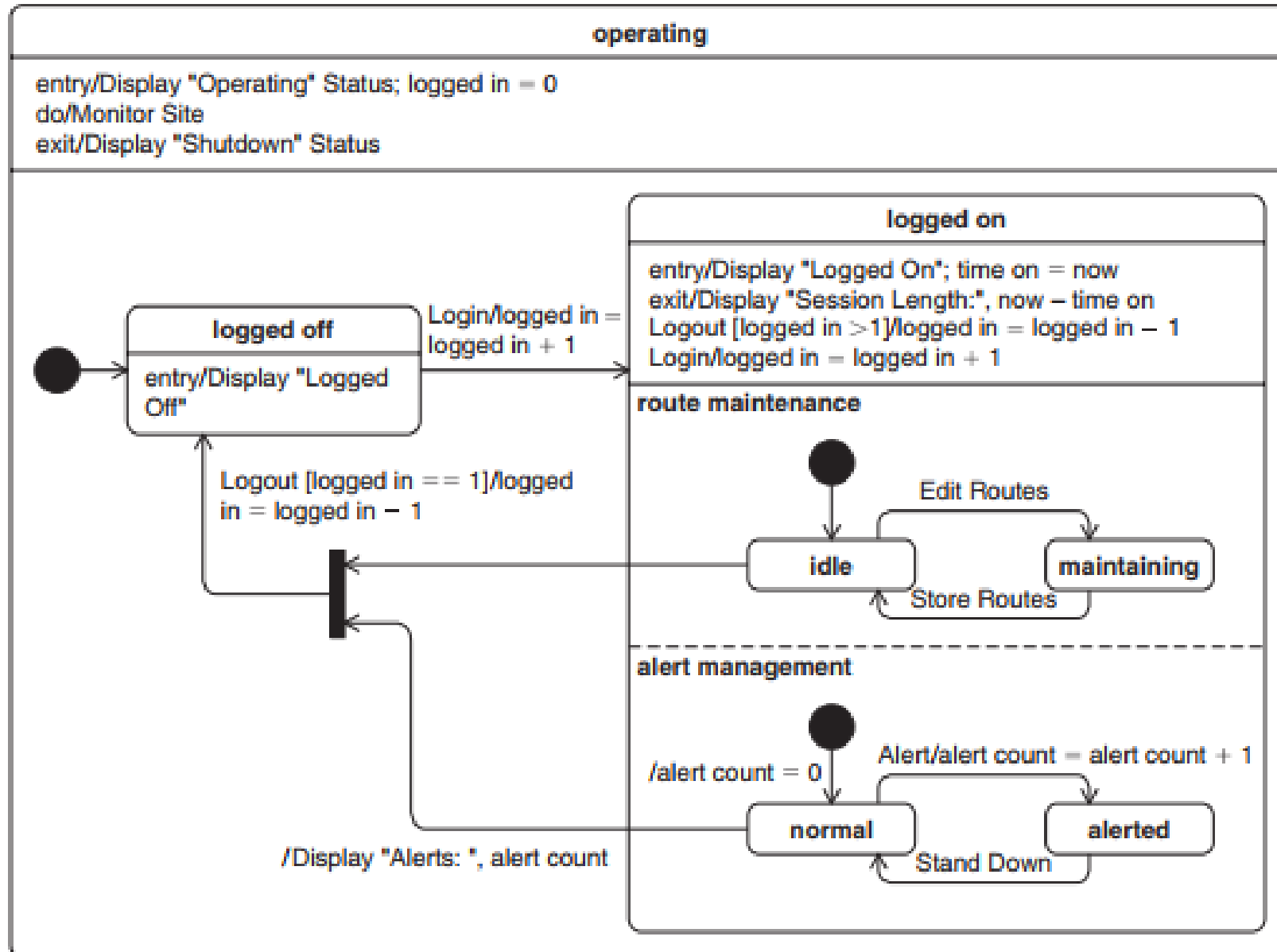


Composite states – concurrence

- When an **orthogonal composite state is active**, each region has its own active state that is **independent** of the others, and any incoming event is independently analyzed within each region.
 - A transition that ends on the composite state will trigger transitions from the initial pseudostate of each region, so **there must be an initial pseudostate in each region** for such a transition to be valid.
- Similarly, a **completion event for the composite state will occur when all the regions are in their final state.**



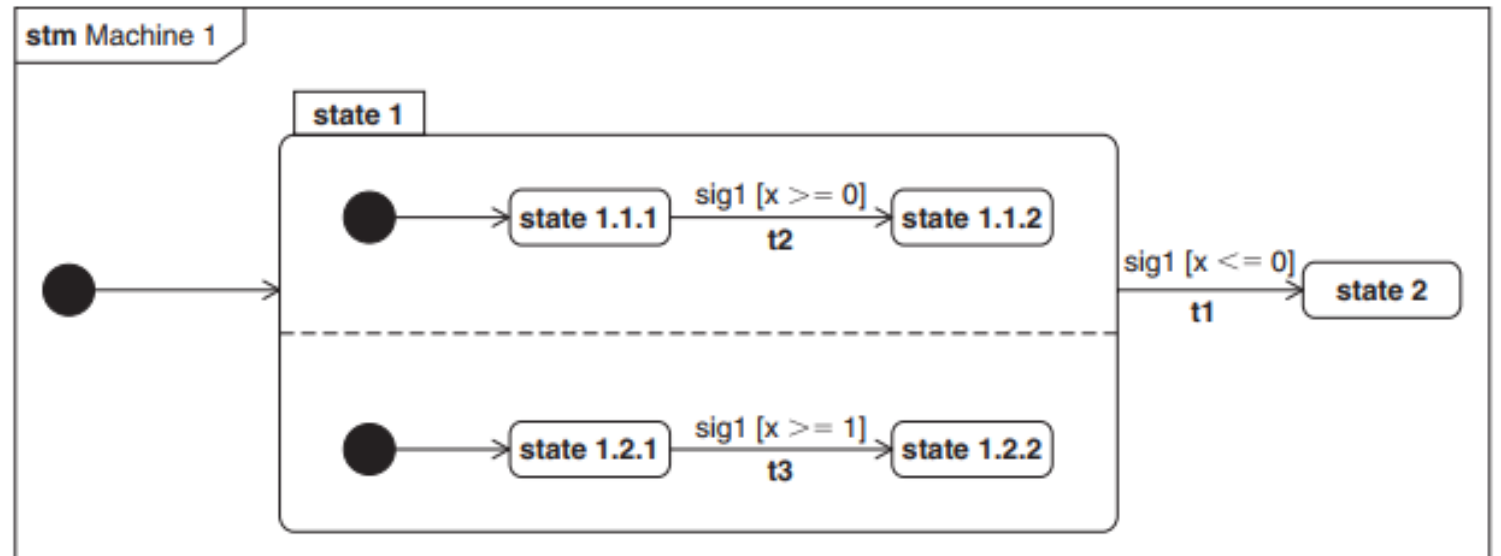
Composite states – concurrency





Firing order

- It is possible that the **same event may trigger transitions at several levels** in a state hierarchy, *and with the exception of concurrent regions, only one* of the transitions can be taken at a time. **Priority is given to the transition whose source state is innermost in the state hierarchy.**





History pseudostate

- In some design scenarios, it is desirable to handle an exception event by interrupting the behavior of the current region, responding to the event, and then returning back to the state that the region was in at the time of the interruption.
 - A **deep history pseudostate** records the states of all regions in the state hierarchy below and including the region that owns the deep history pseudostate.
 - A **shallow history pseudostate** only records the top-level state of the region that owns it.



MODES AND STATES

Modeling system modes, states, configurations with Arcadia and Capella: method and tool perspectives - 27th Annual INCOSE International Symposium (IS 2017)

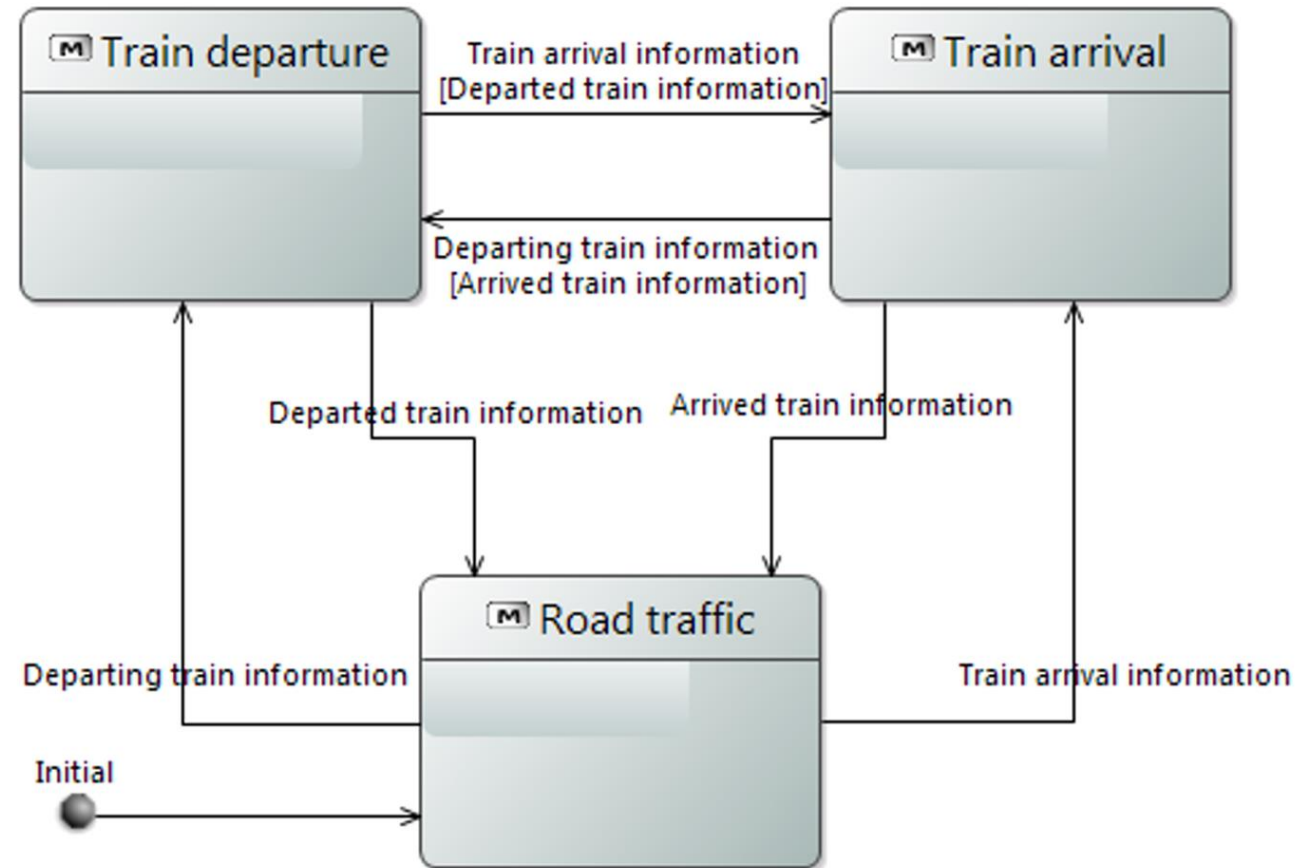


Modes

- The definition of the **system's expected behavior** (or therefore, of one of the elements mentioned earlier) **in situations decided from the design** is captured in the form of **system modes**; each mode is characterized principally by the functional content expected of the system in this mode (as a mnemonic, we talk of a “mode of life” to express the different expectations, priorities and activities in a life, and a “mode of transport” to indicate the means of travel).
- A mode can convey various concepts, such as a mission or process stage, a particular behavior required of the system, conditions of use such as a test or maintenance mode, a training mode, etc.



- *As its principal modes, the traffic control system will naturally have the modes characterizing the principal situations it should manage: train departure, train arrival and road traffic.*





States

- In the course of its life and use, the **system also passes through some states it undergoes** (we say, “What a state you are in!” and we speak of a “state of alert or of emergency” to indicate an unexpected situation).
- Most often, a state characterizes mostly structural elements (presence or absence of a component, availability or breakdown, integrity or lack of it, availability of an external actor or loss of connection with it, etc.).
- **Transition from one state to another is often involuntary**, and will therefore result, for example, in a change in property for one or more elements in the system (availability/unavailability for example).



- *The level crossing can be found in a state occupied by a vehicle stuck on the track, or on the contrary, free (as expressed by the states of the control system itself). This situation is of course foreseen, but not on the initiative of the system, so it is undergone by the system, which must consequently react.*



Configuration

- To characterize the system when it is in a **given mode or state**, we will define the notion of *configuration*: a configuration identifies a set of model elements, of all types (for example functions, components, exchanges, etc.), globally involved in use of the configuration, at a given instant. **A configuration can be attached to one or more modes and/or states.**



- A configuration intended to **describe the expectation of a mode will tend to be** (though not exclusively) **functional dominant** (capabilities, functions, exchanges, functional chains and scenarios, etc.) **to express the expected functional content** – or if it is easier to express, the functional content not present in this mode.
- A configuration intended to **describe a state may be structural dominant** (hosting physical components, physical links, indeed behavioral components hosted on the former, etc.), **but could also include functional aspects**, depending on the nature of the states considered (for example attack or failure scenarios, from a security viewpoint).



Scenario

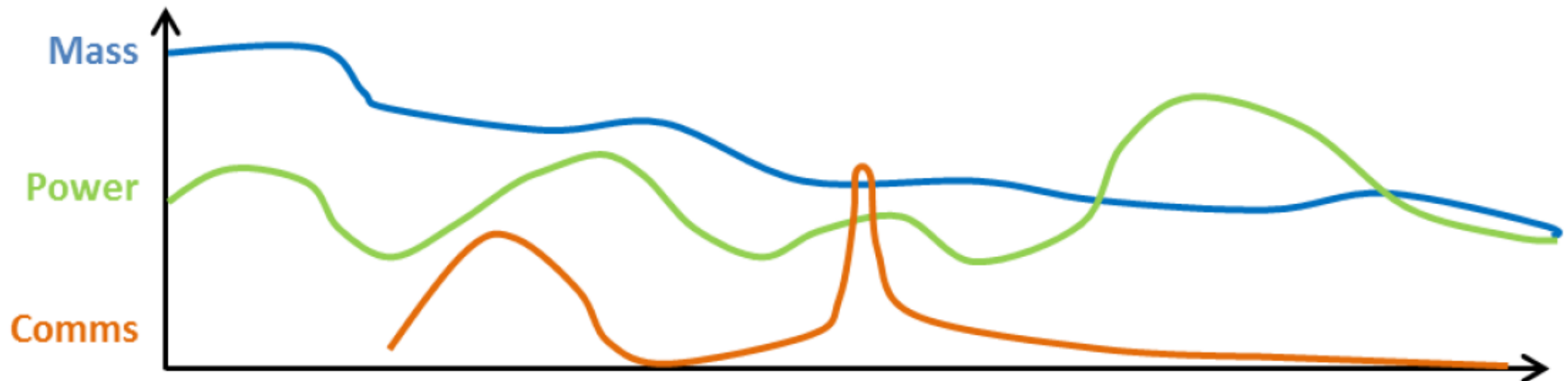
- It is therefore necessary to **define the combination of these states and modes to be able to study their consequences**. For this, we will use the notion of *a situation of superposition*. A situation is defined as a logical combination of modes and states (for example (mode1 AND state1) OR (mode2 AND (state2 OR state3))), which would express the superposition of modes and states likely to occur at a given instant.
- **A scenario can mention the transition from one situation of superposition to another**, in the same way as it will mention changes of states and modes in the course of time.



Example of situations

Mission	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6
----------------	---------	---------	---------	---------	---------	---------

System	Mode 1	Mode 2	Mode 3	Mode 4	Mode 2	Mode 3	
Subsystems	1	Mode A	Mode B	Mode C	Mode A	Mode C	Mode A
	2	Mode X	Mode Y	Mode Z	Mode X	Mode Y	
	3	Mode I	Mode J	Mode I	Mode J	Mode I	Mode J



MEANING OF STATES AND MODES

OA	describe either general situations that the organization considered confronts (usually rather states such as routine conditions, states of crisis, a situation where there is a lack of resources, for example), or the stages of a mission , or of the organization's normal functioning (usually rather modes, such as an airplane's or space launcher's stages of flight).
SA	describing the expectation on the system , as desired by the customer; they are most often perceived and employed by the final users. In particular, they capture the different modes and conditions of use required of the system in different situations, and feared situations, with the minimal behavior required when facing these situations
LA	the system states and modes respond this time to design choices or constraints . New modes and states reflecting the choices of solutions can appear, which cannot be linked to those of need analysis.
PA	applied to the system, but also to each logical architecture component and to the physical architecture components linked to it: modes and states, as well as the content of their associated configurations, should be coherent with traceability links (between functions, between components, between exchanges triggering transitions, etc.) between both architecture perspectives.



Use Cases (Capabilities)

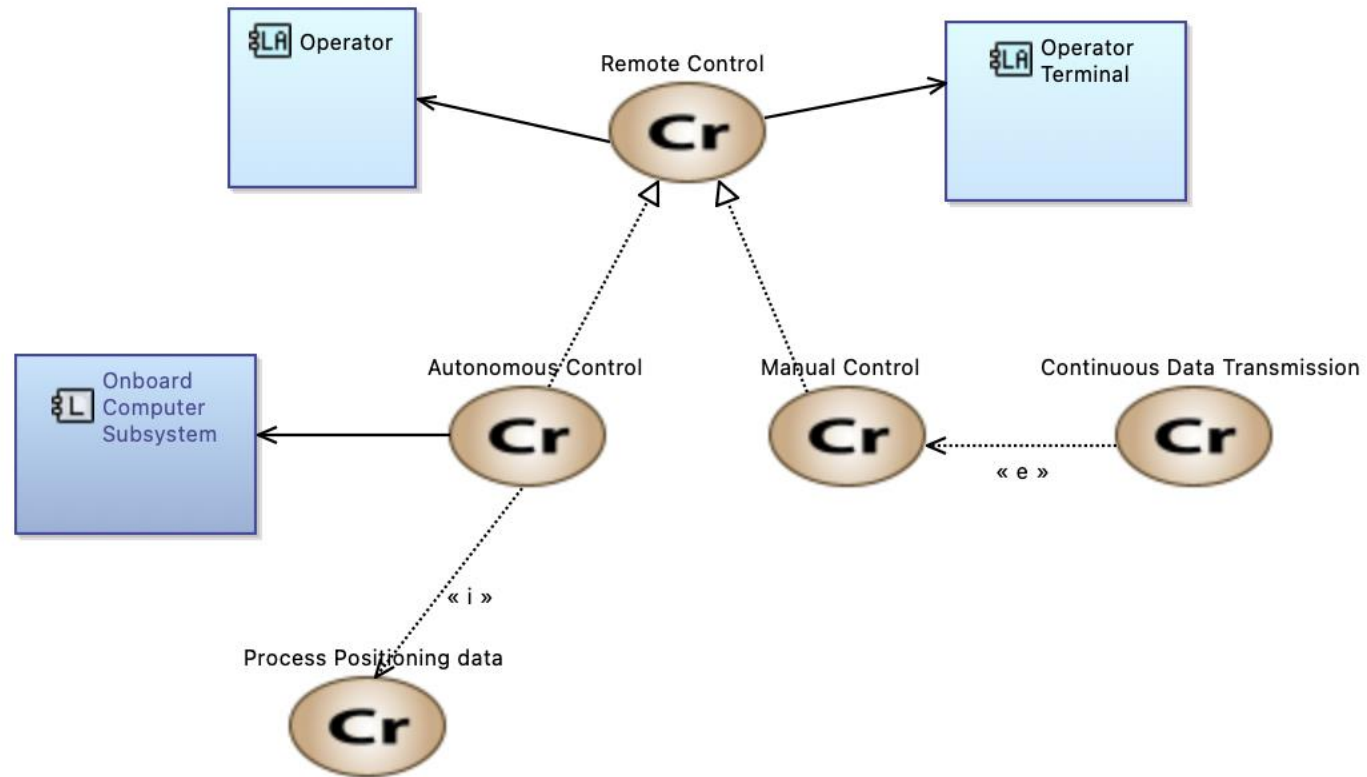


Use case introduction

- Use cases describe the **functionality of a system in terms of how it is used to achieve the goals** of its various users.
- Use cases can also be **classified using generalization**, but in addition, one use case may **include or extend** other use cases.
- The users of a system are described by **actors**, which may represent external systems or humans who interact with the system. Actors are related to the use cases in which they participate.
- *Use cases have been viewed as a mechanism to capture system requirements in terms of the uses of the system.*



Example use case diagram





Actors

- An actor is used to represent the **role of a human, an organization, or any external system** that participates in the use of some system.
 - Actors may interact directly with the system or indirectly through other actors.
 - Actors can be classified using the standard **generalization** relationship.
- **Actor classification** has a similar meaning to the classification of other classifiable model elements.
- For example, a specialized actor participates in all the use cases that the more general actor participates in.



Use cases

- A use case **describes the goals of a system** from the perspective of the users of the system.
- The goals are described **in terms of functionality that the system must support**. Typically, the use case description identifies the goal(s) of the use case, a **main pattern** of use, and a number of **variant** uses.
- A **use case may cover one or more scenarios** that correspond to how the system interacts with its actors under different circumstances.
- Actors are related to use cases by communication paths, which are represented as associations, with some restrictions.

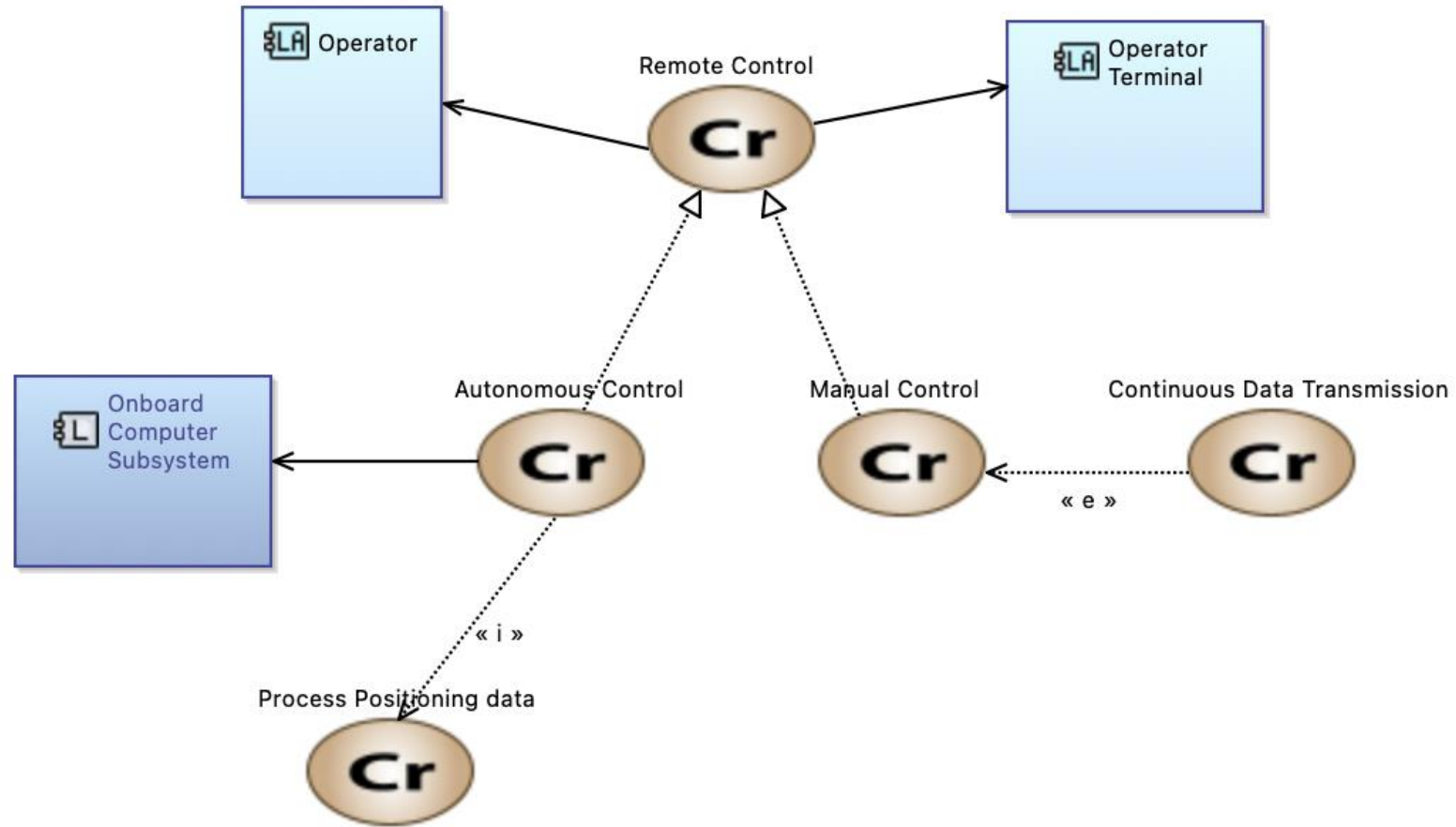


Use case relationships (inclusion / extension / classification)

- The **inclusion** relationship allows one use case, referred to as the base use case, to include the functionality of another use case, called the included use case. **The included use case is always performed when the base use case is performed.**
- A use case can also **extend** a base use case using the extension relationship. The extending use case is a fragment of functionality that is **not considered part of the base use case functionality.** It often describes some exceptional behavior in the interaction, such as error handling between subject and actors that does not contribute directly to the goal of the base use case
- The meaning of **classification** is similar to that for other classifiable model elements. One implication, for example, is that the **scenarios for the general use case are also scenarios of the specialized use case.** It also means that the actors associated with a specialized use case can also participate in scenarios described by a general use case.



A set of use cases for the Surveillance System





Use case description

- A text-based use case description can be used to provide additional information to support the use case definition. This description can contribute significantly to the use case's value.
- A typical use case description may include the following:
 - **Pre-conditions**—the conditions that must hold for the use case to begin.
 - **Post-conditions**—the conditions that must hold once the use case has completed.
 - **Primary flow**—*the most frequent scenario or scenarios of the use case.*
 - **Alternate and/or exception flows**—*the scenarios that are less frequent or other than nominal. The exception flows may reference extension points and generally represent flows that are not directly in support of the goals of the primary flow.*

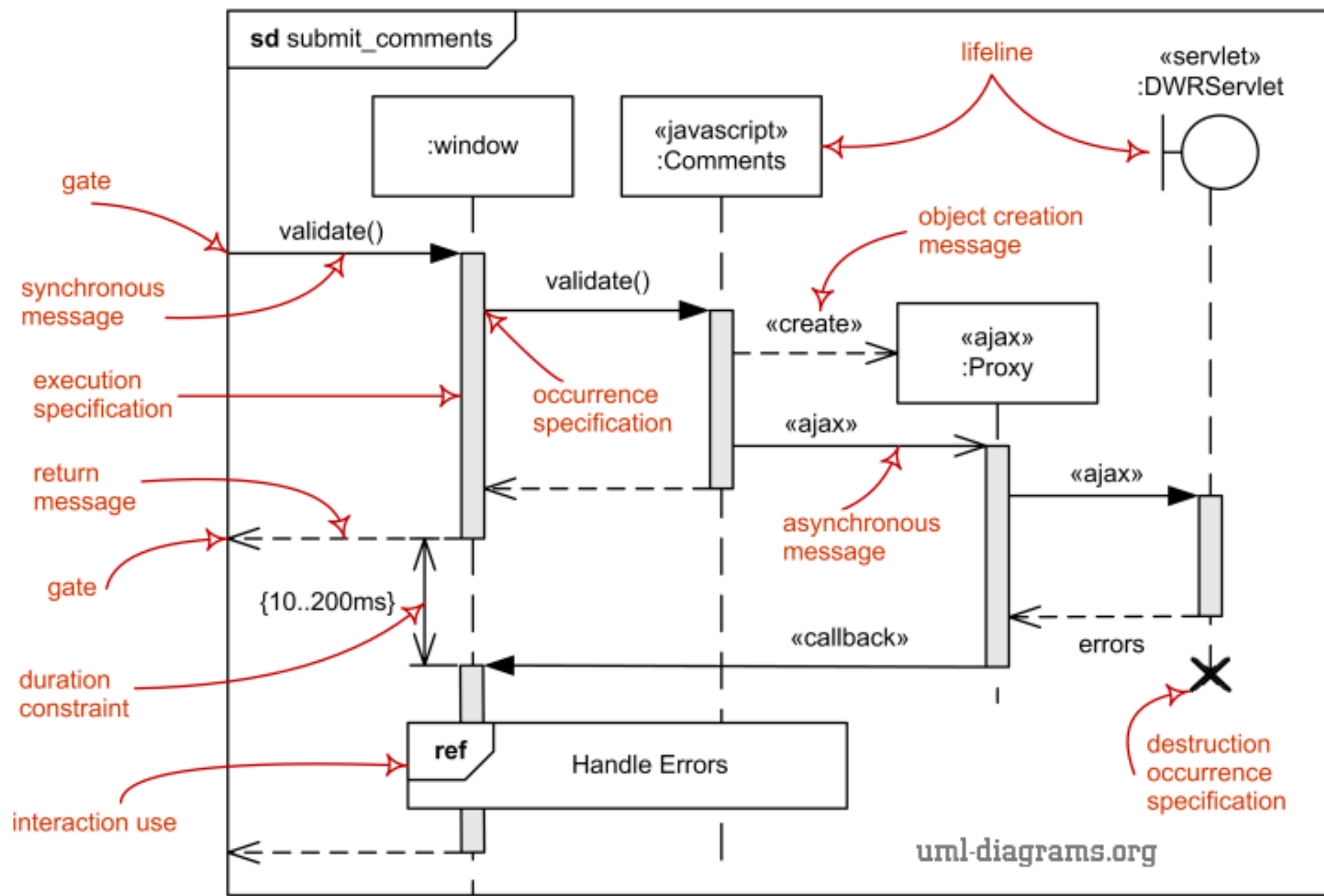


Sequence (Scenarios)



SEQUENCE INTRODUCTION

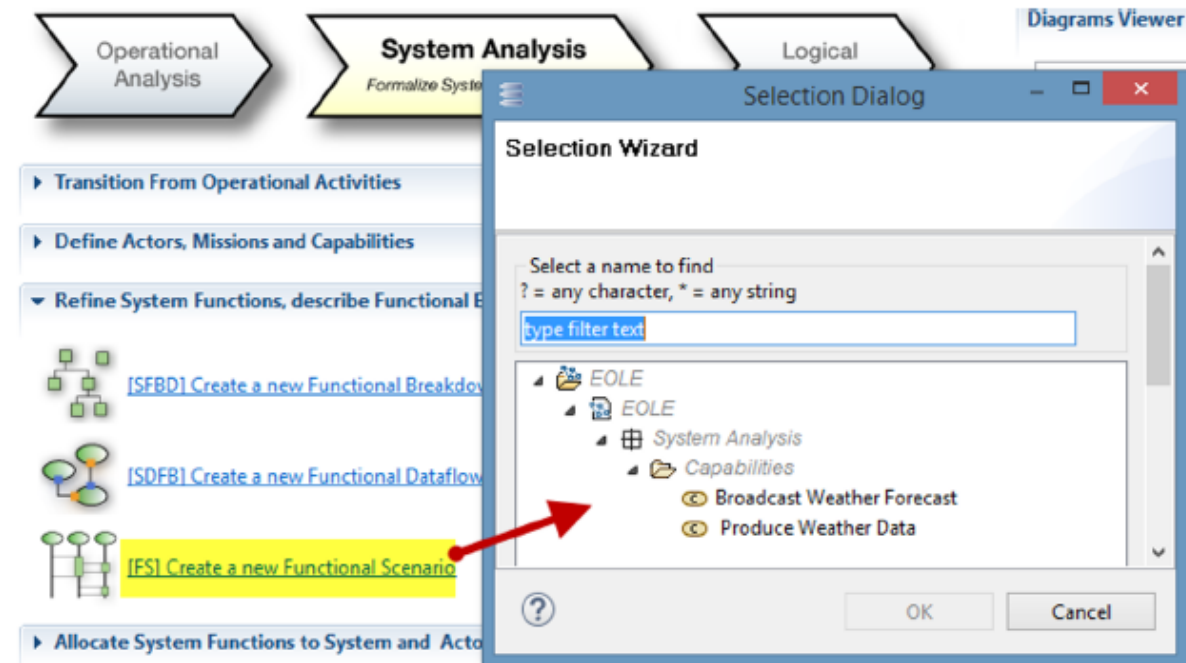
- Represent the **interaction between structural elements** in a model as a sequence of message exchanges.
- A message can represent the **invocation of a service on a system component or the sending of a signal**.
- The structural elements of a block are represented by lifelines on a sequence diagram.
- The sequence diagram **describes the interaction between these lifelines as an ordered series of occurrence** specifications that describe different kinds of occurrences, such as the sending and receiving of messages, the creation and destruction of objects, or the start and end of behavior executions.





Sequence diagram → Scenarios

- A Scenario describes the **behavior of the System in the context of a particular Capability**.
 - *Functional Scenario*
 - *Exchange Scenario*



- Even if we do not wish to use this concept, Capella automatically creates a Capability the first time a Scenario is created, unless of course Capabilities already exist. In this case, Capella asks to choose one Capability to attach the new Scenario.



Messages

- A sender of an **asynchronous message** continues to execute after sending the message, whereas a sender of a **synchronous message** waits until it receives a reply from the receiver that it has completed its processing of the message before continuing execution.
 - An open arrowhead means an asynchronous message.
 - A closed arrowhead means a synchronous message.
 - An arrowhead on a dashed line shows a reply message.

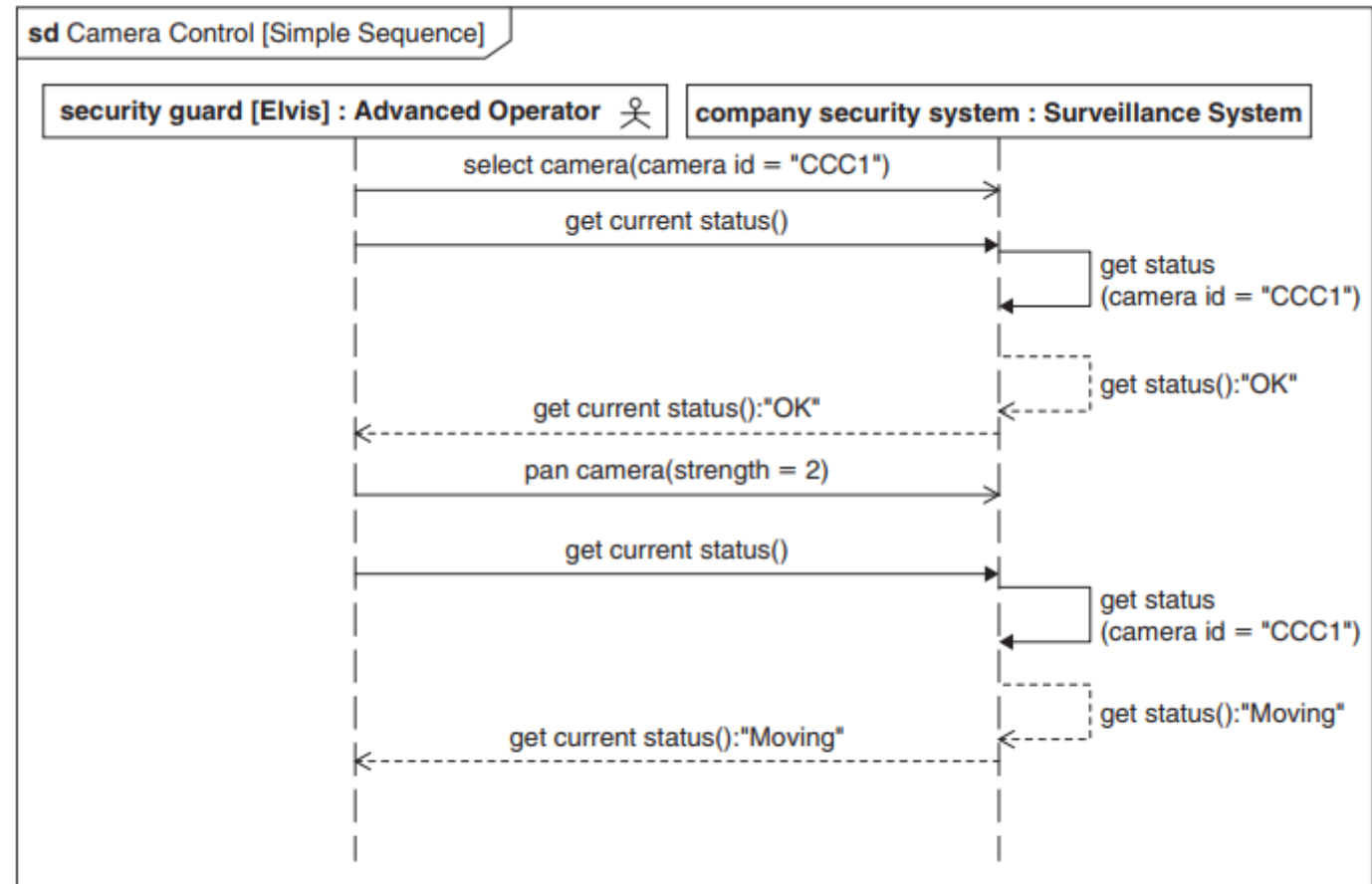


FIGURE 10.5

Synchronous and asynchronous messages exchanged between lifelines.

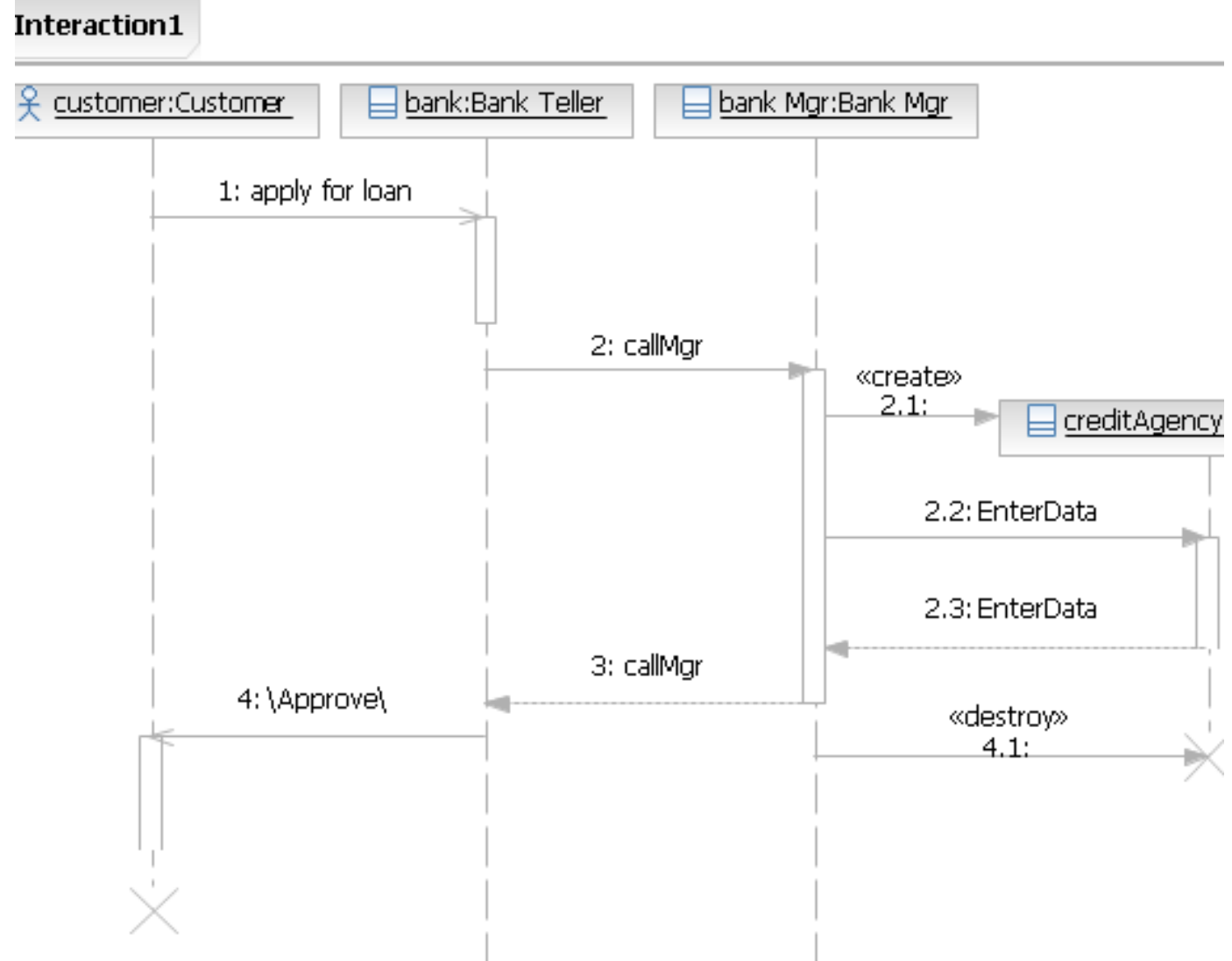


Message	Description	Example
Create message	A create message represents the creation of an instance in an interaction. The create message is represented by the keyword «create». The target lifeline begins at the point of the create message.	In a banking scenario, a bank manager might start a credit check on a client by sending a create message to the server.
Destroy message	A destroy message represents the destruction of an instance in an interaction. The destroy message is represented by the keyword «destroy». The target lifeline ends at the point of the destroy message, and is denoted by an X.	A bank manager, after starting a credit check, might close or destroy the credit program application for a customer.
Synchronous call message	Synchronous calls, which are associated with an operation, have a send and receive message. A message is sent from the source lifeline to the target lifeline. The source lifeline is blocked from other operations until it receives a response from the target lifeline.	A bank teller might send a credit request to the bank manager for approval and must wait for a response before further serving the customer.
Asynchronous call message	Asynchronous calls, which are associated with an operation, typically have only a send message, but can also have a reply message. In contrast to a synchronous message, the source lifeline is not blocked from receiving or sending other messages. You can also move the send and receive points individually to delay the time between the send and receive events. You might choose to do this if a response is not time-sensitive or order-sensitive.	A bank customer could apply for credit but can receive banking information over the phone or request money from an ATM, while waiting to hear about the credit application.
Asynchronous signal message	Asynchronous signal messages, are associated with a signal. A signal differs from a message in that there is no operation associated with the signal. A signal can represent an interrupt or error condition. To specify a signal, you create an asynchronous call message and change the type in the message properties view.	A credit agency could send an error signal message to the bank manager that states a failure to connect to the credit bureau.



Example

- A customer gives the application for the loan to the bank teller.
- The bank teller sends the application to be processed by the bank manager and waits for the manager to finish.
- The bank manager starts the credit check program, enters the data, and waits for the credit agency to send the results.
- The bank manager receives a response and sends a message to the bank teller that states the decision.
- The bank teller sends a message to the customer about whether the loan was approved.
- The bank manager closes the credit agency program and the customer completes the transaction.





Executions

- The receipt of a message by a lifeline may **trigger the execution of a behavior** in the receiver.
 - Activations are rectangular symbols overlaid vertically on lifelines.

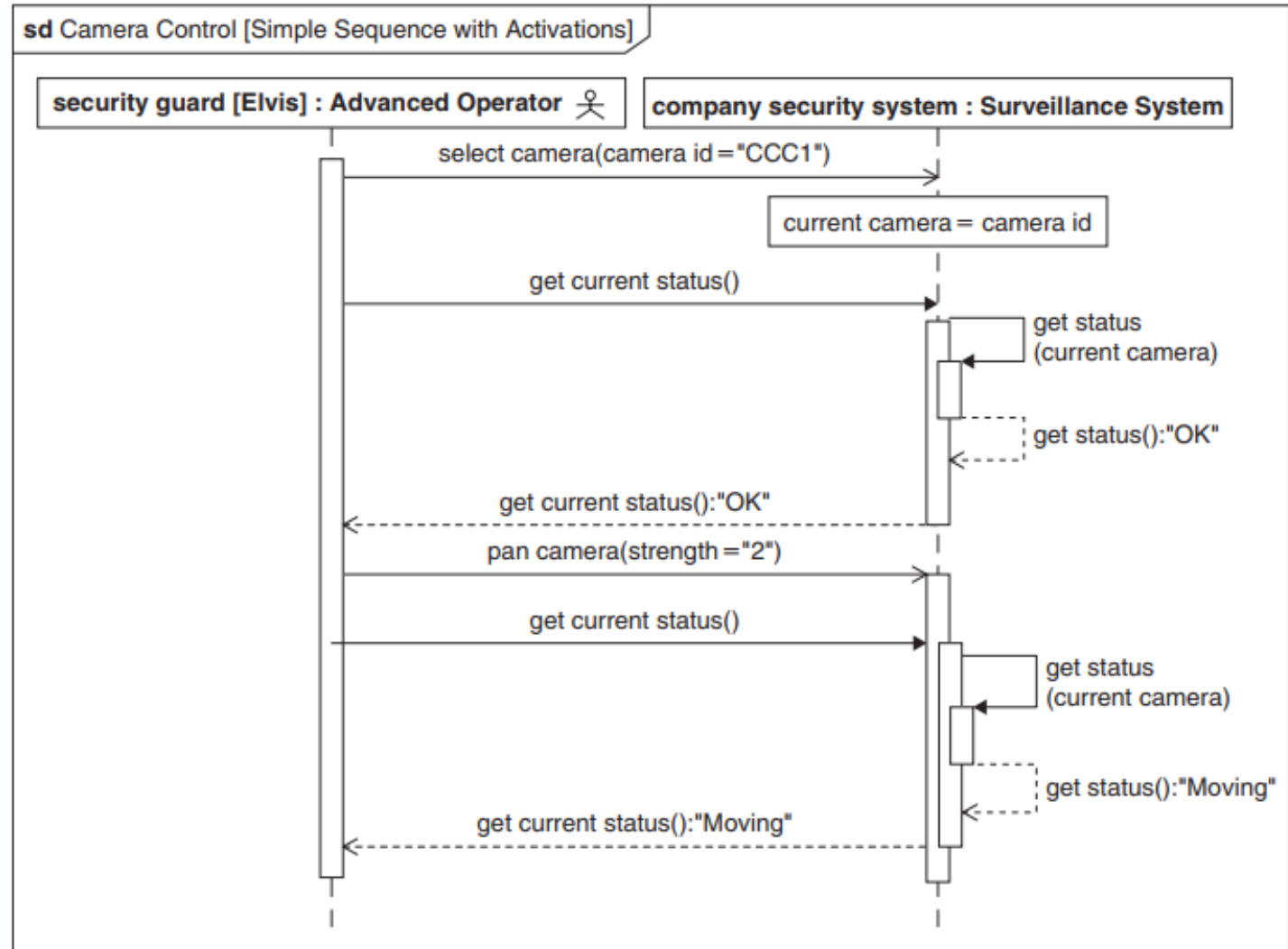


FIGURE 10.7

Lifelines with activations.



Representing time

- A **time observation** refers to an instant in time corresponding to the occurrence of some **event during** the execution of the interaction, and a **duration observation** refers to the **time taken between two instants** during the execution of the interaction.
- A time constraint and a duration constraint can use observations to express constraints involving the values of those observations.

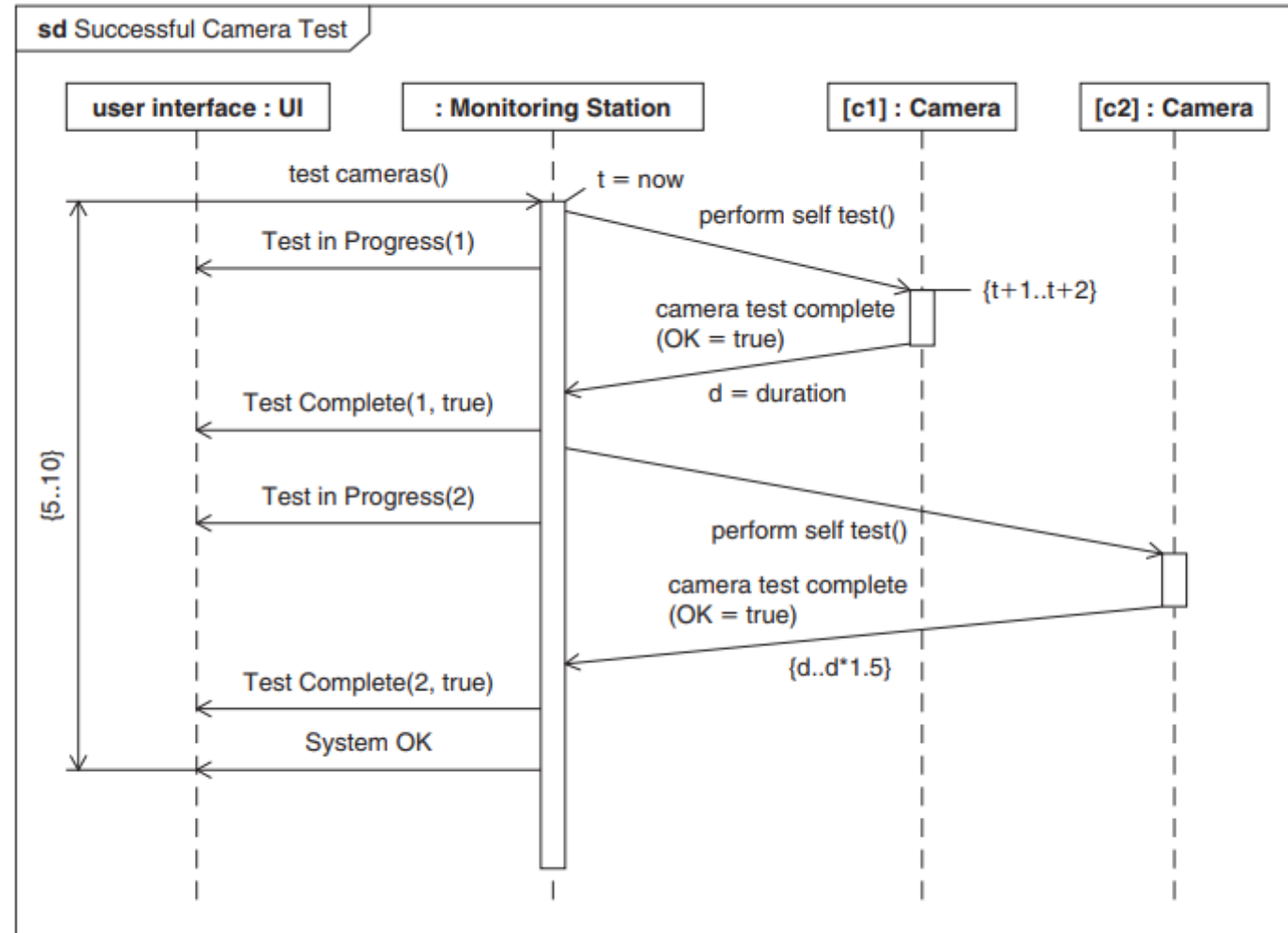


FIGURE 10.10

Representing time on a sequence diagram.



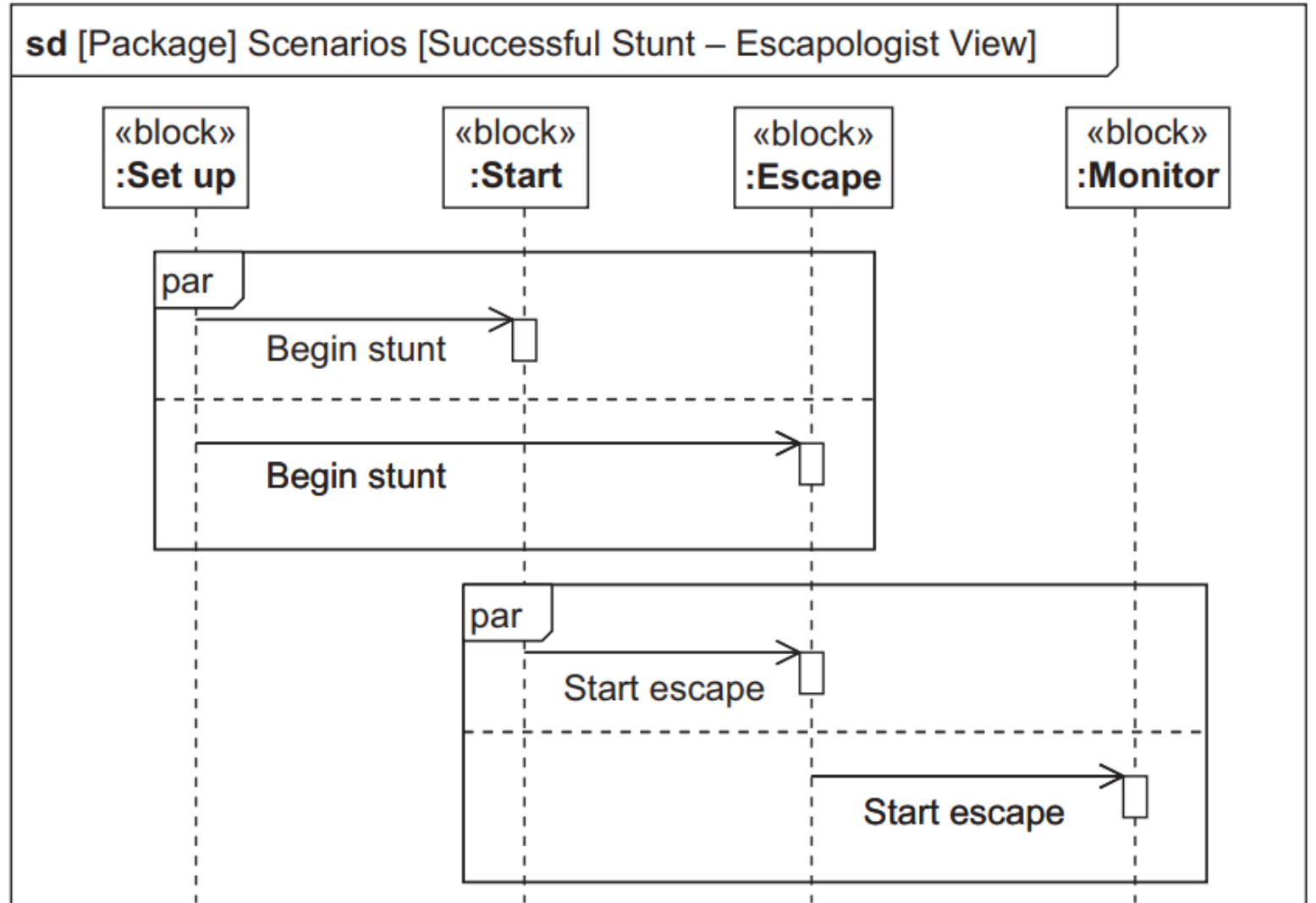
FRAGMENTS

- More complex patterns of interaction can be modeled using constructs called combined fragments.
 - **Par-** an operator in which operands can occur in parallel, each following weak sequencing rules. There is no implied order between occurrences in different operands.
 - **Alt/else**—an operator in which exactly one of its operands will be selected based on the value of its guard. The guard on each operand is evaluated before selection, and if the guard on one of the operands is valid, then that one is selected. If more than one operand has a valid guard then the selection is nondeterministic. An optional else fragment is valid only if none of the guards on the other operands are valid.
 - **Loop**—an operator in which the trace represented by its operand repeats until its termination constraint is met. A loop may define lower and upper bounds on the number of iterations as well as the guard expression.



PARALLEL

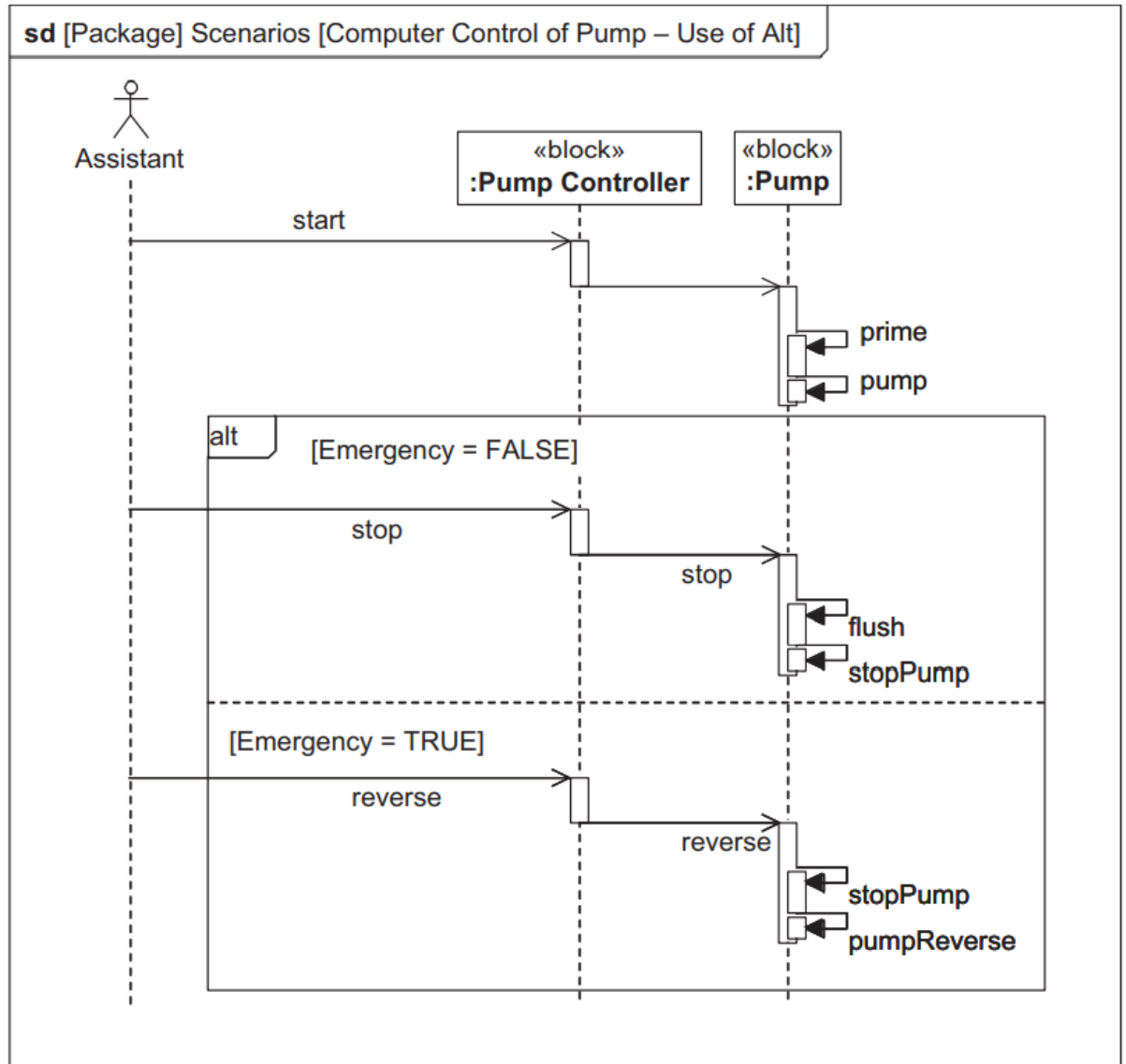
- Each parallel path appears in a separate compartment within the combined fragment frame. The parallel compartments are divided by a dashed line, and the combined fragment uses the keyword `par`.





ALTERNATIVES

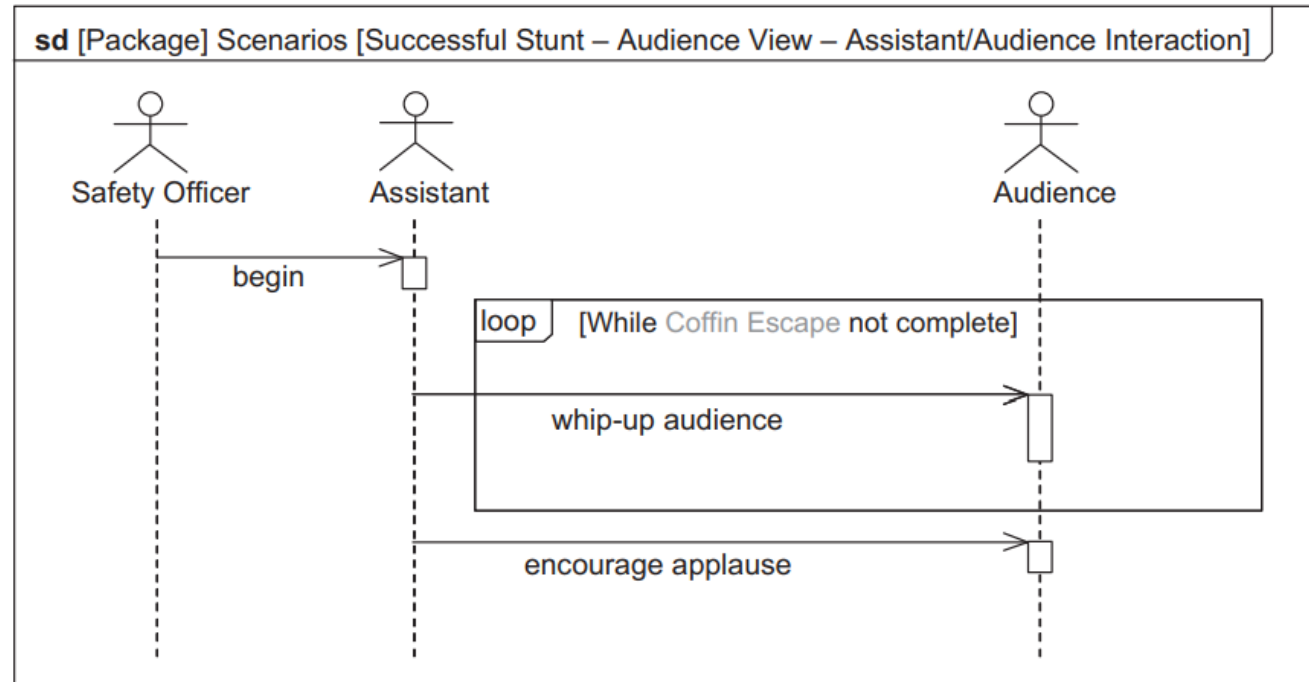
- Sometimes two or more Scenarios are so similar that showing alternative paths on a single diagram rather than one per diagram is desirable. SysML allows Scenarios to be modelled in this way using alternative combined fragments.





LOOP

- The keyword may be accompanied by a repetition count specifying a minimum and maximum count as well as a guard condition. The loop is executed while the guard condition is true but at least the minimum count, irrespective of the guard condition and never more than the maximum count.
- The syntax for loop counts is
 - loop minimum = 0, unlimited maximum
 - loop(repeat) minimum = maximum = repeat
 - loop(min, max) minimum & maximum specified, $\text{min} \leq \text{max}$





Complex interactions described using interaction operators

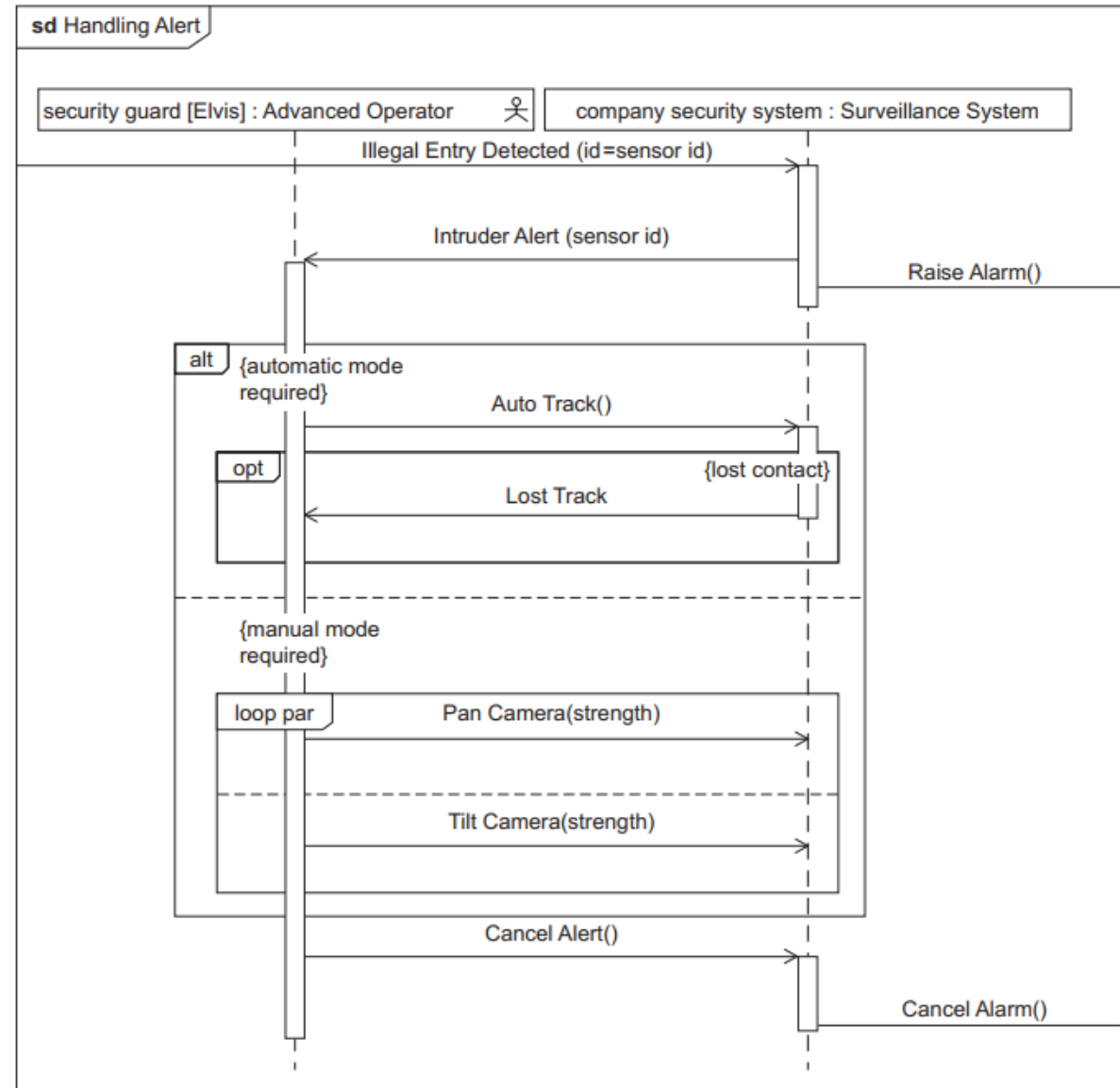
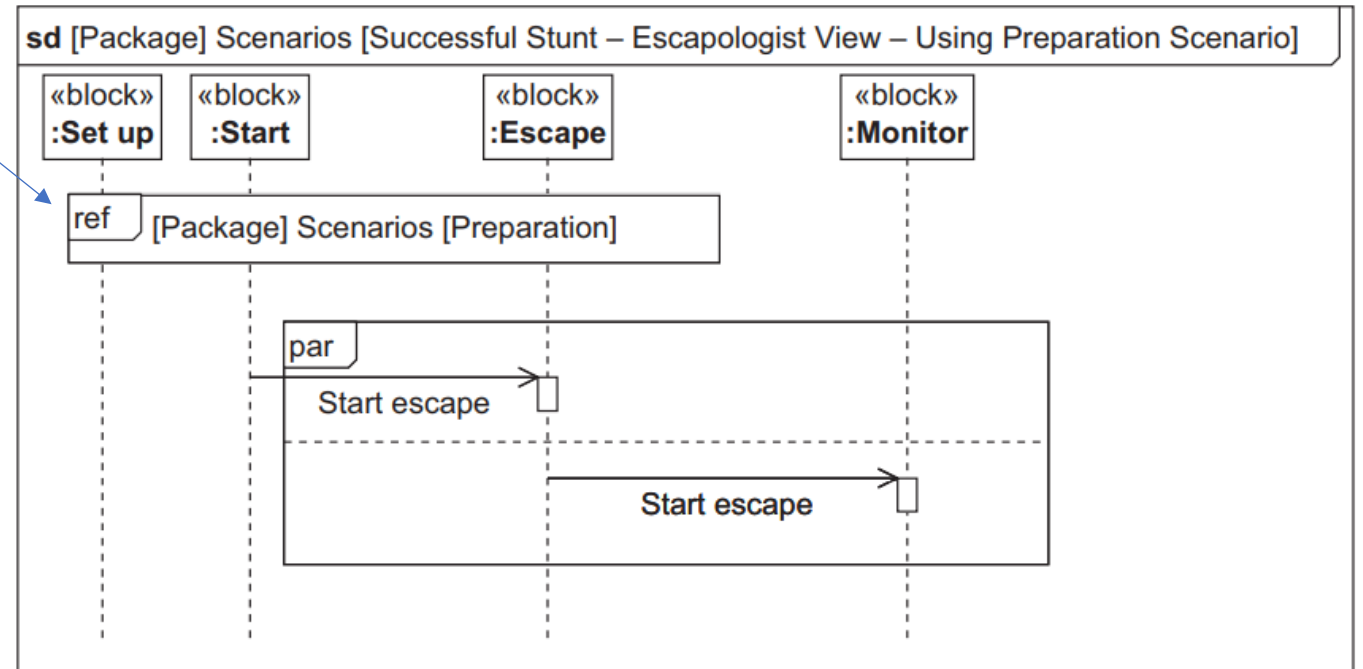
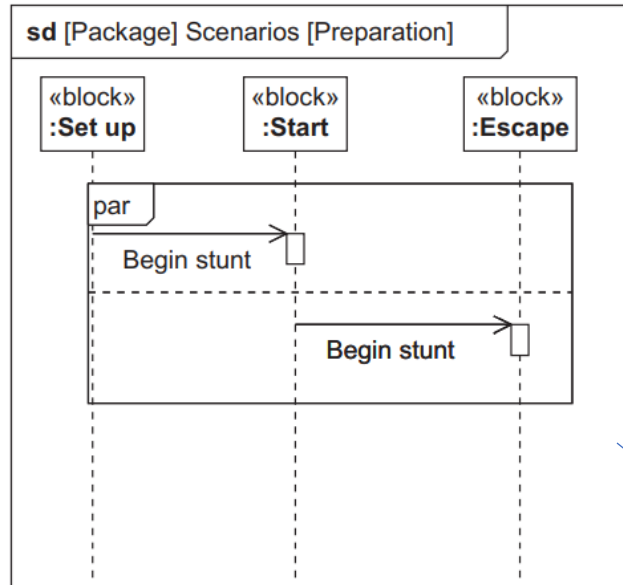


FIGURE 10.12

Complex interactions described using interaction operators.



Reusing of fragments





Using references

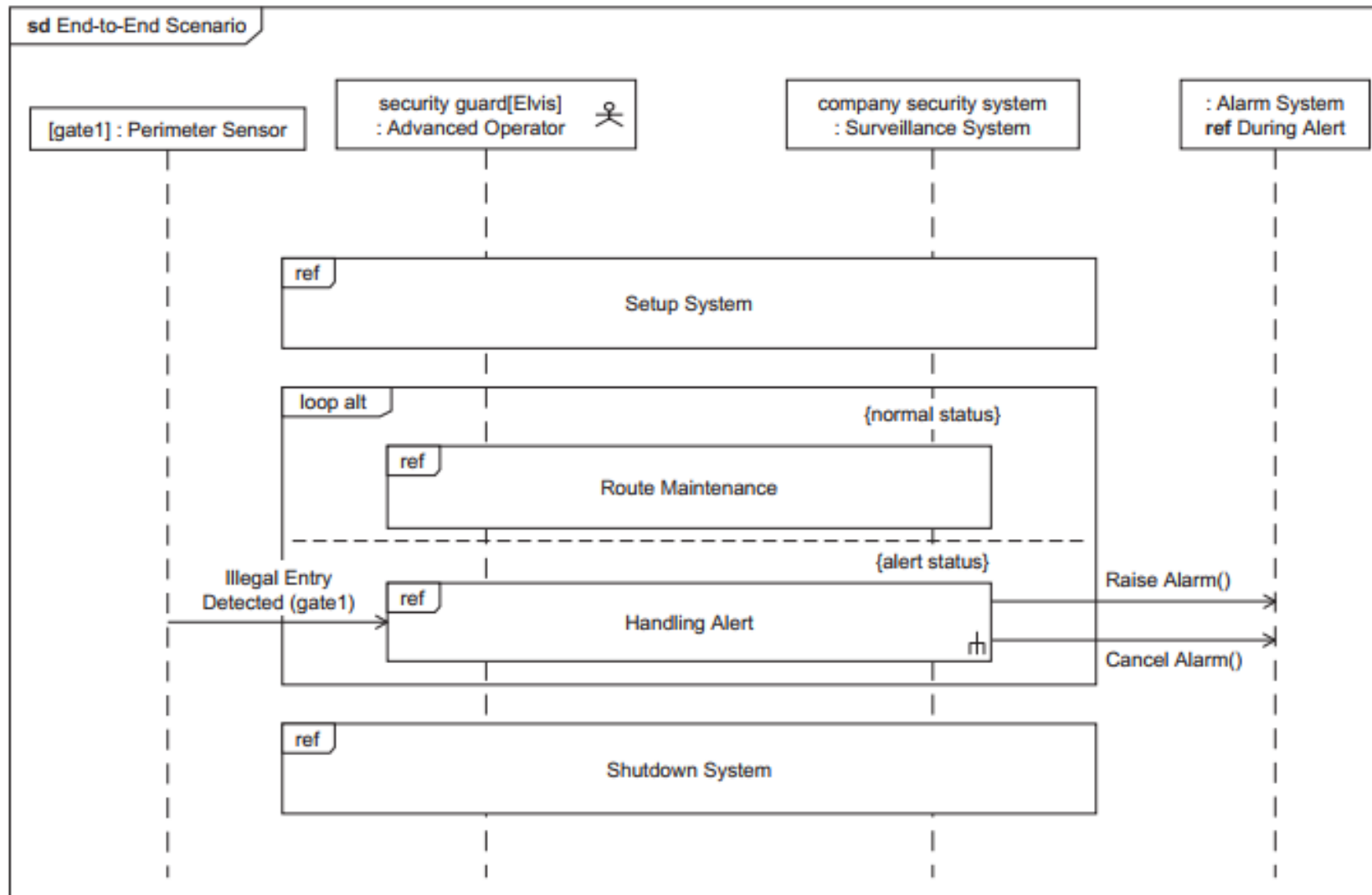
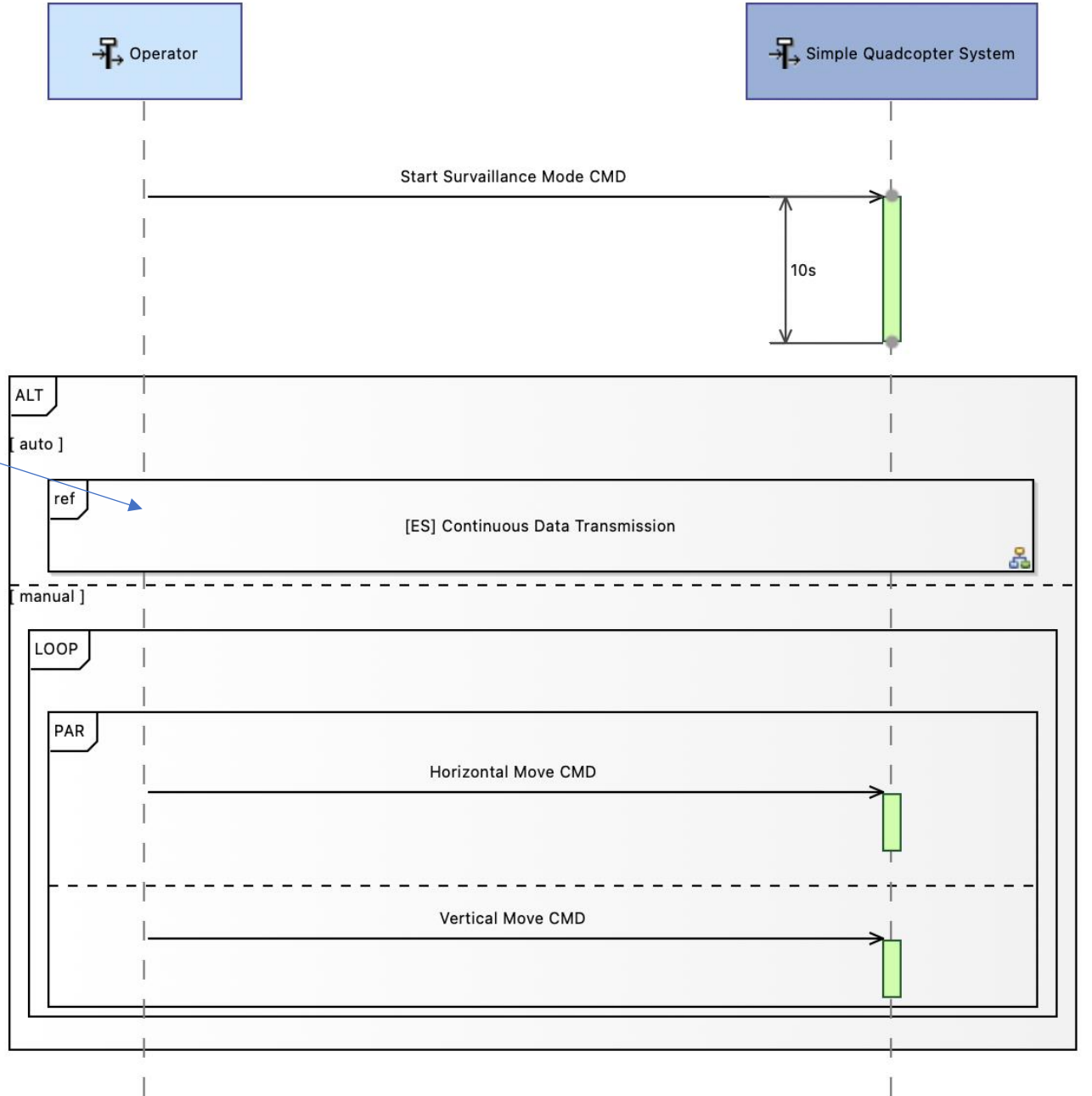
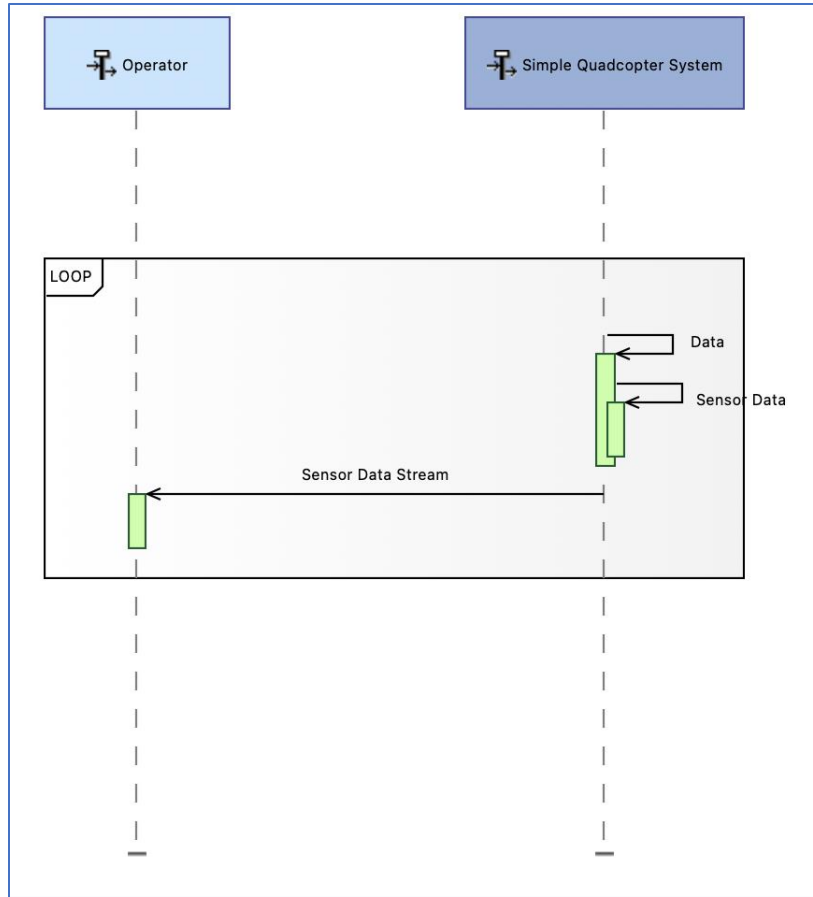


FIGURE 10.15

Reference to another interaction.





FUNCTIONAL CHAINS

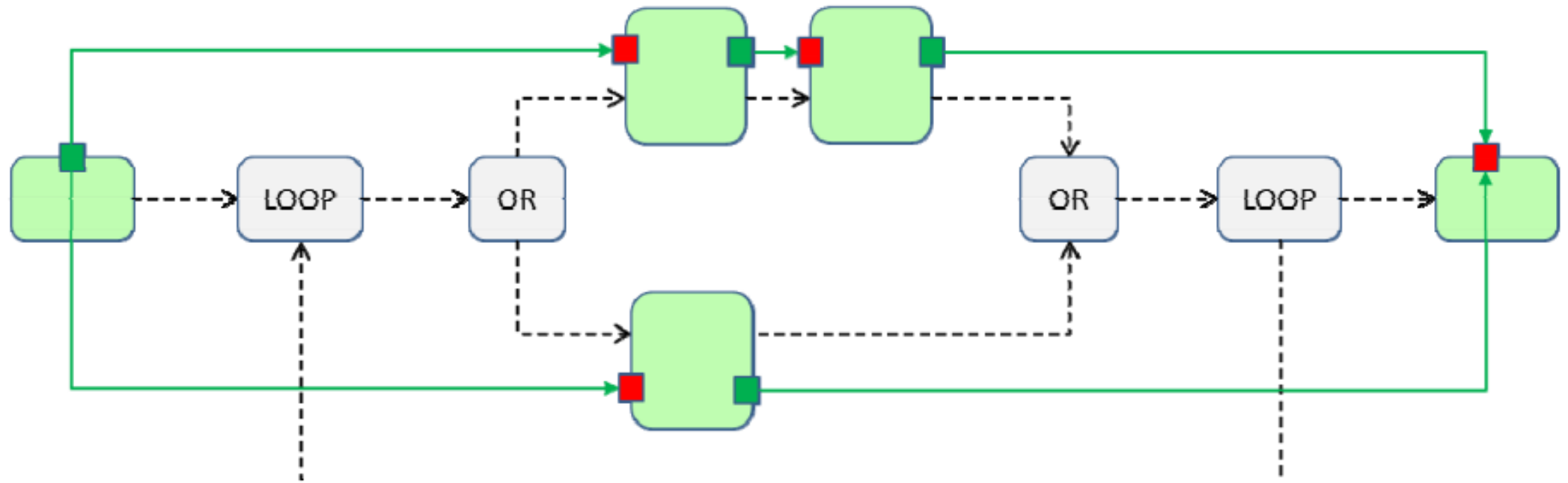


Functional chains and sequence diagrams

- A functional chain is an **ordered set of references to functions and the functional exchanges** that link them, describing one possible path among all the paths forming the dataflow.
- A functional chain is used to **describe the system behavior in a particular usage context**, to contribute to one or more system capabilities, and especially to **specify the non-functional expectations on this path (latency between the chain's start and end, quality of service, criticality level, association with a feared event, etc.)**.



FUNCTIONAL CHAIN WITH CONTROL NODES

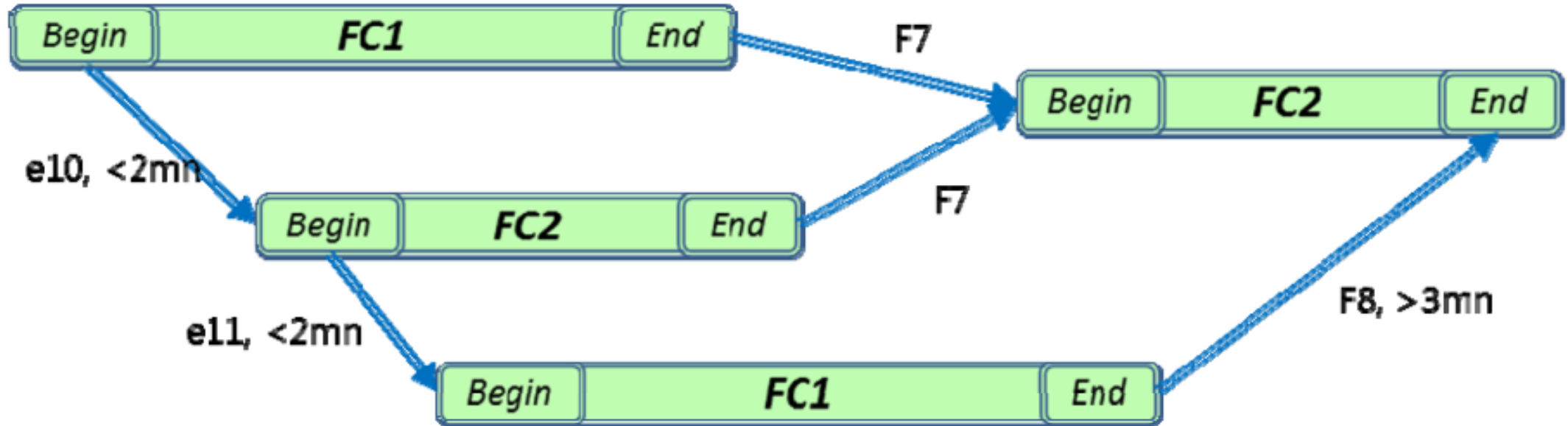


Control nodes can be defined between the sequence links, to express the parallelism or alternative between several sequences of functions, or, also the iteration or condition of a sequence to be realized.

[it is the eFFBD]



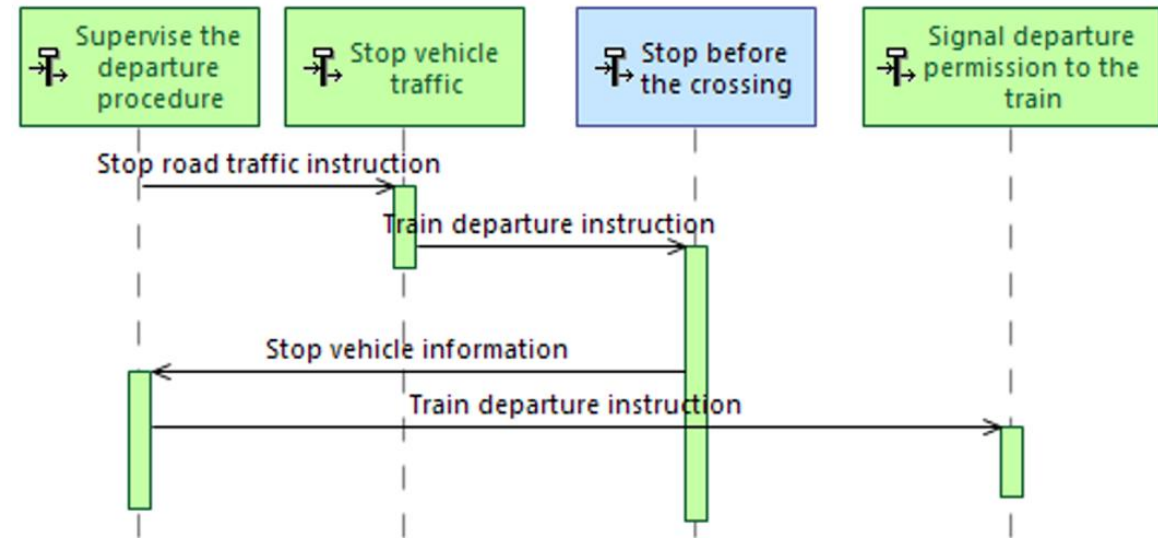
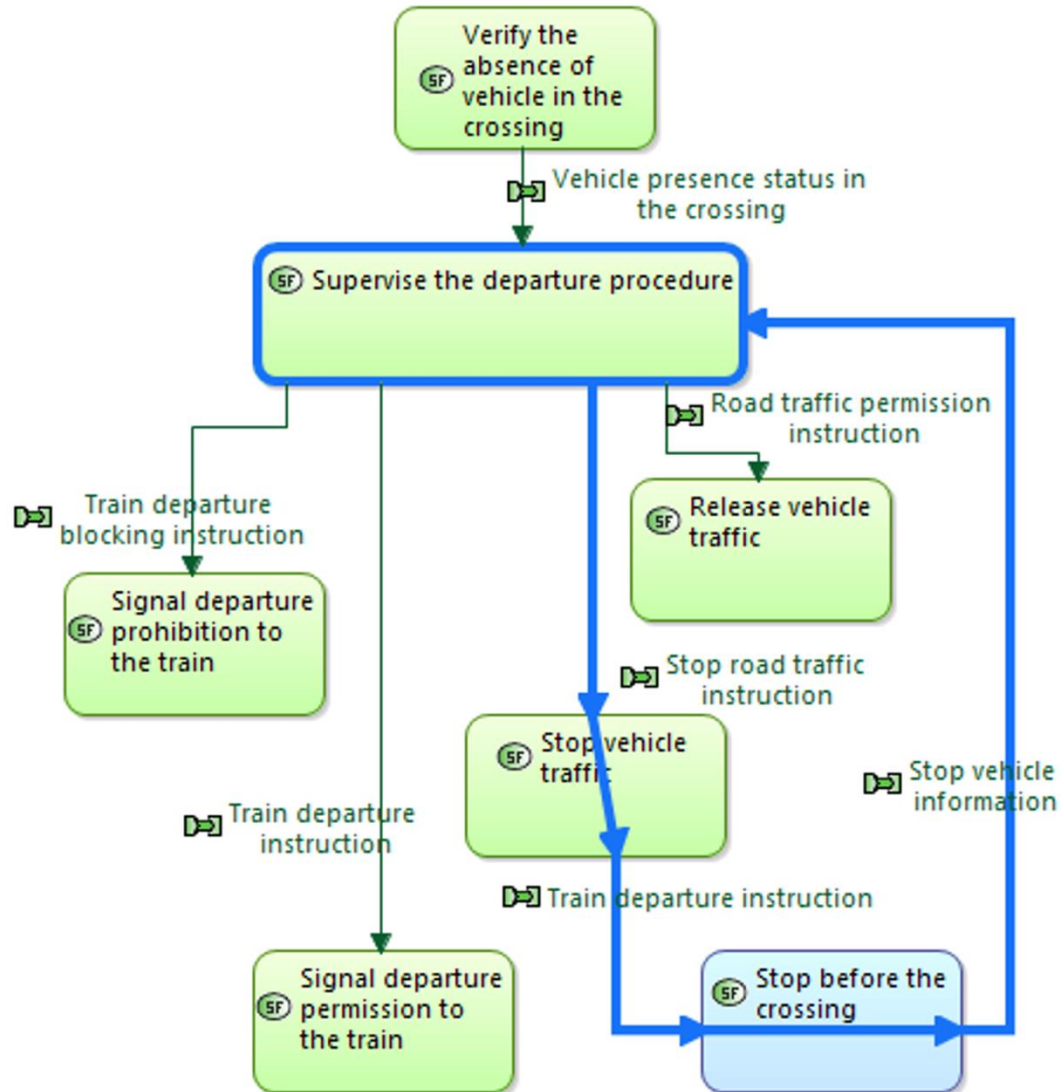
FUNCTIONAL CHAIN ORCHESTRATION

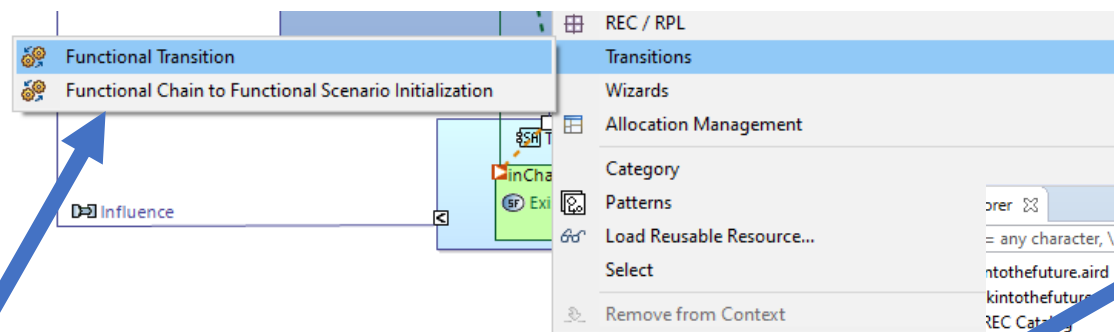


- An orchestration is an ordering of functional chains or scenarios expressing parallelism conditions between them, and the temporal precedence between some of their elements.
- An orchestration is defined by a set of references to functional chains and scenarios, and by precedence links between functions or exchanges belonging to two of them.



RELATION BETWEEN FC AND SCN

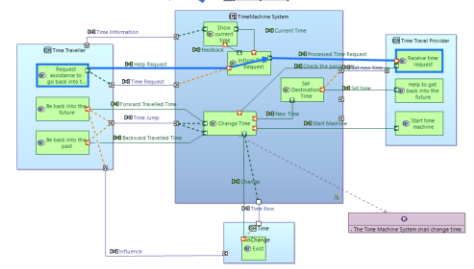




3. Right Click on the Functional chain and create the Functional Chain Description Diagram

2. Right click and create do a transition to a Functional Scenario Initialization

1. Create a functional chain



4. Click on the Capabilities Functional Scenario in the Project Explore and create a Function Scenario

Scenario) [FS] Functional
ancing Elements



Final Considerations