SEQUENCES AND USE CASES

Prepared by Prof. Dr. Christopher Shneider Cerqueira

2025

WEEK	CLASS ACTIVITY	REF	INDIVIDUAL	W	GROUP	W
1 28/Jul	Course Structure and Initial Definitions Systems Engineering Review	[illegible]	IA-01 - Reading and [illegible] Questions (10)	10%		0%
2 04/Aug	Classical Systems Engineering Diagrams (IDEF-0/N2/eFFBD/DFD)	[4] *papers	IA-02	0%	GA-02 - Prepare a representation of your system using classical Diagrams	50%
3 11/Aug	Transition from Legacy to MBSE MBSE Methodologies MBSE Languages	[5] [7] *[illegible]	IA-03 - Reading and [illegible] Questions (10)	10%		0%
4 18/Aug	OPM - Basic	[6]	IA-04 - Exercises	10%		0%
5 25/Aug	OPM - Extended	[6]	IA-05 - Exercises	10%		0%
6 01/Sep	OPM - Group Presentation		IA-06	0%	G6 - Prepare a presentation of your system using OPM	50%
7 08/Sep	SysML Introduction (bdd/ibd)	[7]	IA-07 - Exercises	10%		0%
8 15/Sep	P1 - Conceptual Questions and Case	[1] [2] [3] [4] [6]	IA-08 - Questions and a mini-case	50%	GA-08 -	
				100%		100%

WEEK	CLASS ACTIVITY	REF	INDIVIDUAL	W	GROUP	W
9 29/Sep	SysML (act/stm)	[7]		10%	GA-09 -	0%
10 06/Oct	SysML (seq/uc)	[7]	IA-10 - Exercises	10%	GA-10 -	0%
11 13/Oct	Simulation on SysML		IA-11 -	0%	GA-11 -	0%
12 20/Oct 21/Oct	SysML (pkg/req)	[7]	IA-12 - Exercises	10%	GA-12 -	0%
13 27/Oct	Arcadia process applied into the SysML	[5]	IA-13 -	0%	GA-13 -	0%
14 03/Nov	Some System Analysis on SysML SysML V2 Perspectives	[8]	IA-14 -	0%	GA-14 -	0%
15 10/Nov	SysML Group Presentation Course Ending		IA-15 -	0%	GA-15 - Prepare a presentation of your system using SysML	100%
16 17/Nov	P2 - Conceptual Questions and Case	[5] [7]	IA-16 - Questions and a mini-case	70%	GA-16 -	
				100%		100%
EXAM 24/Nov 08/Dec	**If necessary: Writing an article (min 6pgs / max 10pgs) reporting the case of their group in the SIGE standard.**					**100%**

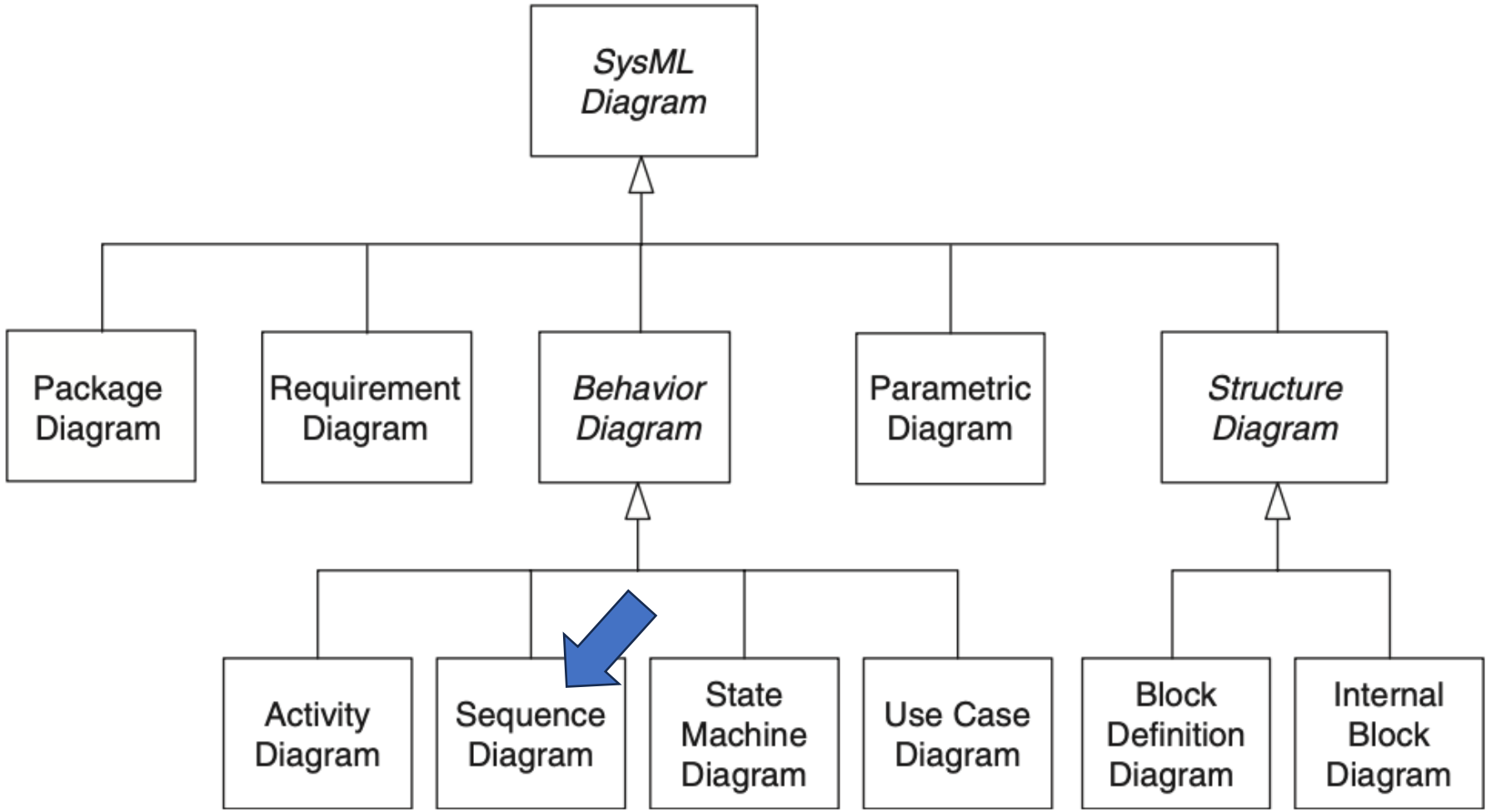


FIGURE 3.1

SysML diagram taxonomy.

Sequence Diagram

MODELING MESSAGE-BASED BEHAVIOR WITH INTERACTIONS – CHAPTER 10

https://developer.ibm.com/articles/the-sequence-diagram/

Message Sequence Charts (MSC)

- MSCs have long been used in standardization and industry for viewing selected message traces.
- Its simplicity and intuitive understanding have made notation quite popular.

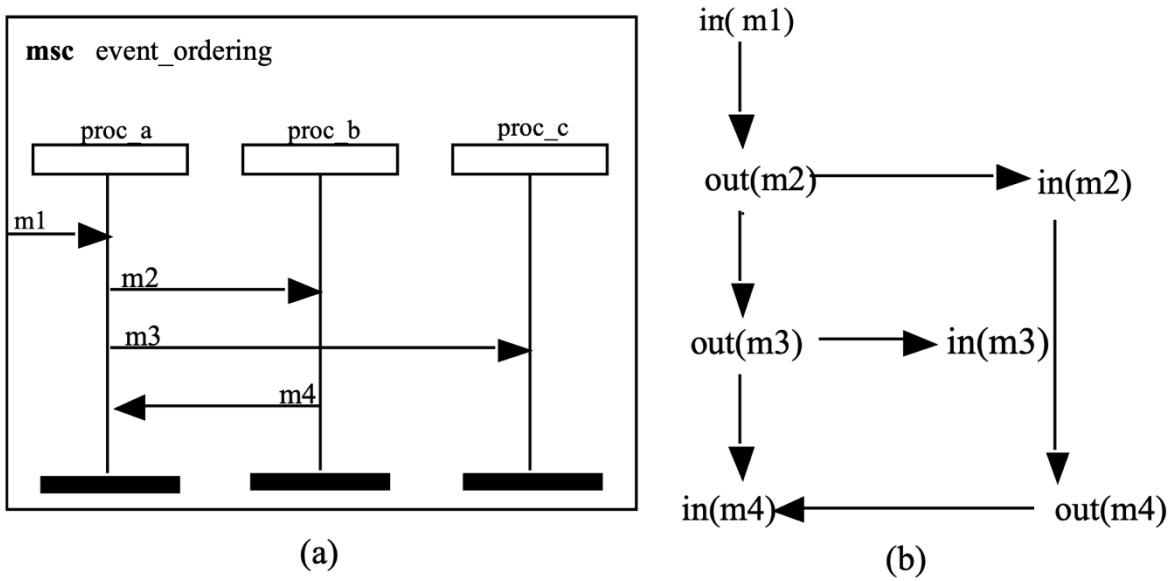


Figure 6 – Message Sequence Chart and corresponding connectivity graph

https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=z.120

Introduction to Sequence Diagram

- Represent the **interaction between elements** in a model as a sequence of message exchanges.
- This representation of behavior is useful when **modeling service-oriented concepts**, when one part of a system requests services from another part.
- A sequence diagram can be written as
  - **a specification of how the parts of a system should interact**, and it can also be used as a
  - **record of how the parts of a system interact**.

Example

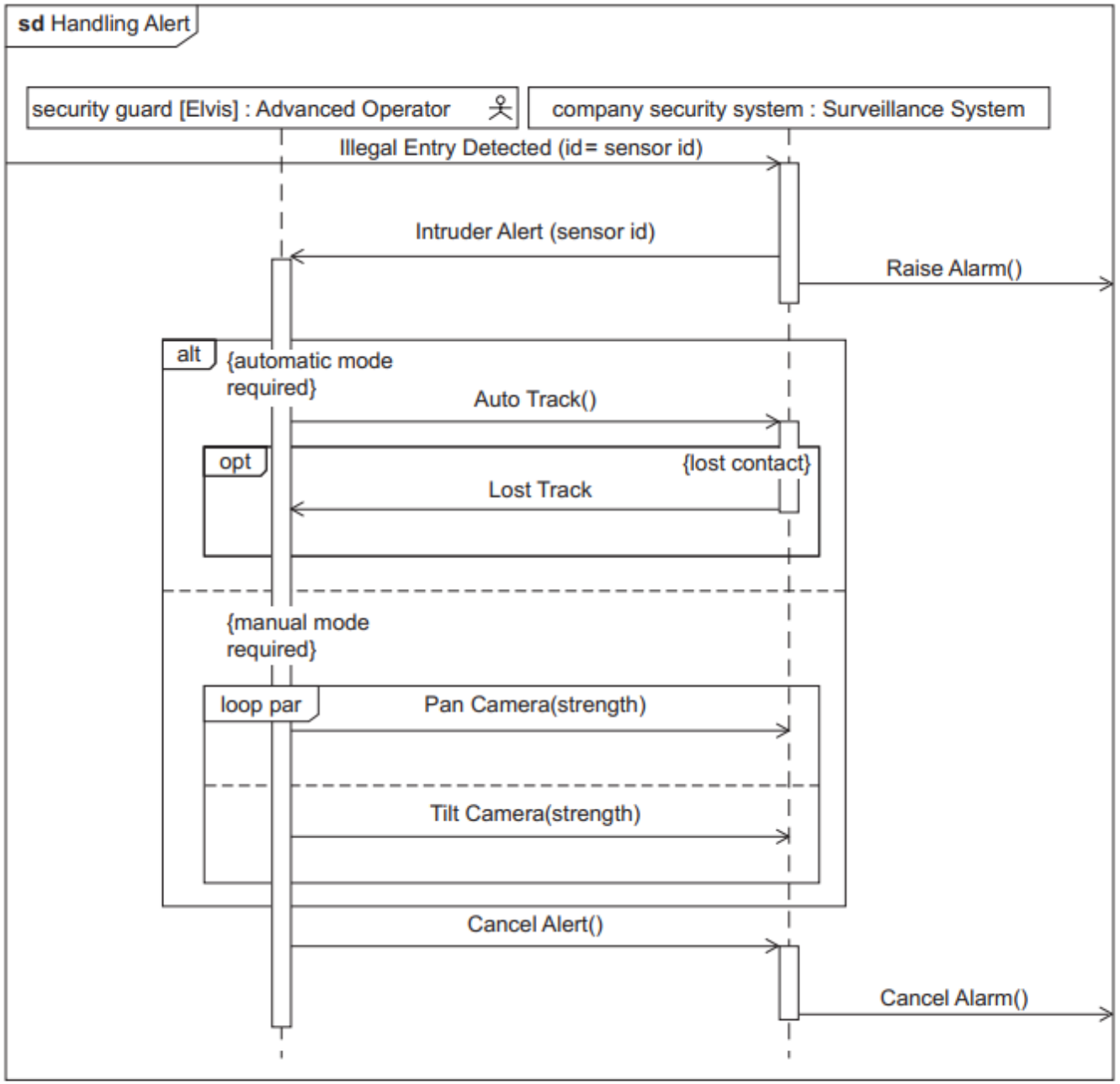


FIGURE 10.1

An example sequence diagram.

Sequence Diagram

- A sequence diagram represents an interaction.
- The full diagram header for a sequence diagram is as follows:

- `sd [interaction] interaction name [diagram name]`

- The diagram type for a sequence diagram is sd, and the type of model element that corresponds to your frame can only be Interaction.

Lifelines

Lifelines

- The main structural feature of an interaction is the lifeline. A lifeline represents the relevant time of a property of the interaction's proprietary block.
- When a block performs an interaction, each lifeline denotes an instance of some part of the block.
- *A lifeline is shown using a rectangle (the head) with a dashed line descending from its base (the tail). The header contains the name and type, if applicable, of the property represented, separated by a colon*.

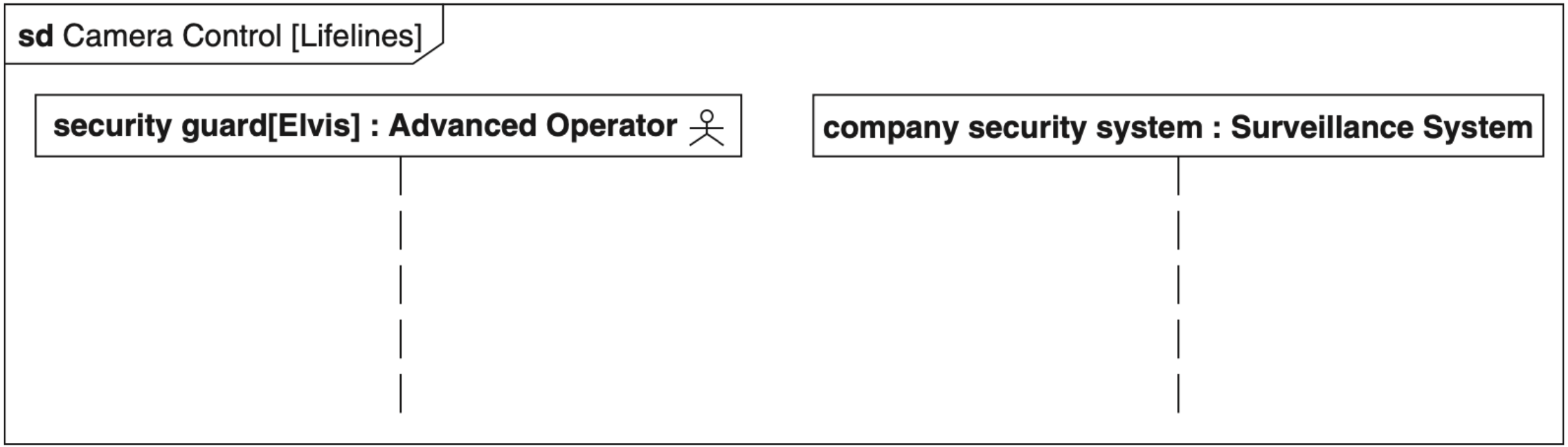


FIGURE 10.3

An interaction with lifelines.

Messages / Events

Messages

- The entity that sends asynchronous messages continues to execute after the message is sent, however if a synchronous message is sent it must wait for the response from execution.
  - An open arrowhead means an asynchronous message.
  - A closed arrowhead means a synchronous message.
  - An arrowhead on a dashed line shows a response message.

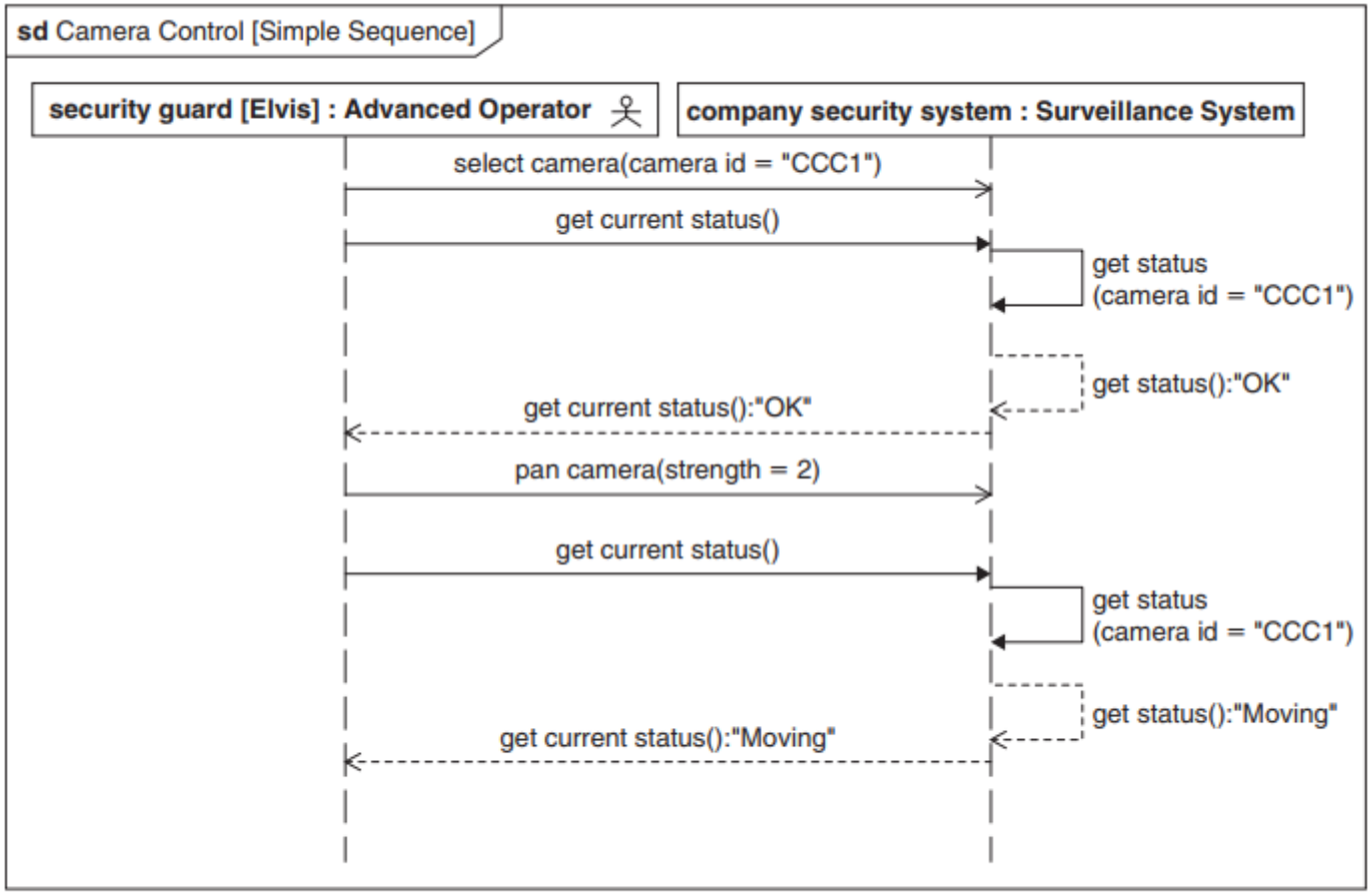
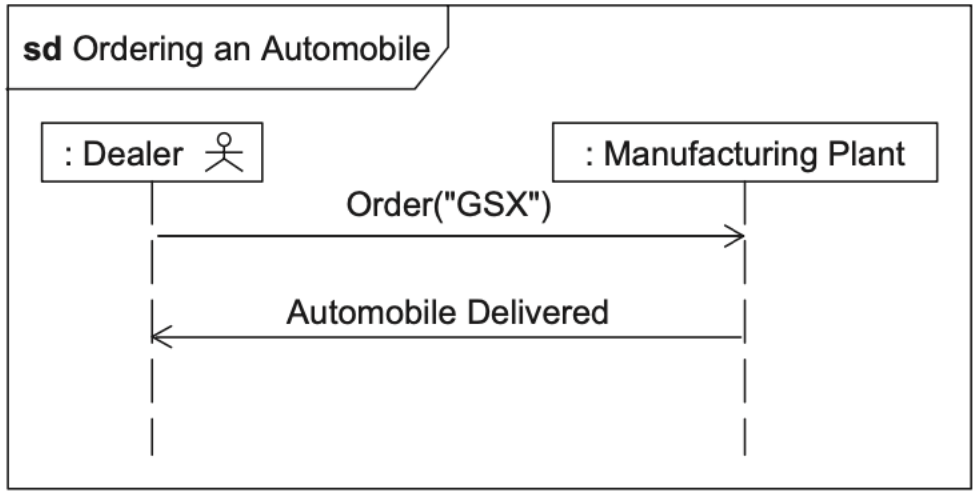


FIGURE 10.5

Synchronous and asynchronous messages exchanged between lifelines.

Event Sequence Description

- When an interaction is executed, the set of time-ordered occurrences is called a trace.
- A comparison of the order and structure of the actual occurrences determines whether the trace is consistent with the interaction.
- Messages can be exchanged between entities.
- A message can be sent from a timeline to itself to represent a message that is sent and received by the same entity.



Executions

- Receiving a message from a lifeline can trigger the **execution of a behavior**.
  - Activations are vertically overlapping rectangular symbols on lifelines.

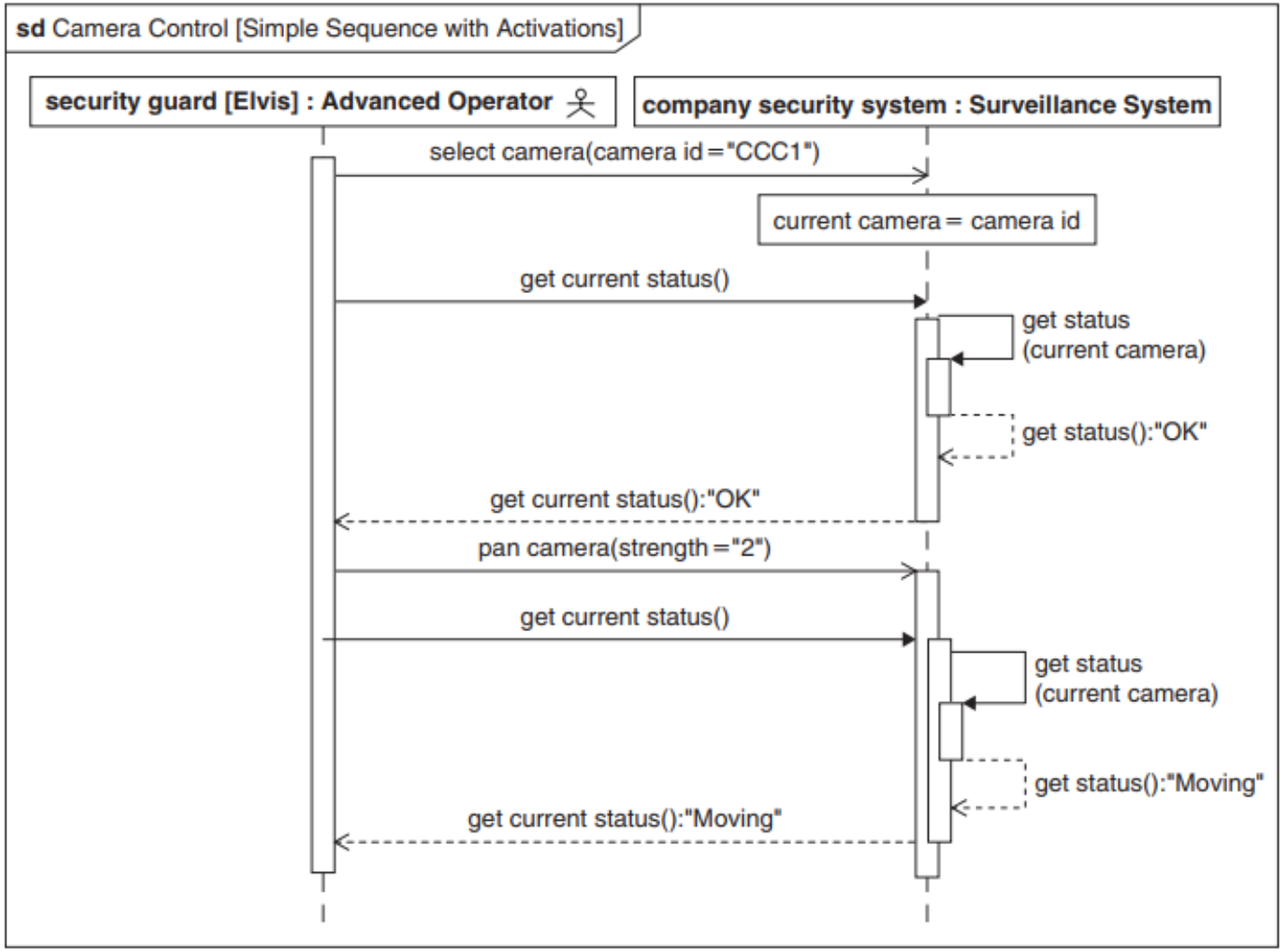


FIGURE 10.7

Lifelines with activations.

Creation and destruction

- A create message represents the creation of an instance and is therefore the first occurrence on the timeline.
- A destroy message ends in a special type of occurrence called a destruction occurrence, which must be the last occurrence in a lifeline.

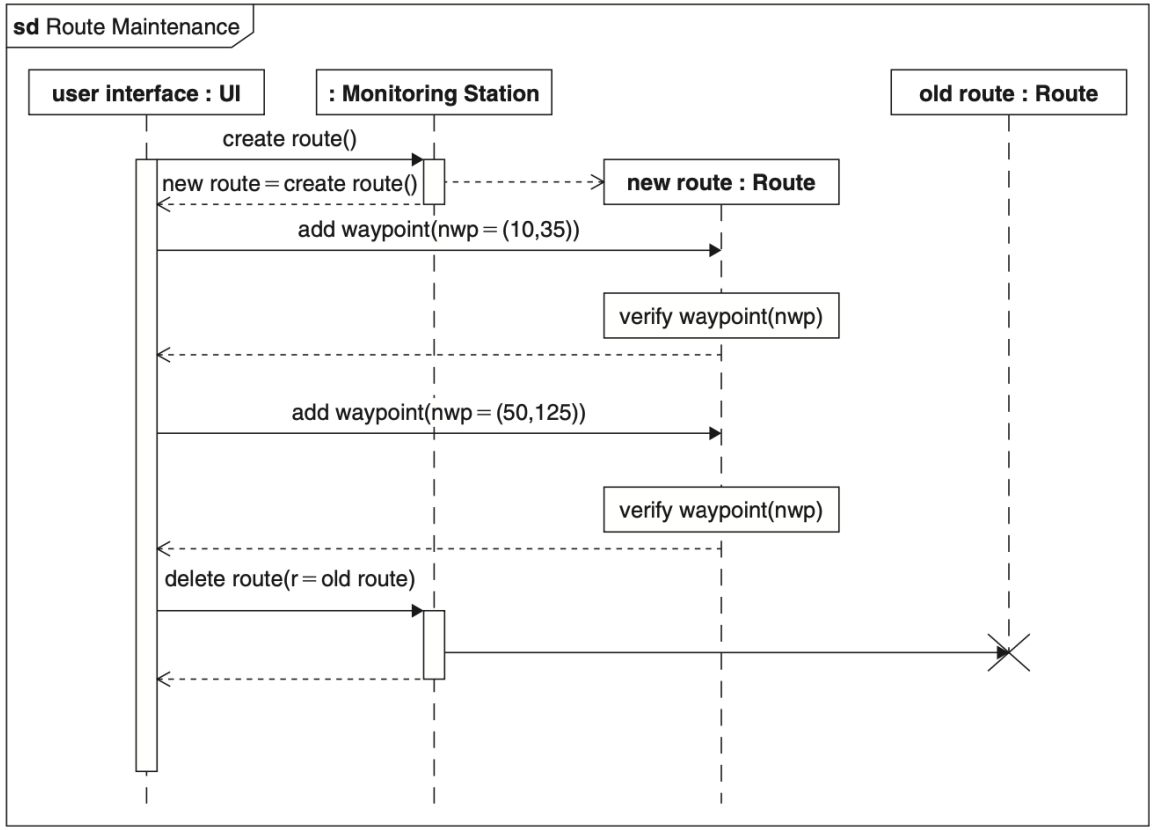


FIGURE 10.8

Create and destroy messages.

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 spent 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.

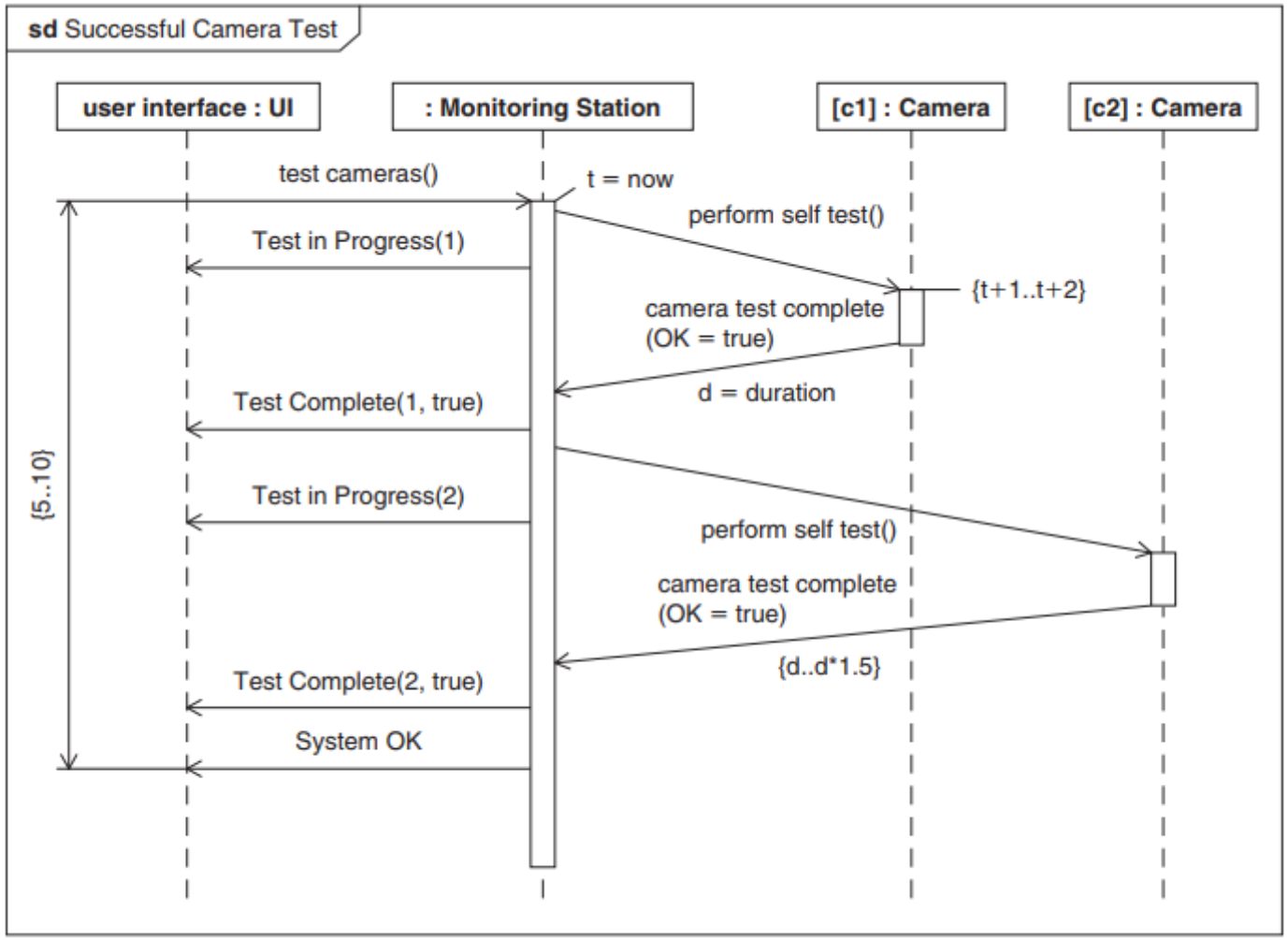


FIGURE 10.10

Representing time on a sequence diagram.

Fragments

Using fragments

- The most basic form of an interaction is a weak sequence of occurrences—usually read from the top down of the sequence diagram**. However, more complex patterns of interaction can be modeled using constructs called combined fragments.**
- A combined fragment consists of an interaction operands and its operands.
  - An interaction operand defines a group of messages and cases that span one or more lines.
  - Each operand has a guard containing a constraint expression that indicates the conditions under which it is valid for the operand to start execution.

Basic interaction operators

- **Seq** - weak sequencing. Weak sequencing is the standard form of sequencing for all operands, it is rarely explicitly indicated.
- **Par** - an operator in which operands can occur in parallel. There is no implicit 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 hold on each operand is evaluated before the selection, and if the hold on one of the operands is valid, it is selected.
- **Opt** - a unary operator that is equivalent to an ALT with only one operand. This implies that the operand is either executed or ignored, depending on the validity of the guard.
- **Loop** - an operator in which the trace represented by its operand repeats until its termination constraint is met.

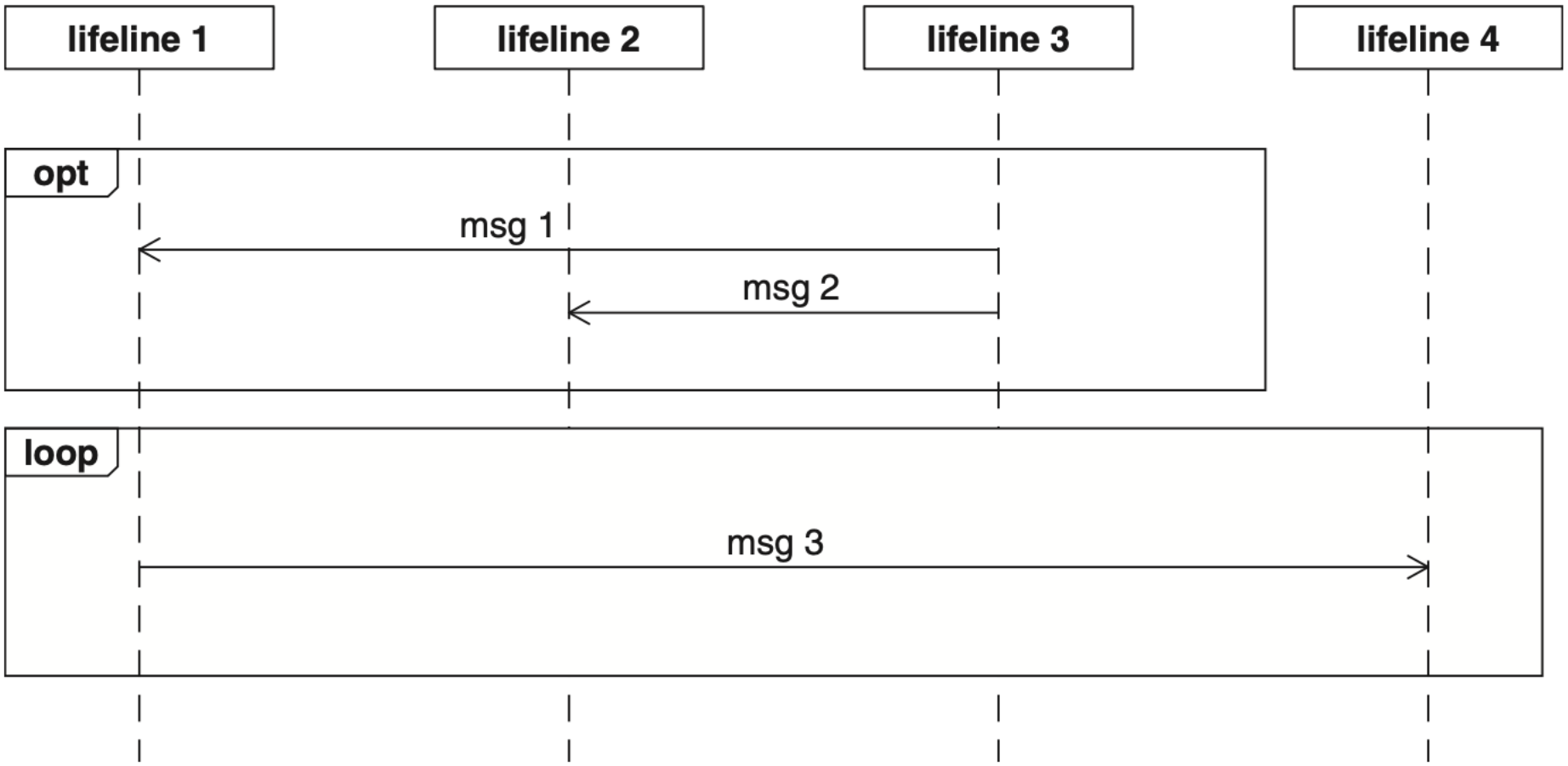


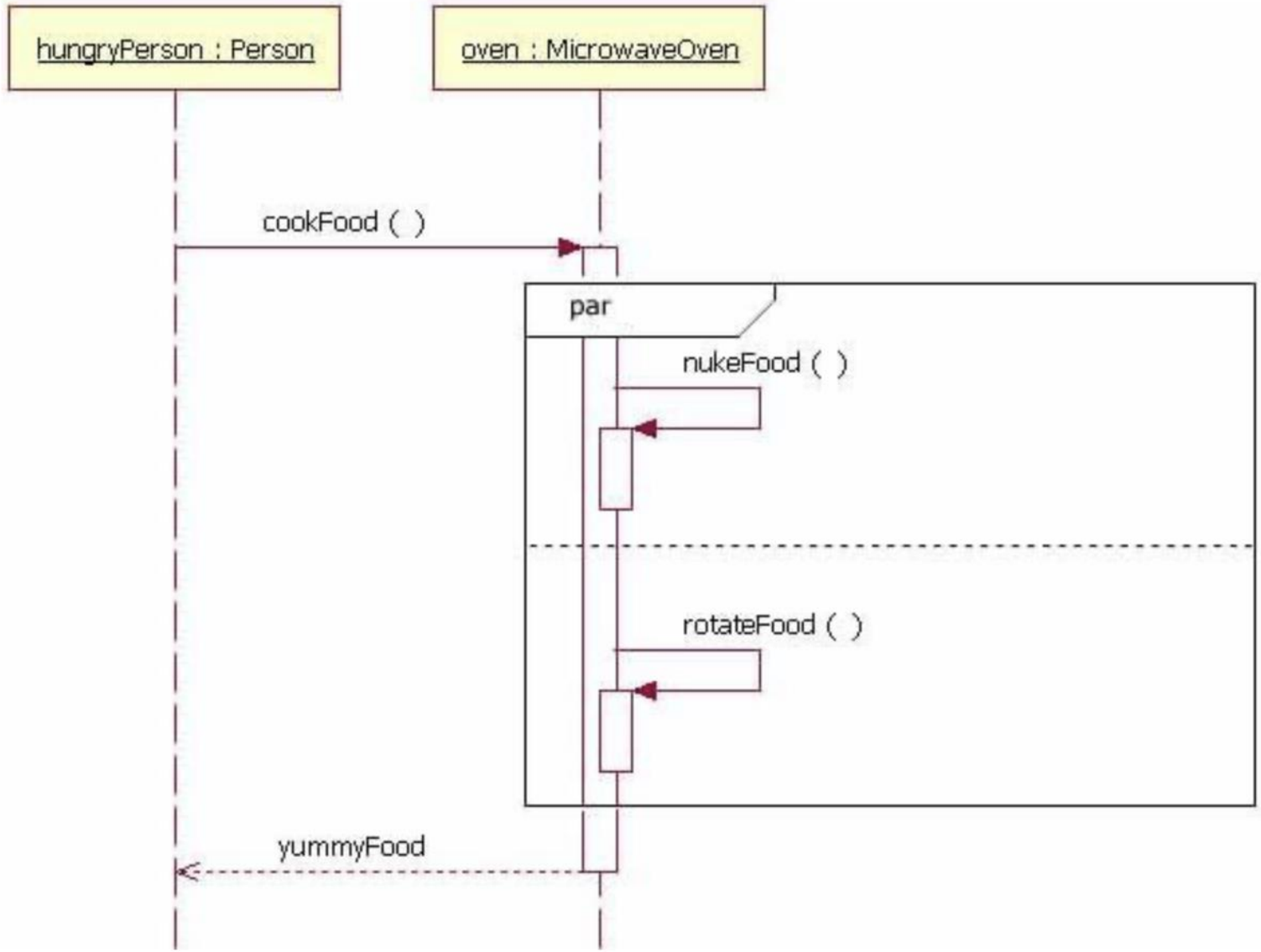
FIGURE 10.11

Example of overlapping and nonoverlapping lifelines.

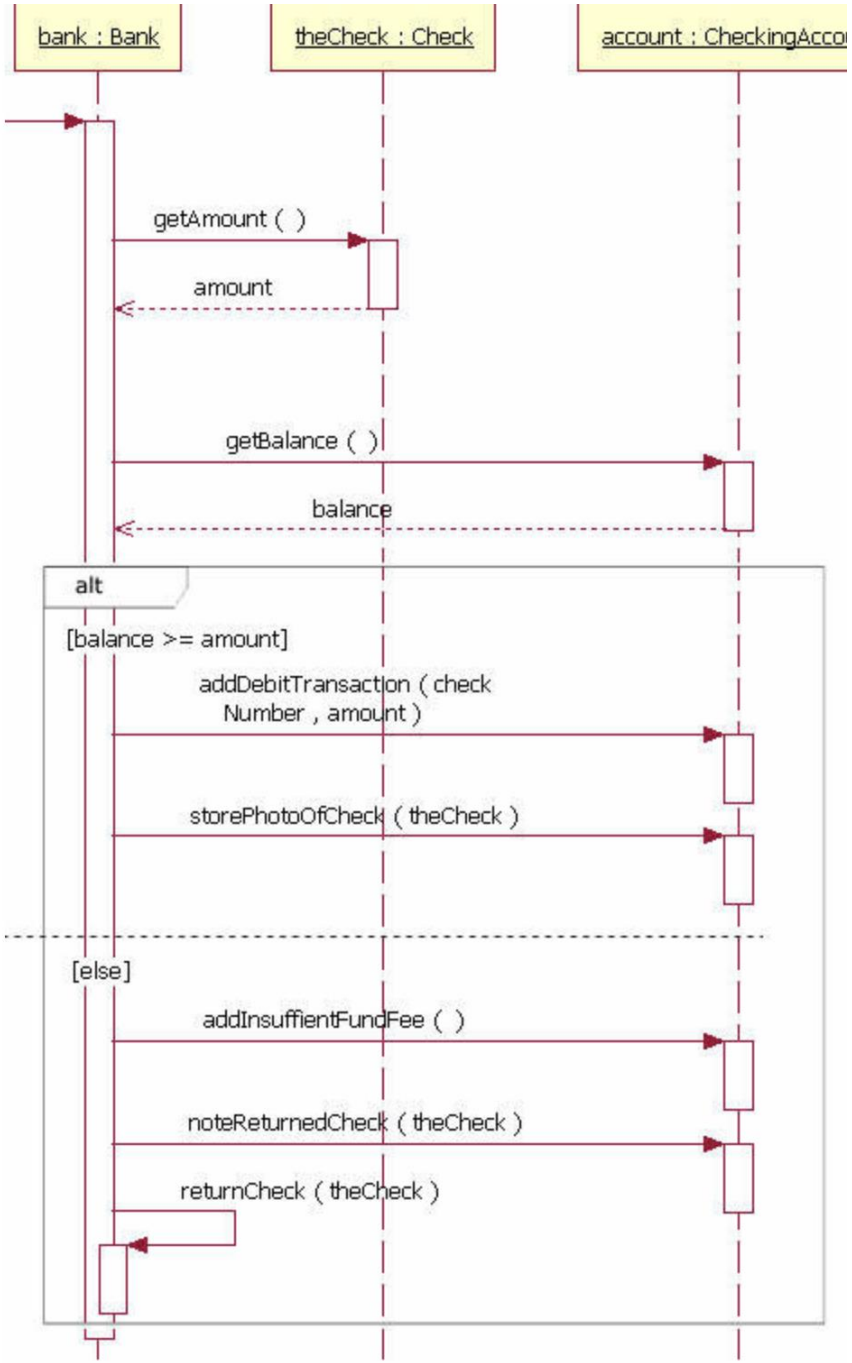
Note on alt, par e loop

- Alt and even operators **have multiple partitions separated by dashed lines** that correspond to their operands.
  - Other operators have **only a single partition**. Messages, activations, and possibly other fragments combined are nested in each operand.
  - *Guards are shown in brackets superimposed on the line to which it is linked.*
- When an operator has a single operand that is a combined fragment, the frames of the operator and operand can be merged into one.
  - *The merged frame label is used to indicate all operators, such as loop pair.*

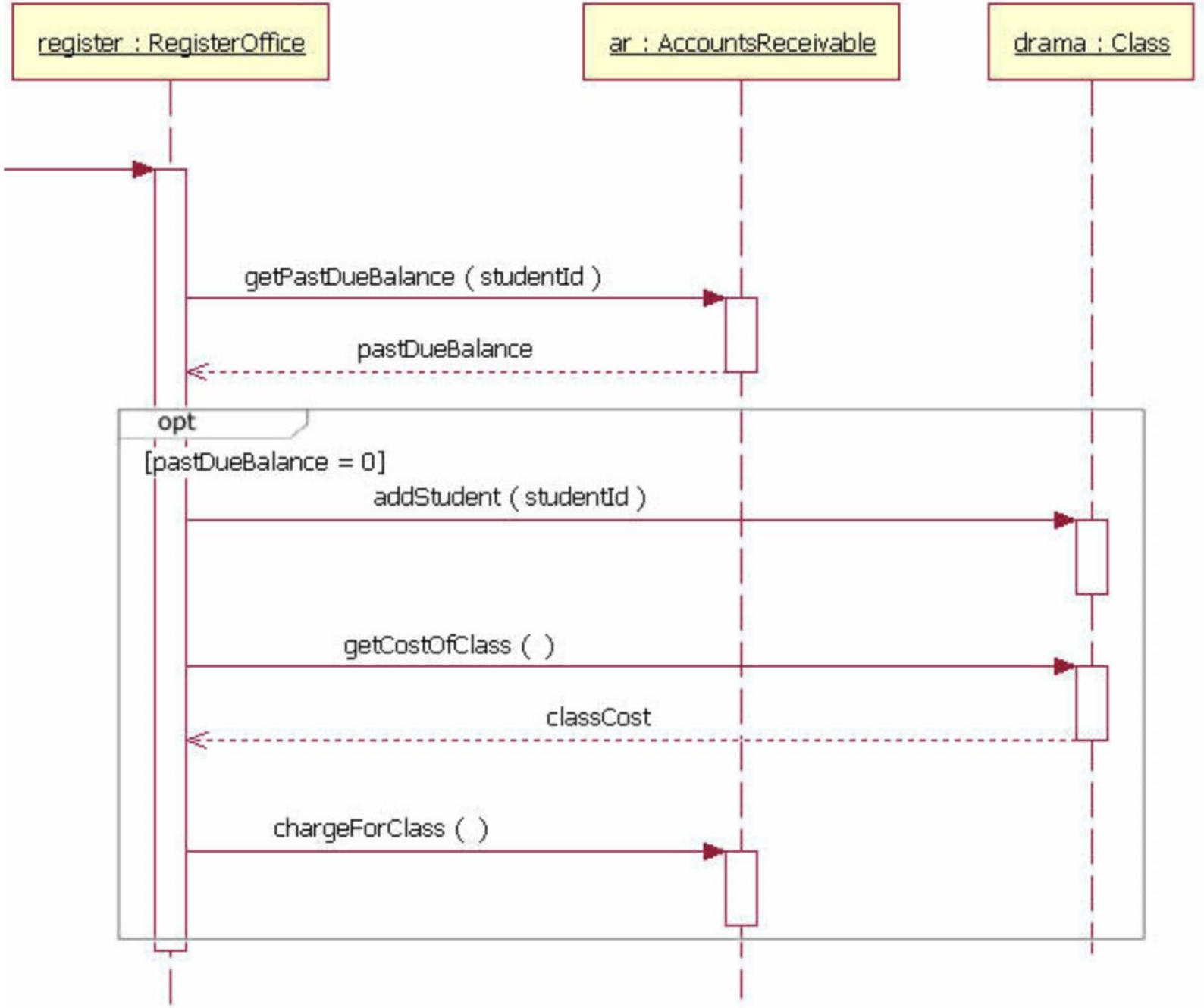
Parallel


hungryPerson : Person
oven : MicrowaveOven
cookFood ( )
par
nukeFood ( )
rotateFood ( )
yummyFood

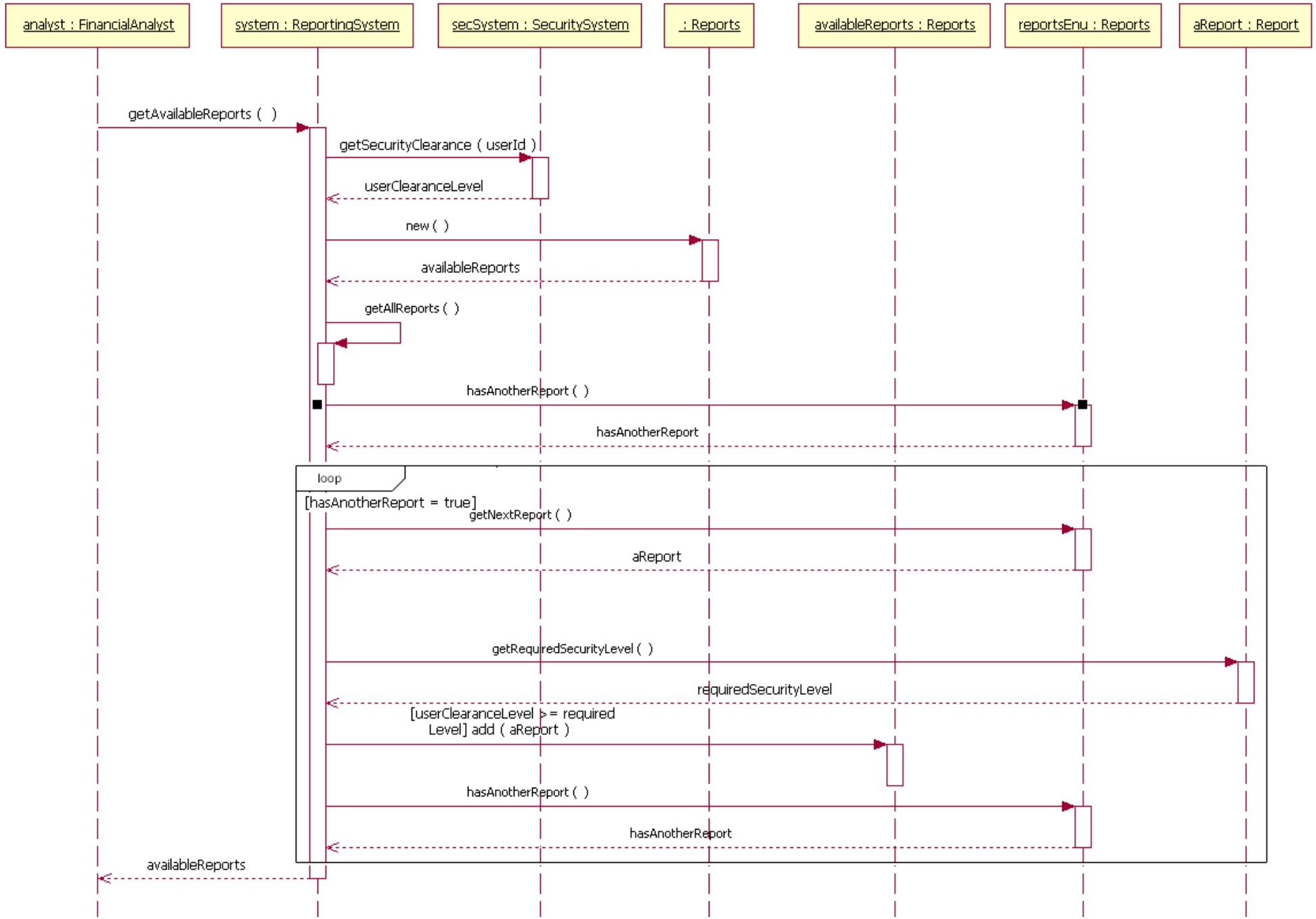
Alternatives



Optional


register : RegisterOffice
ar : AccountsReceivable
drama : Class
getPastDueBalance (studentId)
pastDueBalance
opt
[pastDueBalance = 0]
addStudent (studentId)
getCostOfClass ( )
classCost
chargeForClass ( )

Loop


analyst : FinancialAnalyst
system : ReportingSystem
secSystem : SecuritySystem
: Reports
availableReports : Reports
reportsEnu : Reports
aReport : Report
getAvailableReports ( )
getSecurityClearance (userId)
userClearanceLevel
new ( )
availableReports
getAllReports ( )
hasAnotherReport ( )
hasAnotherReport
loop
[hasAnotherReport = true]
getNextReport ( )
aReport
getRequiredSecurityLevel ( )
requiredSecurityLevel
[userClearanceLevel >= required Level] add (aReport)
hasAnotherReport ( )
hasAnotherReport
availableReports

Complex Interactions

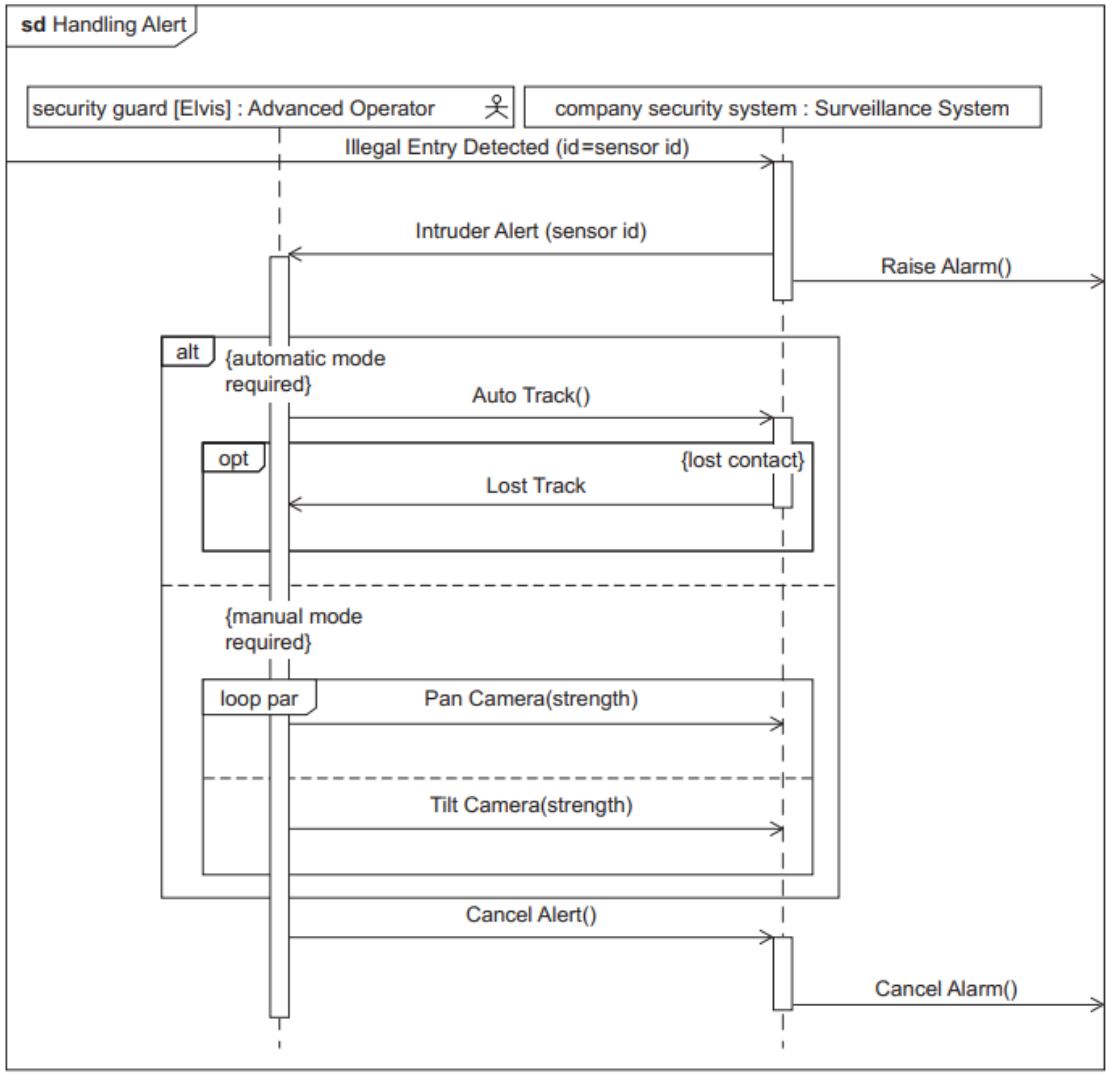


FIGURE 10.12

Complex interactions described using interaction operators.

Reference to other timelines

- To support large-scale uses of interactions, an interaction might include an **interaction usage that references an interaction described in another sequence diagram**.
- Interaction uses can be nested, because one referenced interaction can in turn reference another.
  - This feature significantly increases the scalability of interactions.
  - It also makes reuse easier, since an interaction can be used (i.e., referenced) by more than one using interaction.

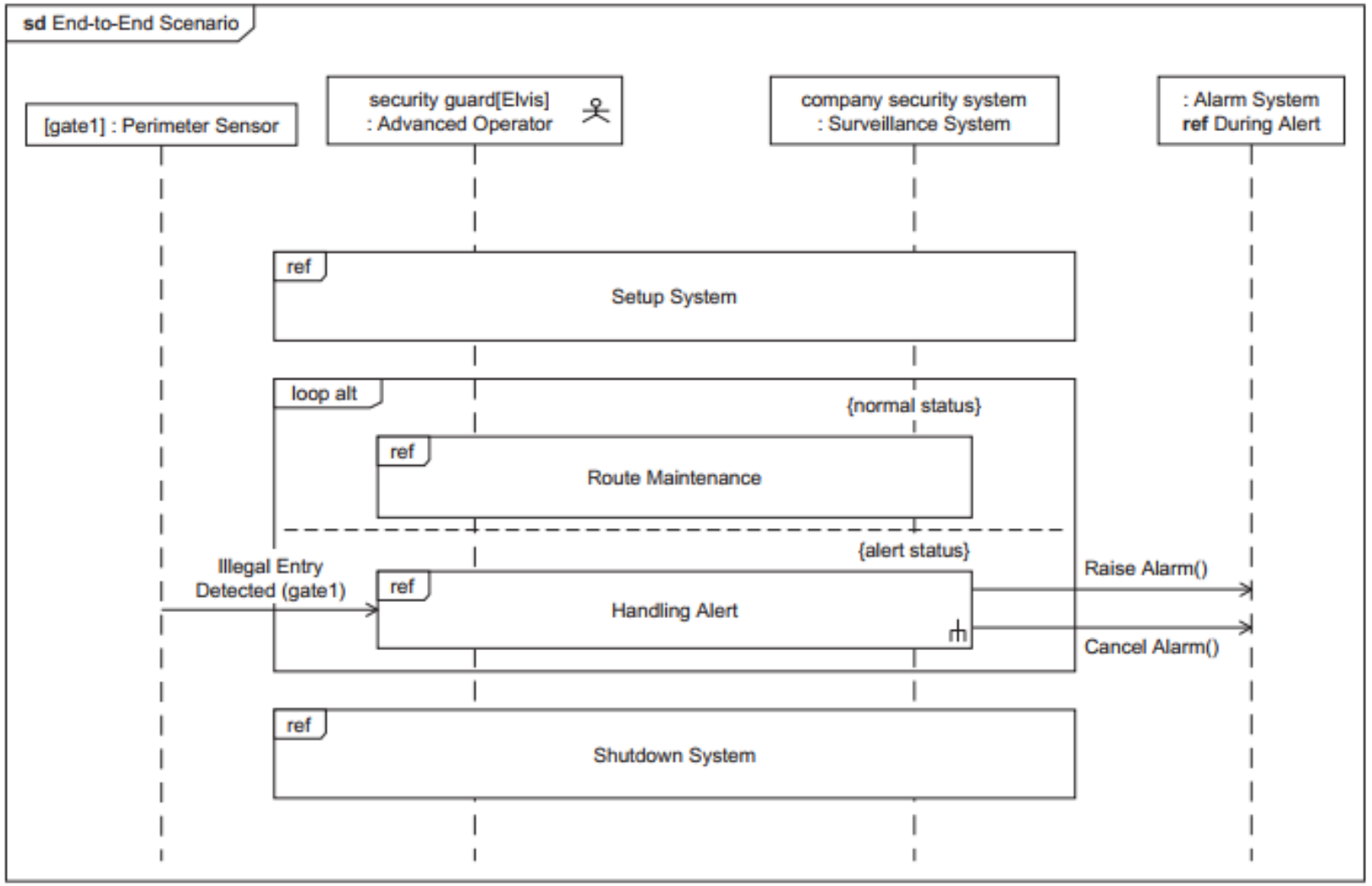
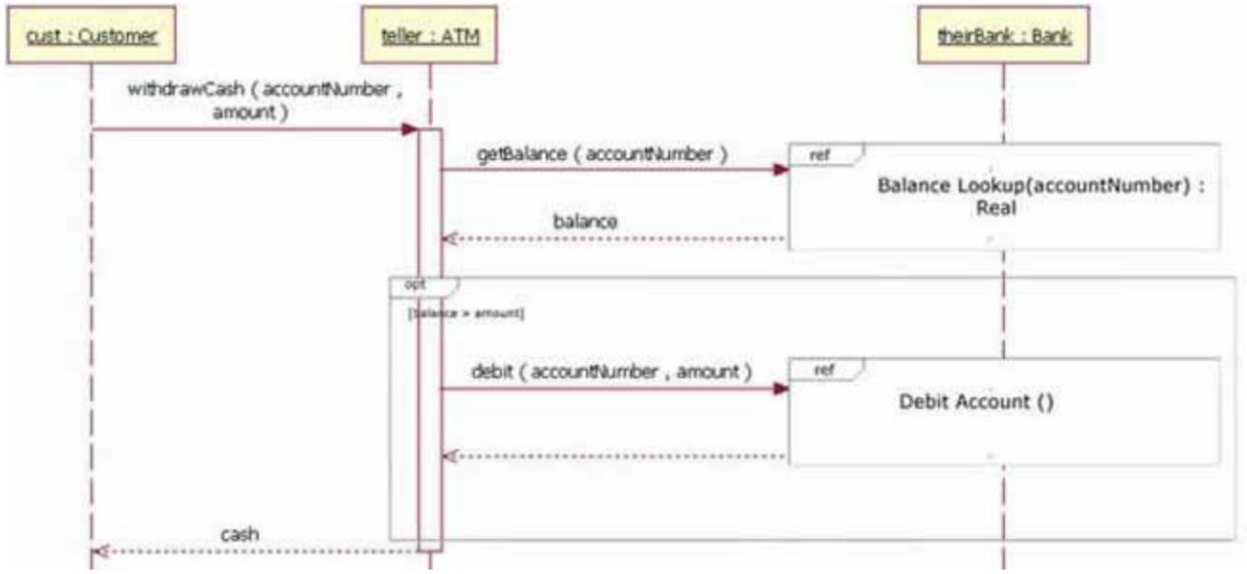


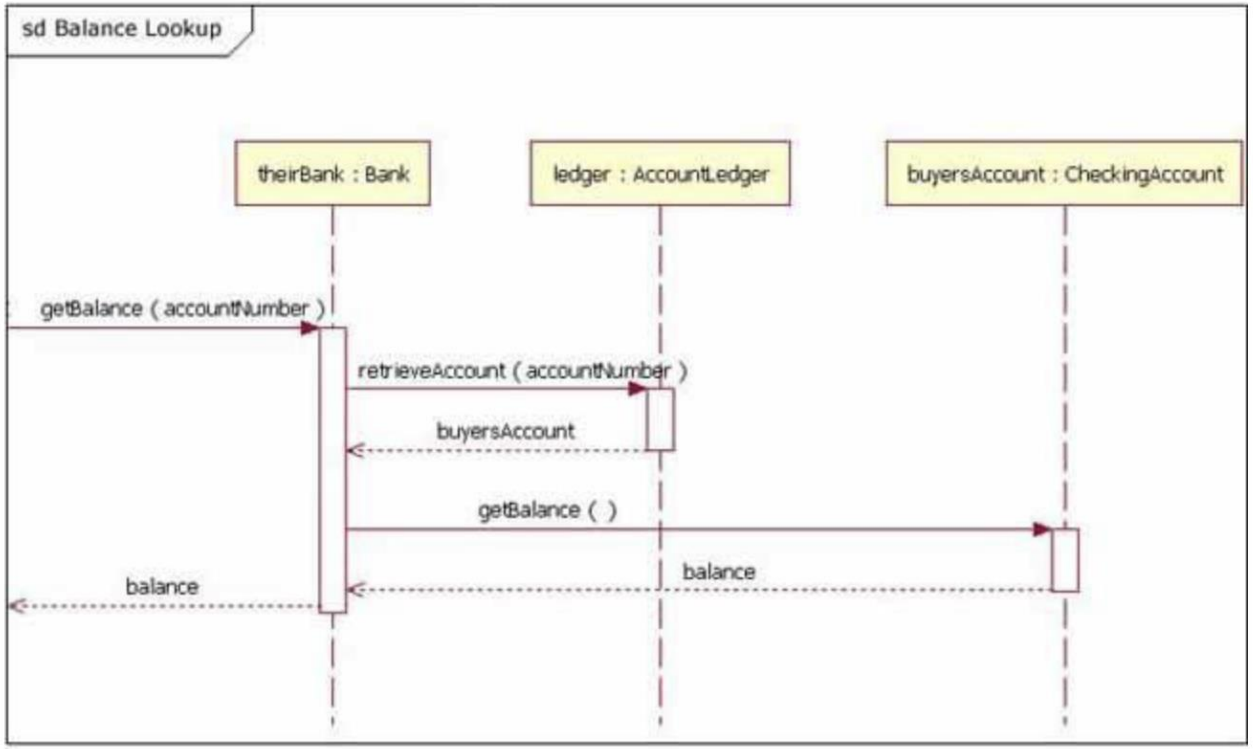
FIGURE 10.15

Reference to another interaction.

Gates

- **Gates** can be an easy way to model the **passage of information between a sequence diagram and its context**.
- A gate is merely a message that is illustrated with **one end connected to the edge of the sequence diagram frame** and the other end connected to a lifeline.





(Start) States

- It is often useful to augment the representation by adding constraints on the required state of a lifeline at a given point in a sequence of hits. This can be achieved by using a state on a timeline.

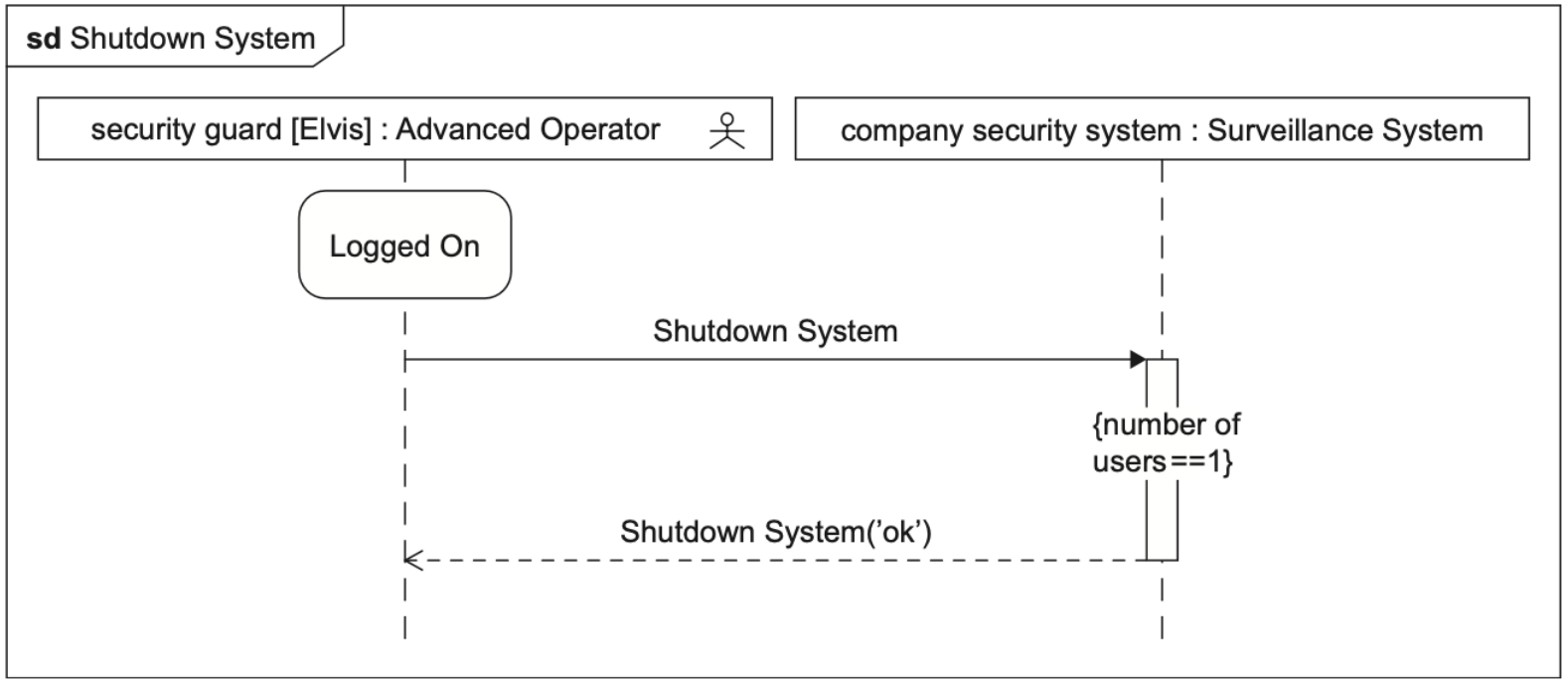


FIGURE 10.14

State invariants.

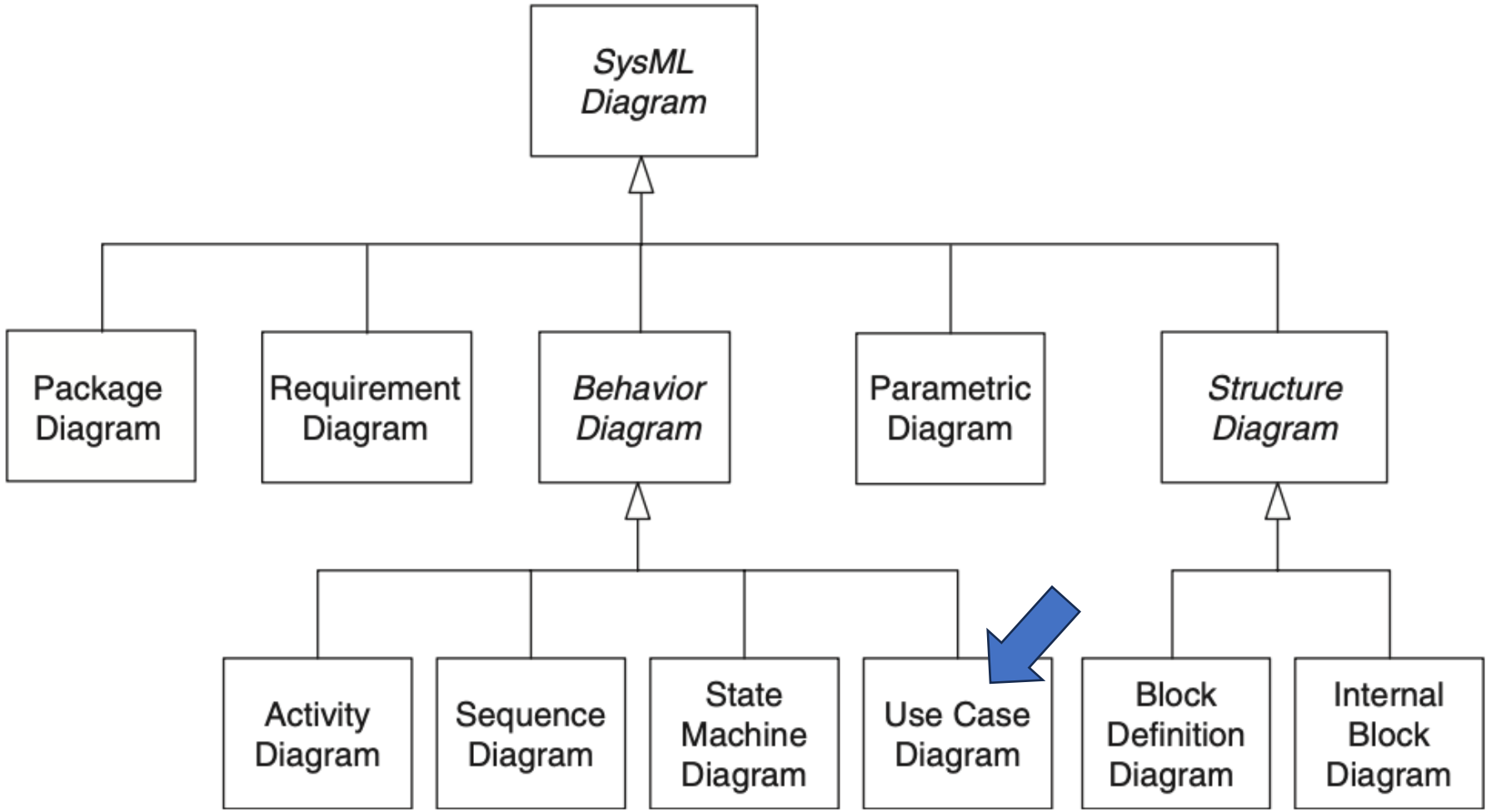


FIGURE 3.1

SysML diagram taxonomy.

Use Case Diagrams

Modeling Functionality with Use Cases – Chapter 12

Introduction

- Use cases describe the **functionality of a system** in terms of **how it is used to achieve the goals** of its various users.
- The users of a system are described by actors, who can represent **external systems or humans interacting with the system**.

- *Model the high-level functionality of a system with use cases.*

Introduction

- Use cases have been seen as a **mechanism to capture the requirements** of the system in terms of system uses.

- Different methodologies apply use cases in different ways:
  - For example, some methods require a text description for each case, which may include pre- and post-conditions and primary, alternate, and exceptional flows.
  - Use cases are often elaborated with detailed descriptions of their behavior, using activities, interactions, and/or state machines.

The Use Case Diagram

- In a use case diagram, the frame corresponds to a package, model, model library, or block, and the contents of the diagram describe a set of actors and use cases and the relationships between them.

- The full diagram header for a use case diagram is as follows:
  - *uc [model element kind] model element name [diagram name]*

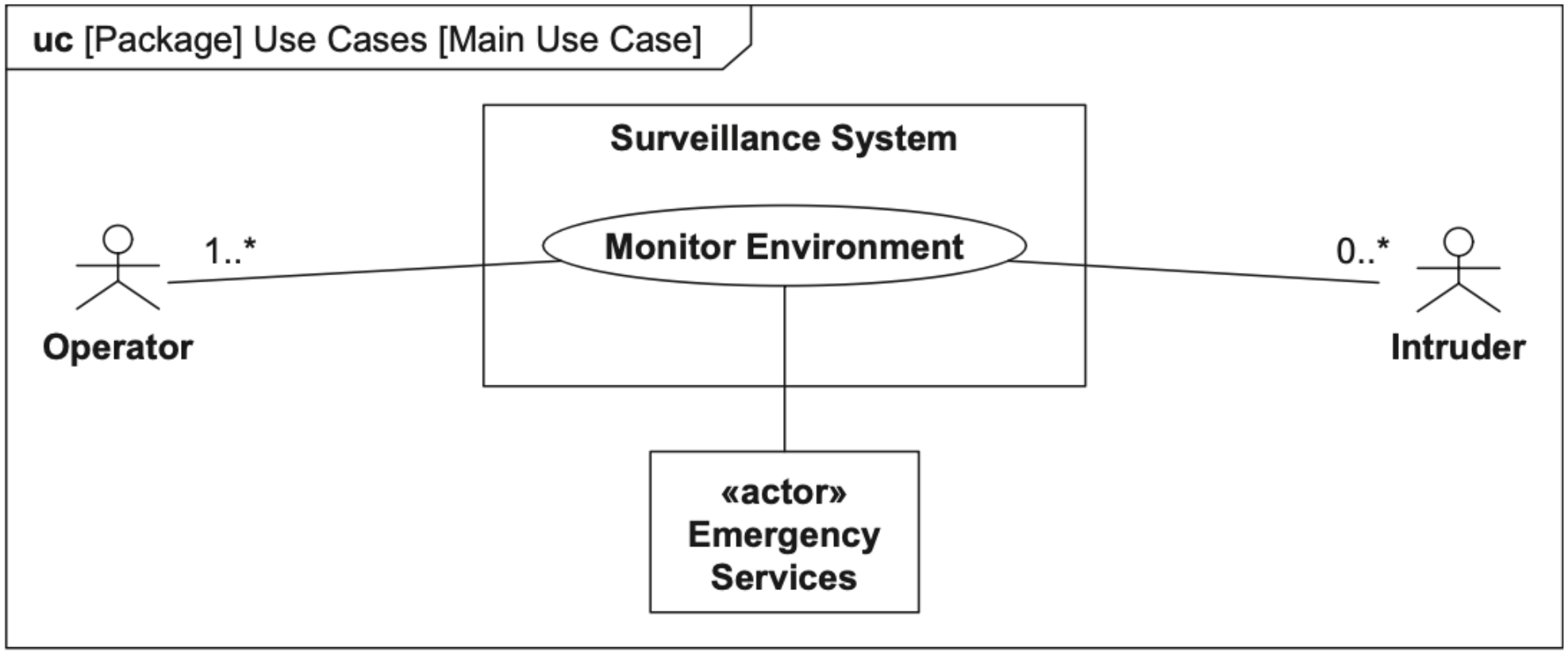


FIGURE 12.1

Example use case diagram.

Actor (or stakeholders?)

Definition of actor

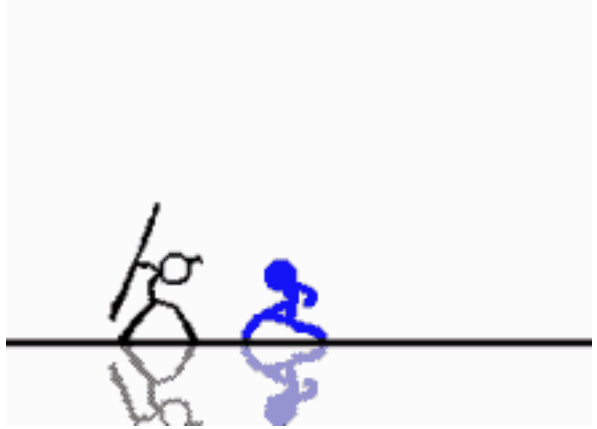
- An **actor** is the “entity” to play the role of a human being, an organization, or any external system that participates in the use of some system.
- Actors can interact directly with the system or indirectly through other actors.

- *It should be noted that "actor" is an umbrella term.*
  - *An actor that is external to one system may be internal to another.*

Actors

- An actor is shown as a strawman with the actor's name underneath or as a rectangle containing the actor's name below the keyword «actor».

- *The choice of symbol depends on the tool and the method being used*.



Classification: Generalization/Specialization

- The actor's rating is represented using the standard SysML generalization symbol - a line with an empty triangle at the general end.
- **Actors can be sorted using the standard generalization ratio**.
- The actor rating has a similar meaning to the rating of other classifiable model elements.
  - For example, a specialized actor participates in all the use cases in which the more general actor participates.

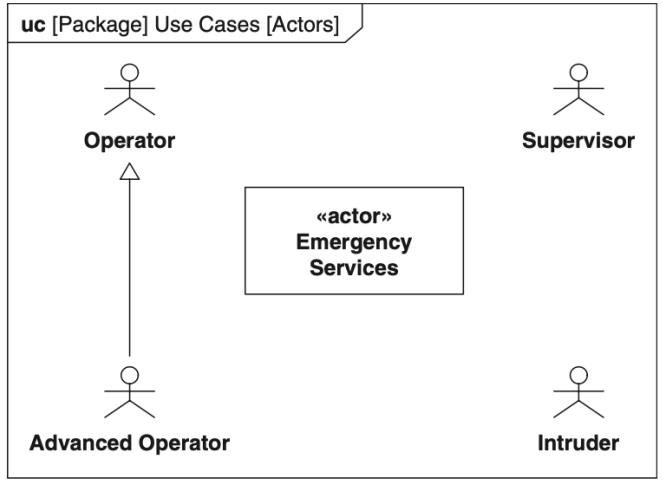


FIGURE 12.2
Representing actors and their interrelationships on a use case diagram.

A use case

Use Cases

- A use case describes **the goals of a system** from the perspective of the system's users.
- The objectives are described in terms of the **functionality that the system must support**.
- Typically, the use case identifies the use case purpose(s), a primary usage pattern, and several alternative uses (variants).

Use Cases - System

- The system that provides functionality in support of use cases is called a *System under Consideration* and usually represents a system that is being developed.
- The system under consideration is sometimes called the **subject** and is represented by a block.

Use Cases - Scenarios

- A use case can encompass one or more scenarios that correspond to how the system interacts with its actors in different circumstances.

Use Cases – Actor Relationship

- The actors are related to the use cases of **communication paths**, which are represented as **associations**, with some restrictions.
- Membership ends can have multiplicities, where multiplicity at the actor's end describes the number of actors involved in each use case.

Use cases graphically:

- A use case is shown as **an ellipse** with the name of the use case inside it.
- **Associations between actors and use cases are shown using standard association notation.**

- The **subject** of a set of use cases can be shown as a **rectangle wrapping around the use cases**, with the subject name centered at the top.

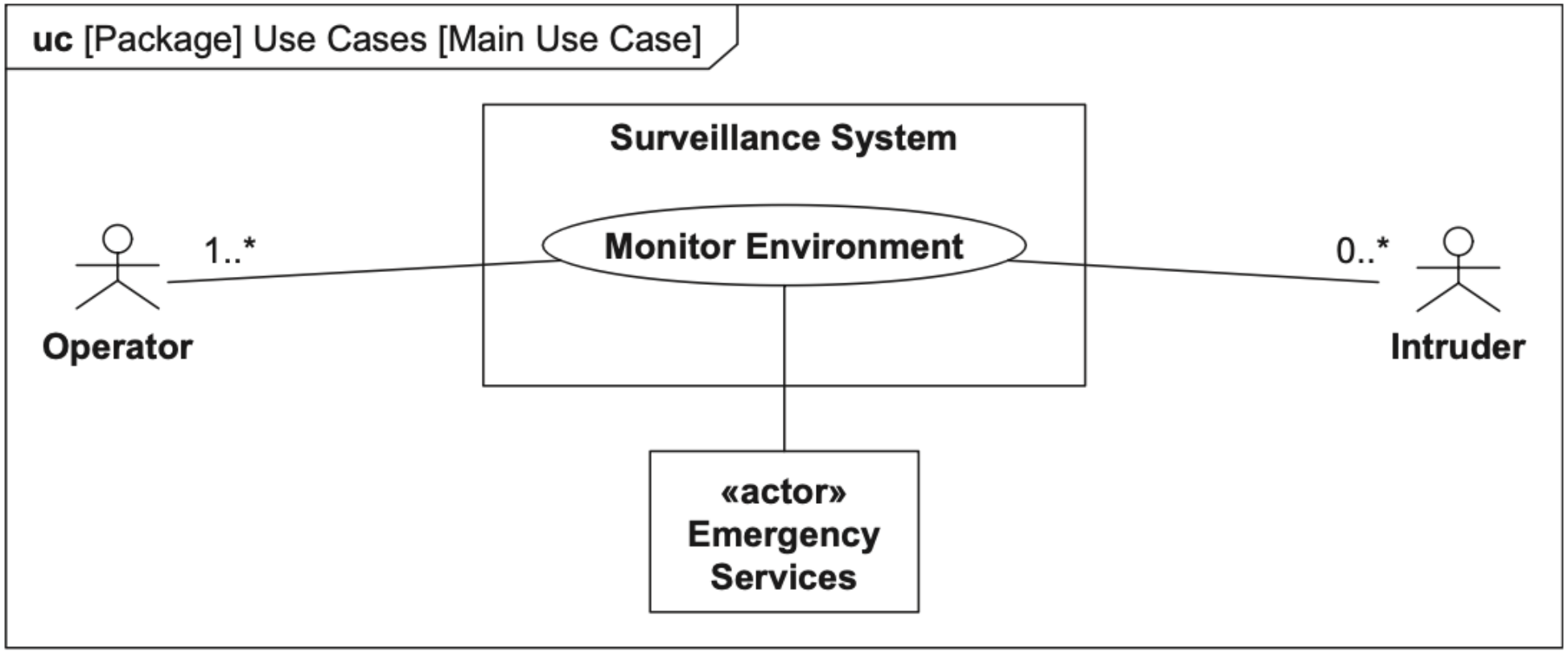


FIGURE 12.3

A use case and the actors that participate in it.

Relationships Between Use Cases

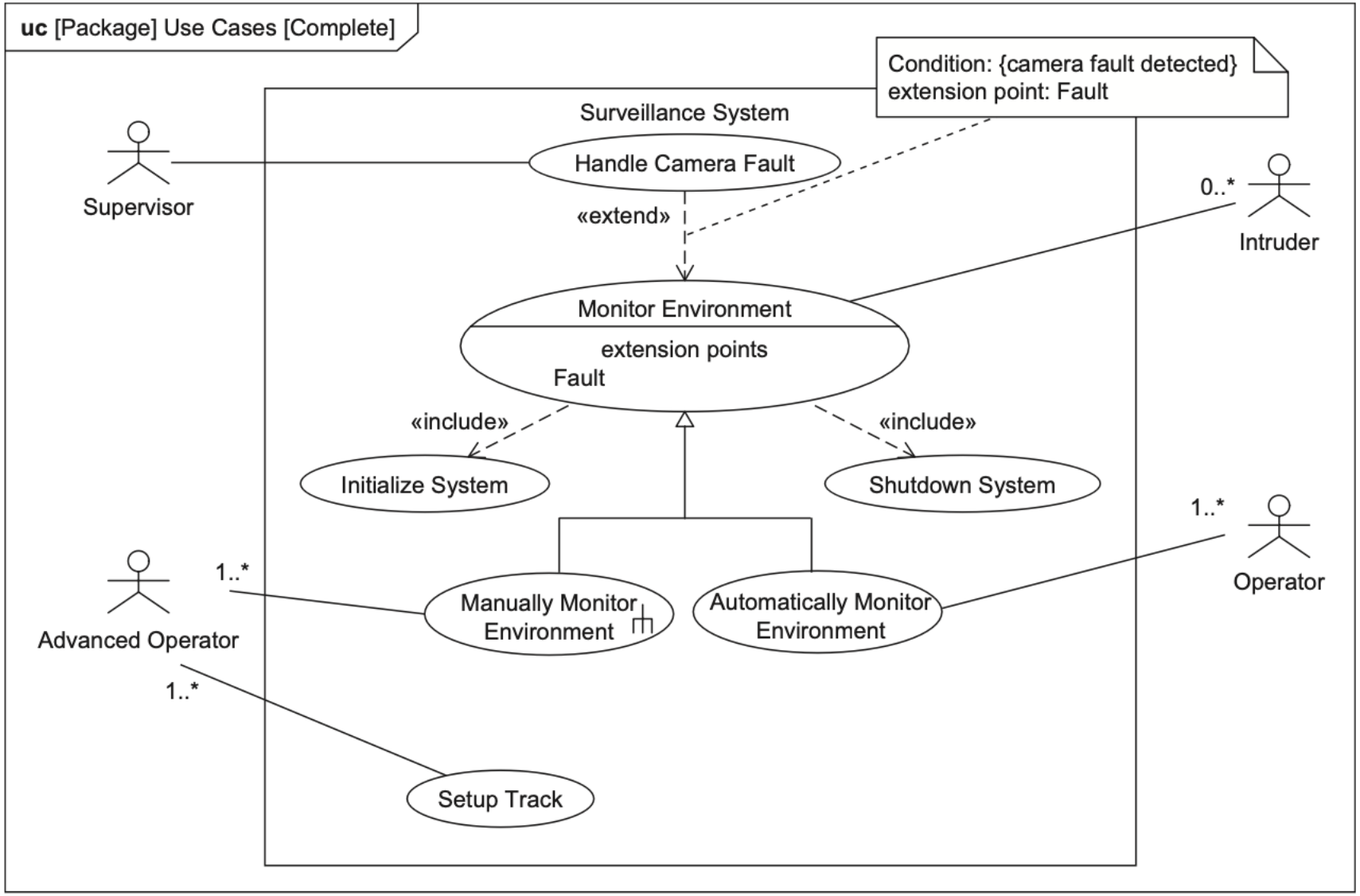


FIGURE 12.4

A set of use cases for the *Surveillance System.*

Inclusive relationship

- The include relationship allows a use case, known as the base use case, to include the functionality of another use case, called the included use case.
- The included use case always runs when the base use case runs.

- *It is implicit in the definition of inclusion that any participant in a base use case can participate in an included use case, so an actor associated with a base use case does not need to be explicitly associated with any included use case.*

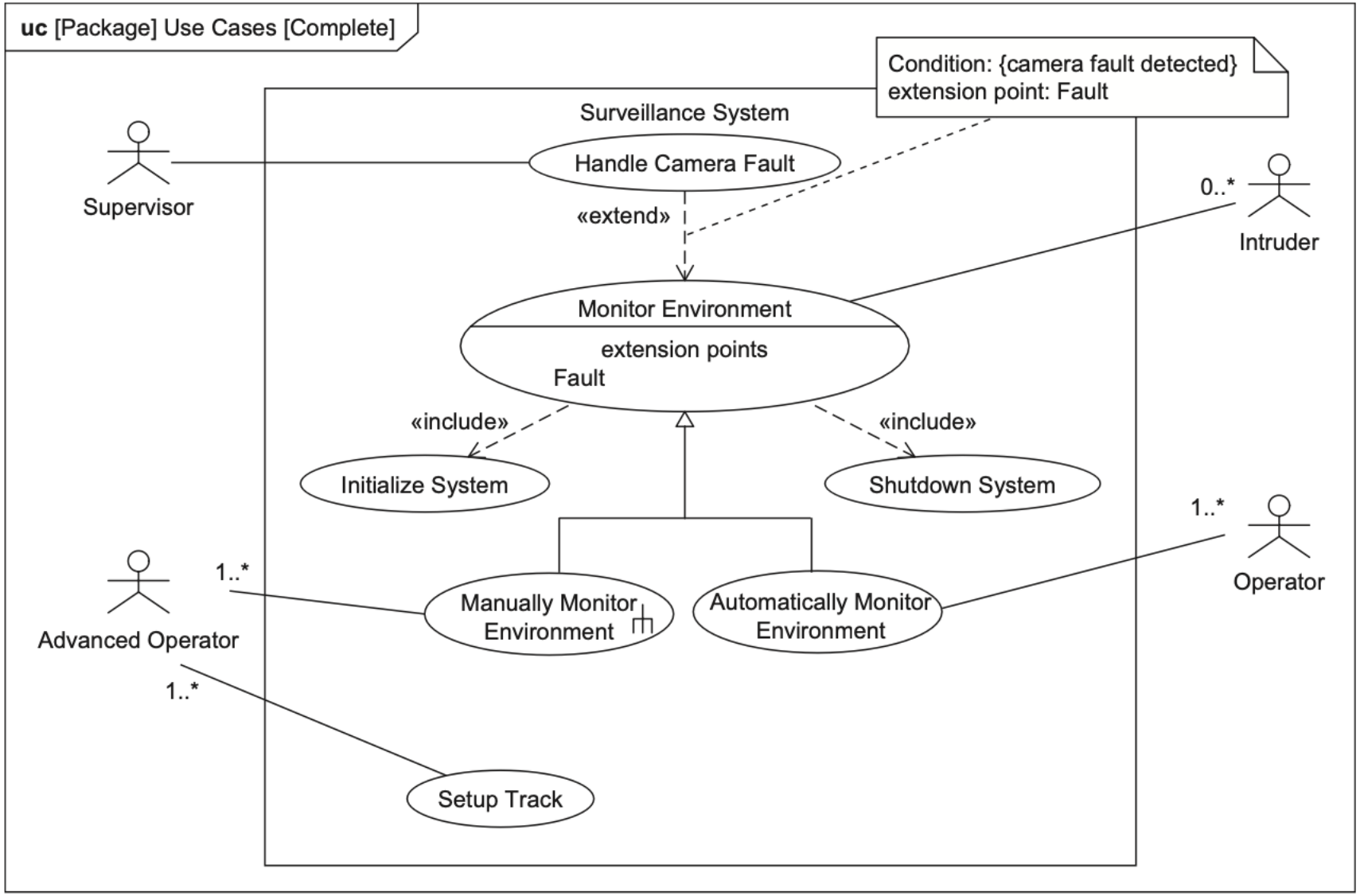


FIGURE 12.4

A set of use cases for the *Surveillance System.*

Subtle observation: it is not functional decomposition!!

- The included use cases **are not intended to represent a functional breakdown of the base use case**, but rather to describe common functionality that can be included by other use cases.

- *In a functional decomposition, the lower-level functions represent a complete decomposition of the higher-level function. On the other hand, a base use case and its included use cases often describe different aspects of the required functionality.*

Extension relationship

- A use case can also extend a base use case by using the **extension relationship.**
- The **extended use case** is a fragment of functionality that is not considered part of the functionality of the base use case.

- *It often describes some exceptional behavior in the interaction, such as error handling between the subject and the actors that do not directly contribute to the goal of the base use case*

Extension does not generate dependency

- Unlike an included use case, the base use case does not rely on an extended use case.
- However, an extended use case may depend on what's happening in your basic use case;
  - For example, the extended use case might assume that some exceptional circumstance in the base use case arose.

- There is no implication that an actor associated with the base use case participates in the extended use case, and the extended use case may in fact have entirely different participants.

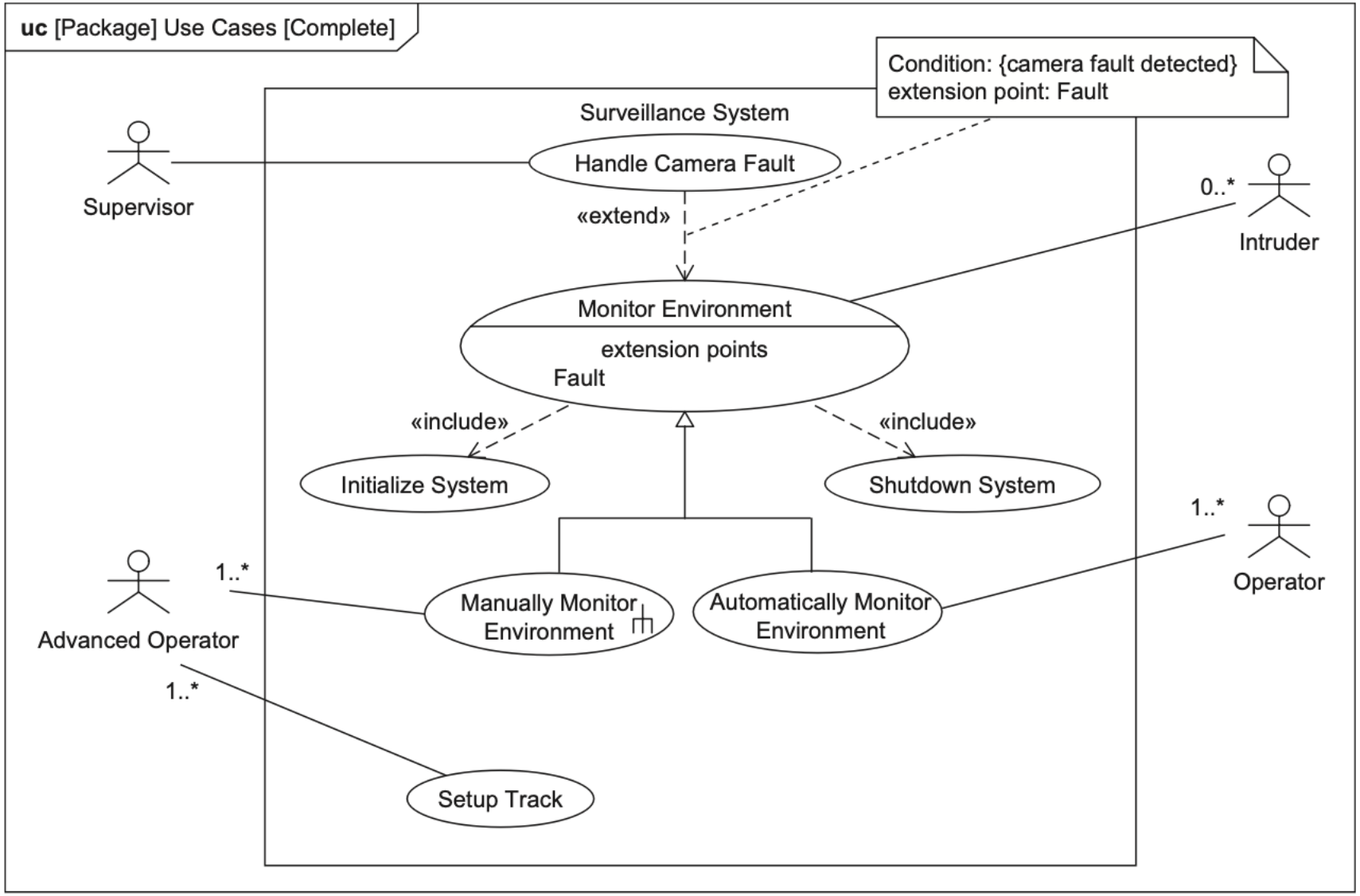


FIGURE 12.4

A set of use cases for the *Surveillance System.*

About relationship graphical representation

- Inclusion and extension are shown using **dashed lines with an open arrowhead** at the included and extended ends, respectively.
- An include line has the keyword «include» and an extension line has the keyword «extend».
- The direction of the arrows should be read as the rear end includes or extends the end of the head.

- Thus,
  - a basic use case includes an included use case, and
  - an extended use case extends a basic use case.

Classification (heritage) Relationship

- Use cases can be classified using the standard SysML generalization ratio.
- One implication, for example, is that the scenarios for the general use case are also scenarios for the specialized use case.
  - This also means that actors associated with a general use case can also participate in scenarios described by a specialized use case.
- Classification of use cases is shown using the standard SysML generalization symbol.

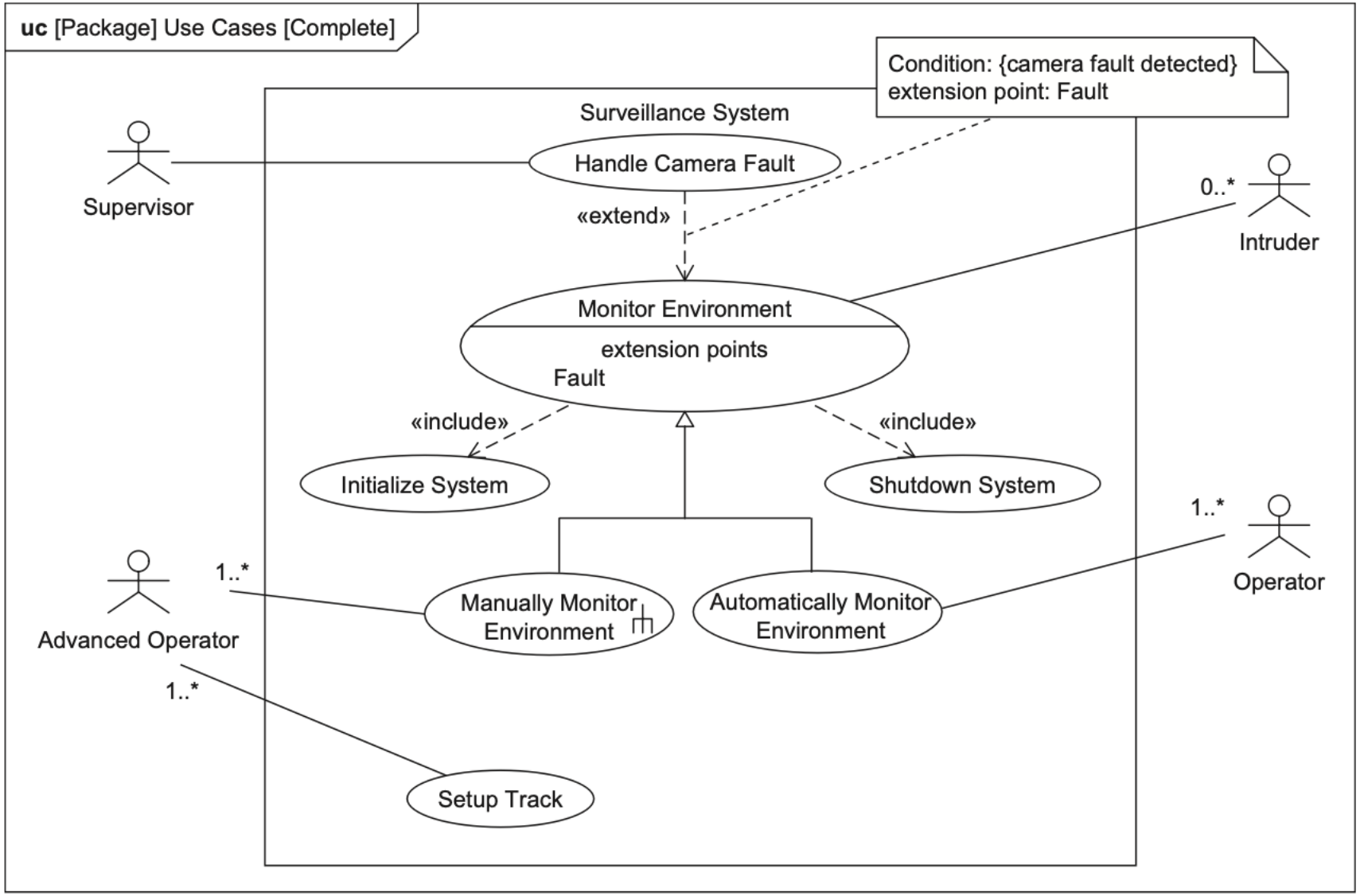


FIGURE 12.4

A set of use cases for the *Surveillance System.*

Textual description of use cases

- 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 value of the use case**.

- Description text can be captured in the template as a single or multiple comments. You can also treat each step in a use case description as a SysML requirement.

A typical use case description might include the following fields

- *Pre-conditions* - The conditions for the use case to begin.
- *Post-conditions* - the conditions after the completion of the use case.
- *Primary flow* - The most frequent scenario or scenarios of the use case.
- *Alternate and/or exception flows* - the least frequent or non-nominal scenarios. Exception flows can reference extension points and often represent flows that don't directly support the goals of the primary flow.

Here is an extract from the use case description for *Monitor Environment*:

Pre-condition

The *Surveillance System* is powered down.

Primary Flow

The *Operator* or *Operators* will use the *Surveillance System* to monitor the environment of the facility under surveillance. An *Operator* will initialize the system (see *Initialize System*) before operation and shut the system down (see *Shutdown System*). During normal operation, the system's cameras will automatically follow preset routes that have been set to optimize the likelihood of detection.

If an *Intruder* is detected, an alarm will be raised both internally and with a central monitoring station, whose responsibility it is to summon any required assistance. If available, an intelligent intruder tracking system—which will override the standard camera search paths—will be engaged at this point to track the suspected intruder. If an intelligent intruder tracking system is not available, the *Operators* are expected to maintain visual track of the suspected intruder and pass this knowledge on to the *Emergency Services* if and when they arrive.

Alternate Flow

Immediately after system initialization but before normal operation begins, it is possible that a fault will arise, in which case it can be handled (c.f. *Fault* extension point), but faults will not be handled thereafter.

Post-condition

The *Surveillance System* is powered down.