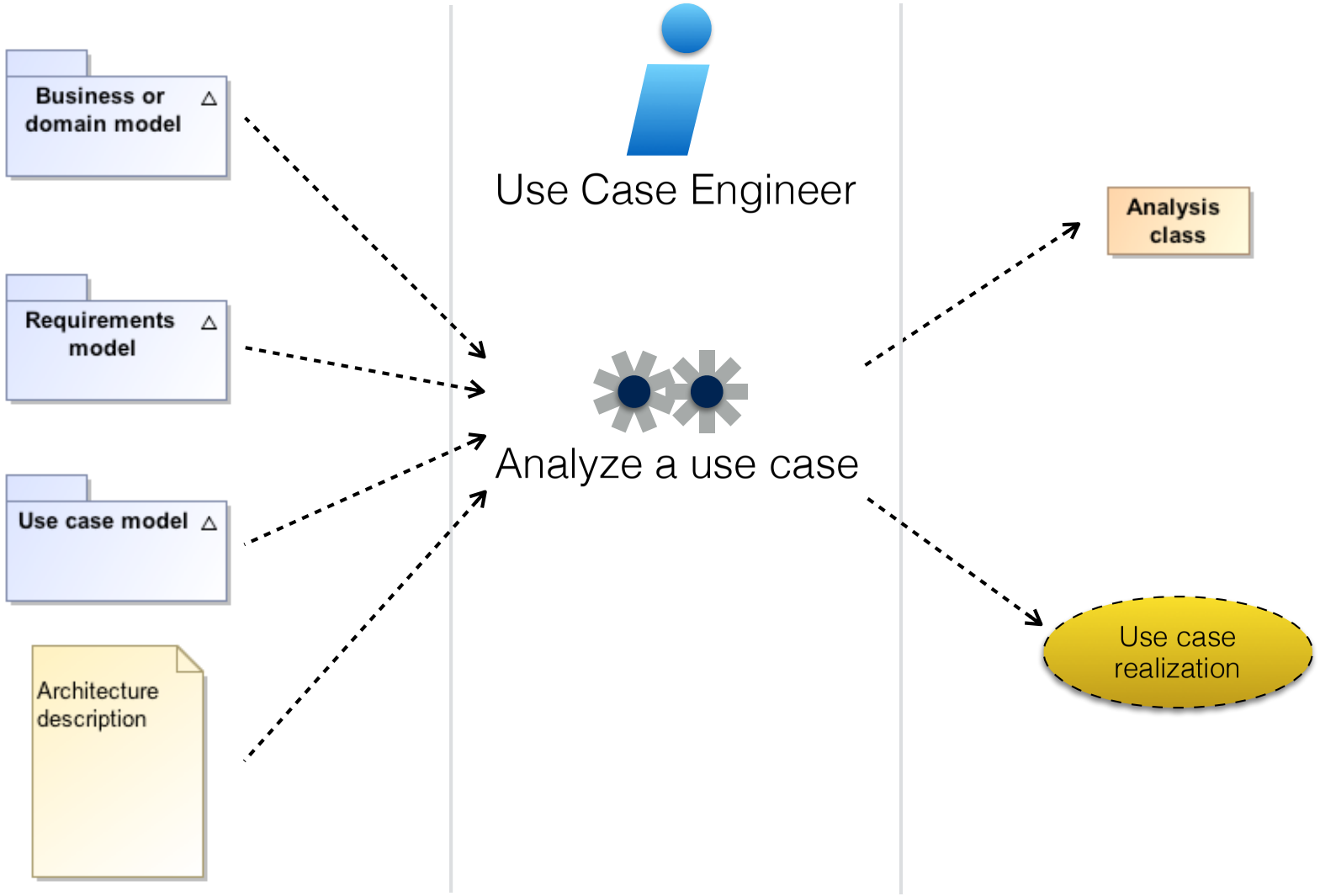
Analysis - use case realization

Analyze a use case



What are use case realizations?

- Each use case has exactly one use case realization. This consists of a subset of the model that shows how analysis classes collaborate together to realize the behavior specified by the use case
- Use case realizations model how the use case is realized by the analysis classes we have identified - hence use case *realization*
- They form an implicit part of the backplane of the model and are rarely modeled explicitly. However, it is possible to show them as a stereotyped collaboration

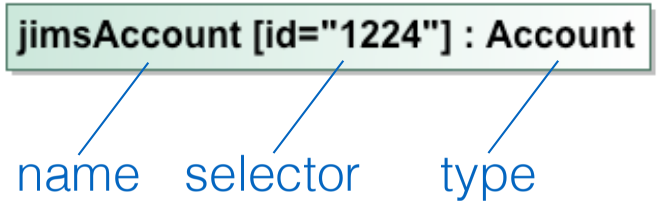
Use case realization elements

- Use case realizations comprise the following elements:
  - **Analysis class diagrams** that show relationships between the analysis classes that interact to realize the use case
  - **Interaction diagrams** that show collaborations between specific objects that realize the use case. They are “snapshots” of the running system
  - **Special requirements** - the process of use case realization may well uncover new requirements specific to the use case. These must be captured
  - **Use case refinement** - we will probably discover new information during realization that means that we have to update the original use case

Interactions

- Interactions are units of behavior of a context classifier
- In use case realization, the context classifier is a use case and the interaction shows how the behavior specified by the use case is realized by instances of classifiers
- Interaction diagrams capture an interaction as:
  - Lifelines – participants in the interaction
  - Messages – communications between lifelines

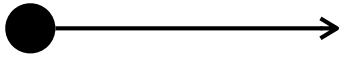
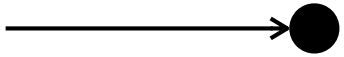
Lifelines



A lifeline has the same icon as the classifier tat it represents

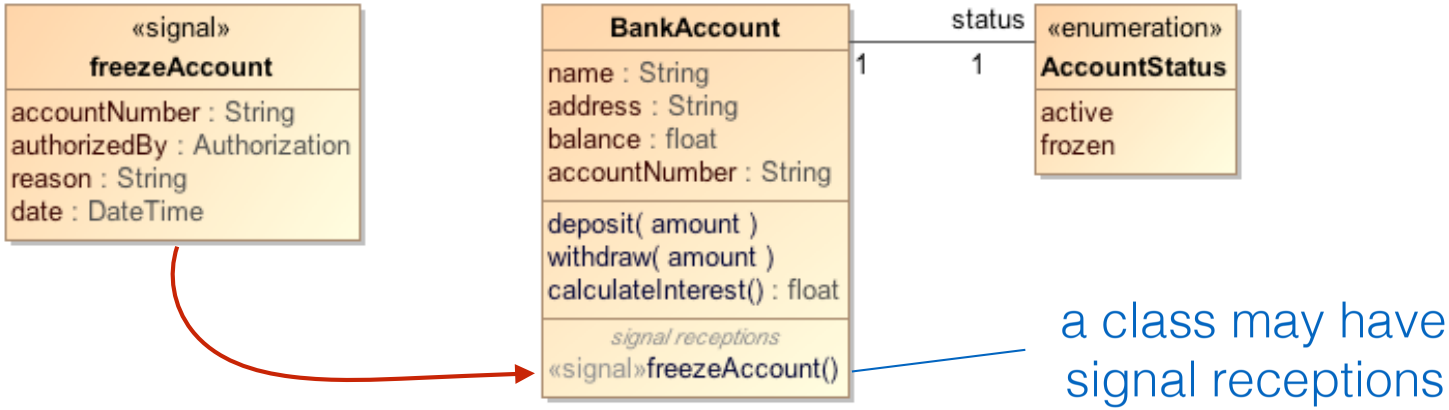
- A lifeline represents a single participant in an interaction. It shows how a classifier instance may participate in the interaction
- Lifelines may have:
  - name - the name used to refer to the lifeline in the interaction
  - selector - a boolean condition that selects a specific instance
  - type - the classifier that the lifeline represents an instance of
- They must be uniquely identifiable within an interaction by name, type or both
- The lifeline jimsAccount represents an instance of the Account class and the selector [id = "1234"] selects for an Account instance with id = "1234"

Messages

Sender → Receiver	Message type	Semantics
	synchronous	The sender sends a message to the receiver, and waits for a reply
	asynchronous	The sender sends a message to the receiver, and *does not* wait for a reply
	return	The return from a synchronous operation call. Focus of control is returned to the sender
	found	The message is received from outside the scope of the interaction
	lost	The message is sent to outside the scope of the interaction

- A message represents a communication between two lifelines
- A message can communicate a signal, an operation invocation or be unspecified
- Note: signal sends are *always* asynchronous, operation invocation may be either

Signals



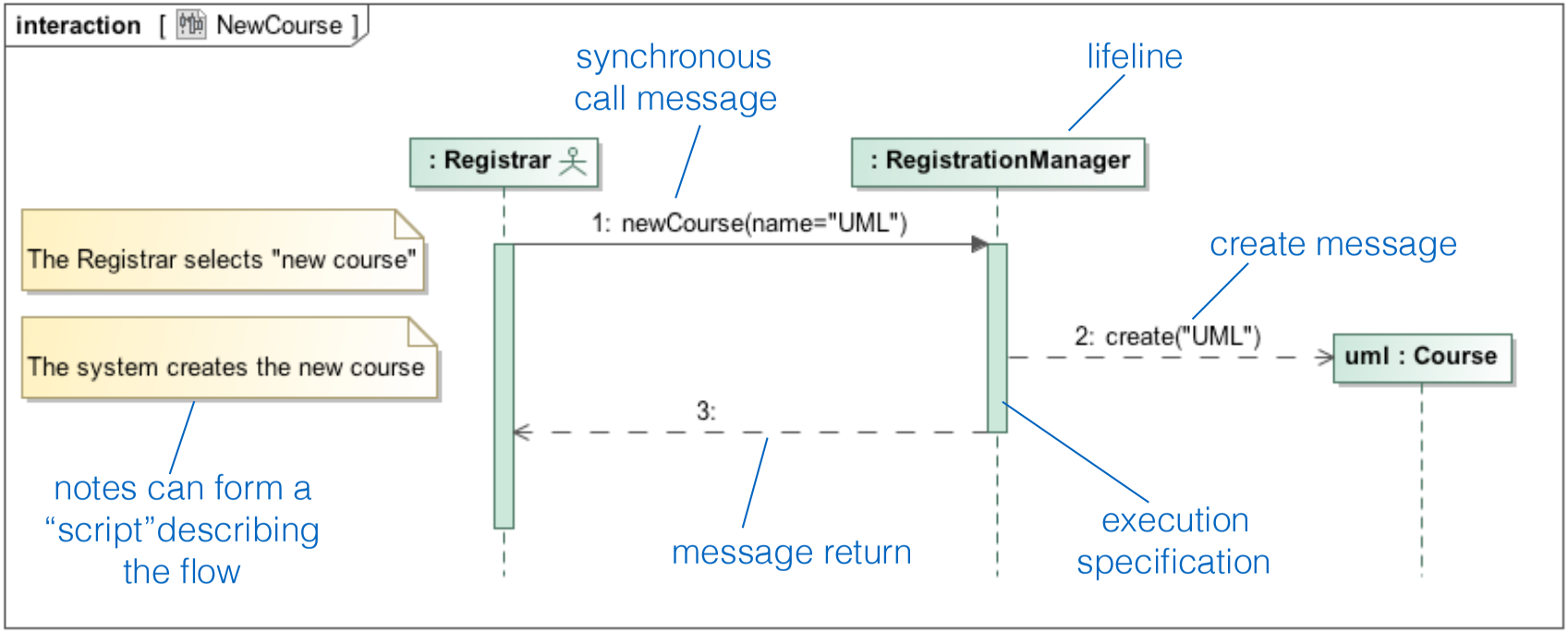
- Signals are how UML represents information communicated *asynchronously* between modeling elements
- A signal is a classifier with the stereotype «signal» that has attributes, but *no* operations - it is pure data
- Classes and other classifiers may have signal receptions that allow them to accept signals asynchronously

Interaction diagrams

Interaction diagram type	Emphasis	Capabilities
Sequence	Time-ordered *sequence* of message sends	Used to show interactions arranged in a time sequence. They do *not* show lifeline relationships explicitly - these are inferred from the message sends. They are the richest and most expressive type of interaction diagram
Communication	Structural relationships between lifelines	Useful when you need to make lifeline relationships very explicit. Quick and easy to use for simple interactions, but don't scale well
Interaction overview	How individual interactions relate to each other	Useful when you need to show how a complex interaction is generated from a set of simpler ones
Timing	Real-time aspects of the interaction	Useful whenever timing is of critical importance. *Very* useful for real time embedded systems

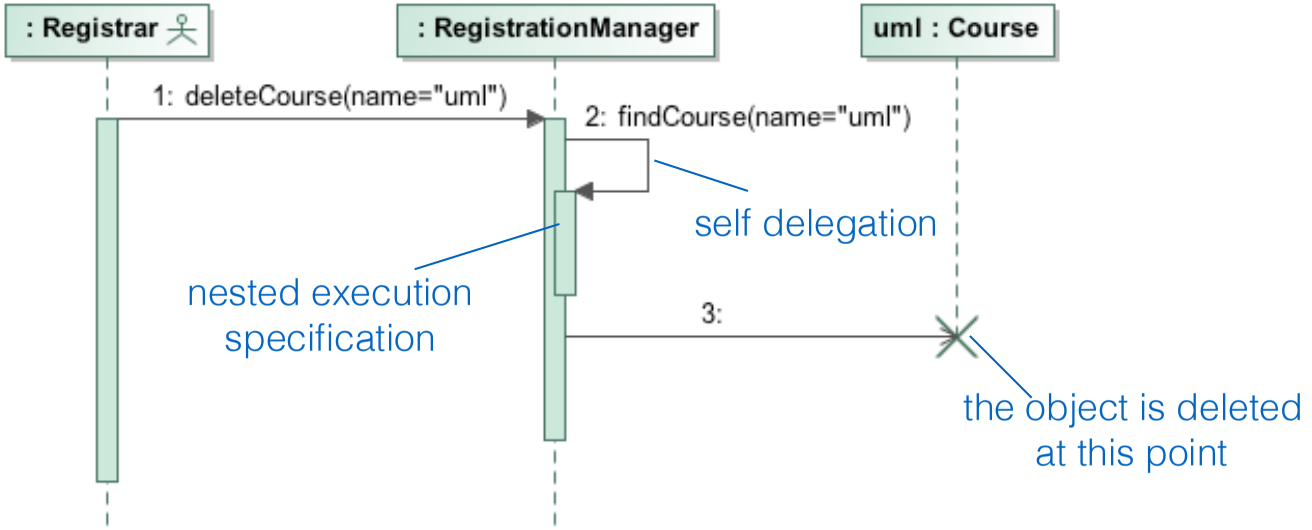
- There are four types of interaction diagram. They all show interactions, but emphasize different aspects of the interaction
- Sequence diagrams are the most widely used

Sequence diagram syntax



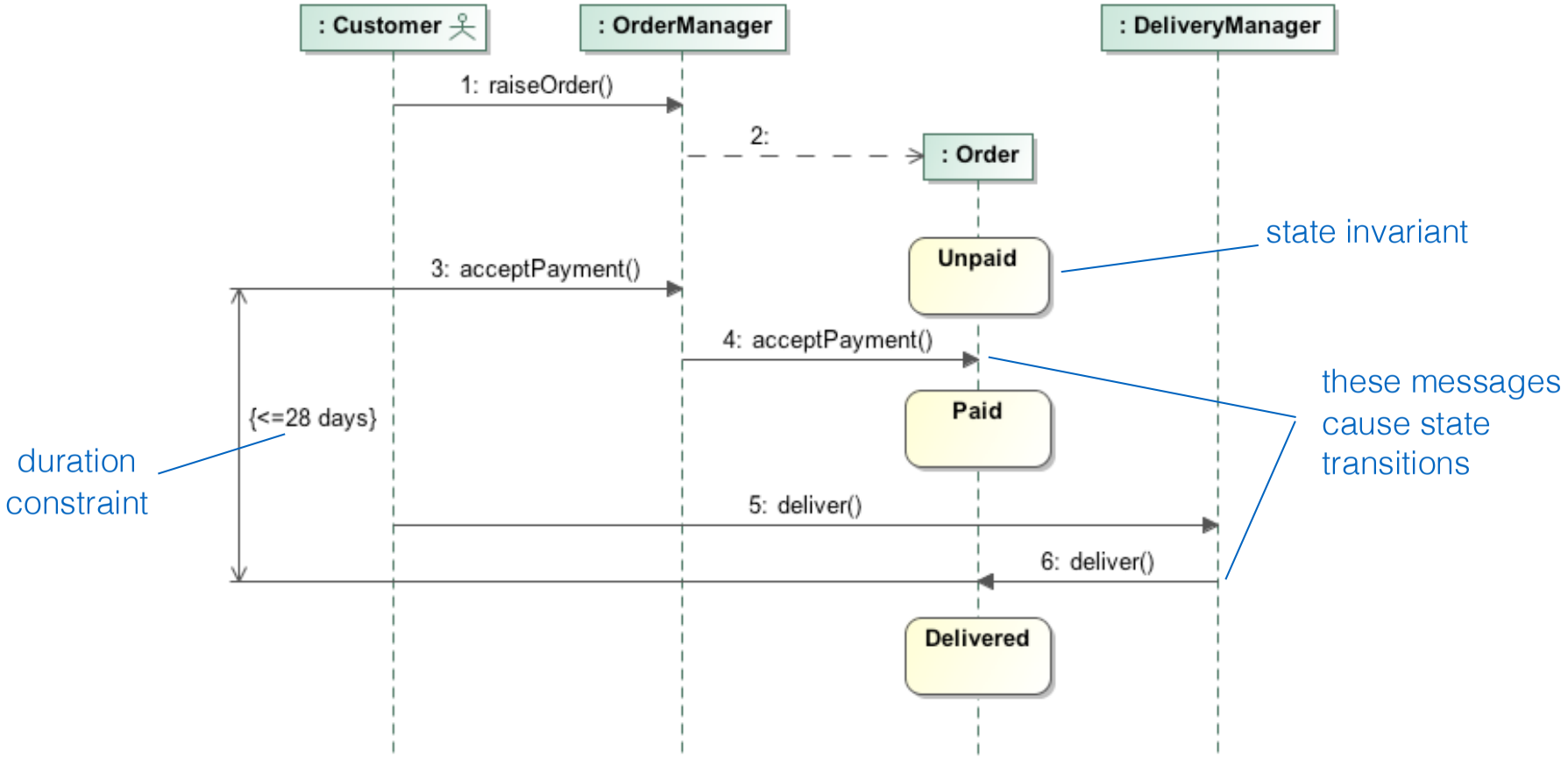
- The synchronous call message invokes an operation on the receiver
- The create message indicates that an instance of the lifeline classifier is created at this point
- Execution specifications (sometimes called activations) show how focus of control shifts between lifelines. This isn't so useful and they are often omitted

Deletion and self-delegation



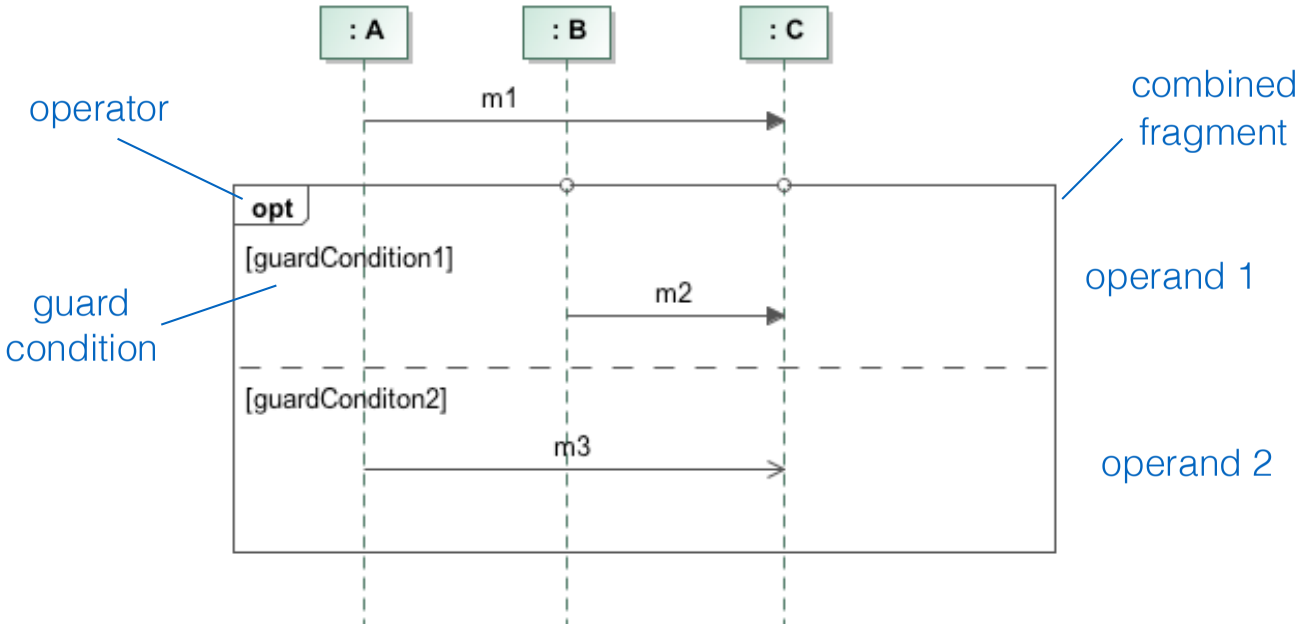
- Self delegation is when a lifeline sends a message to itself. This generates a nested execution specification
- Object deletion is shown by terminating the lifeline's tail at the point of deletion by a large X

State invariants and constraints



- The state invariant represents a state from the state machine of the lifeline classifier (see later!)

Combined fragments



- A combined fragment is an area of a sequence diagram with an *operator* and one or more *operands*
- The operator determines *how* the operands are executed
- The operand guard conditions determine *whether* an operand can execute. It will execute only if the Boolean expression in the guard condition evaluates to true

Common operators

Operator	Long name	Semantics
opt	Option	There is a single operand that executes if the condition is true (like if … then)
alt	Alternatives	The operand whose condition is true is executed. The keyword else may be used in place of a Boolean expression (like select… case)
loop	Loop	This has a special syntax: loop min, max [condition] Iterate min times then iterate while condition is true up to max times
break	Break	The combined fragment is executed rather than the rest of the enclosing interaction
ref	Reference	The combined fragment uses another interaction - an interaction use
par	Parallel	All operands execute in parallel

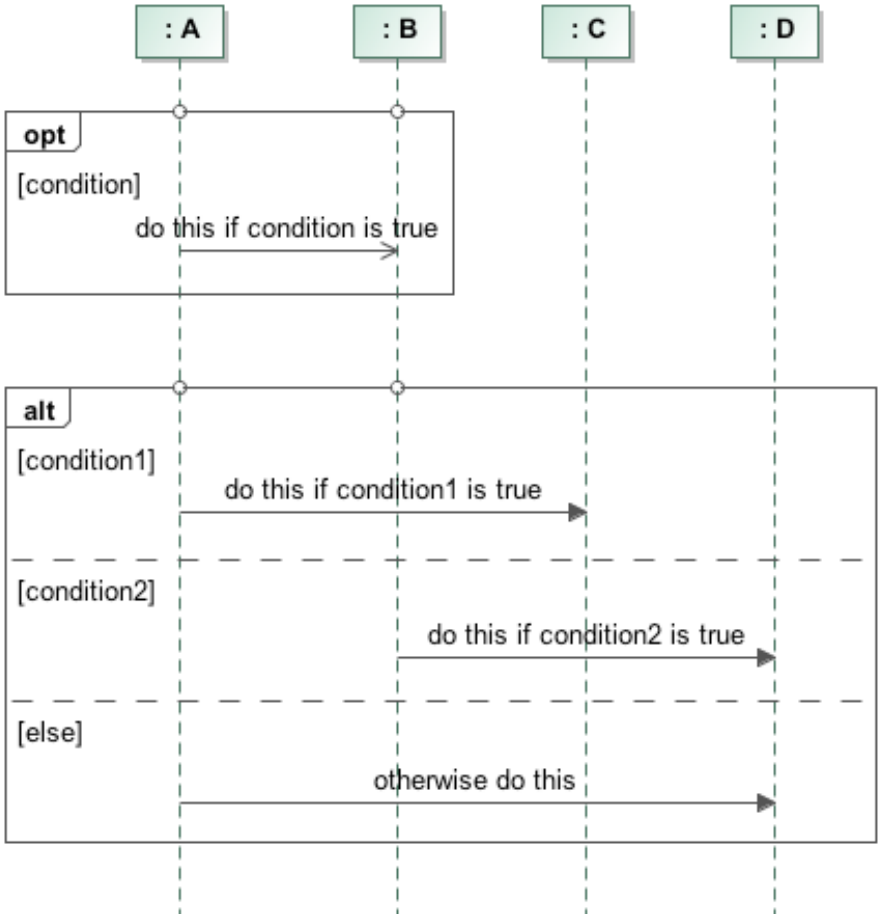
Other operators

Operator	Long name	Semantics
seq	Weak sequencing	The operands execute in parallel subject to the constraint that event occurrences on the *same* lifeline from *different* operands must happen in the same sequence as the operands
strict	Strict sequencing	The operands execute in strict sequence
neg	Negative	The combined fragment contains interactions that are invalid. Used to illustrate interactions that should not occur
critical	Critical region	The interaction must execute atomically without interruption
ignore	Ignore	Lists messages that are intentionally ignored in the interaction
consider	Consider	Lists the messages that are considered in the interaction (all others are ignored)
assert	Assertion	The operands of this combined fragment are the only valid continuations of the interaction

- These operators are not as widely used

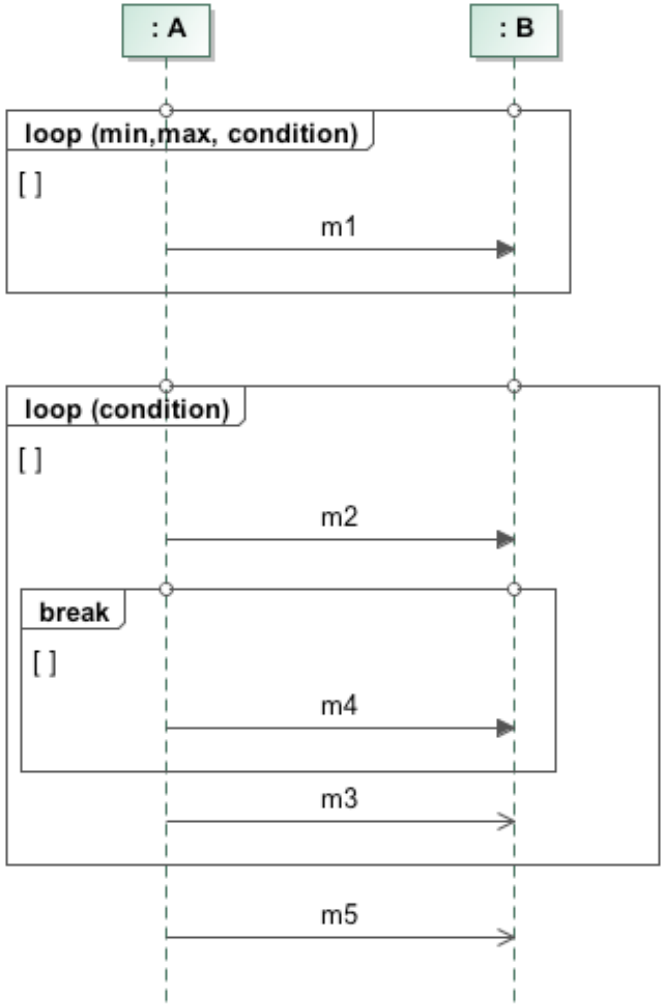
Branching with opt and alt

- opt semantics: A single operand that executes if the condition is true
- alt semantics: Two or more operands each protected by its own condition
- An operand can execute only if its condition is true
- Use else to indicate the operand that executes if *none* of the conditions are true



Iteration with loop and break

- loop semantics: Loop min times, then loop (max – min) times while condition is true
- loop syntax: A loop without min, max or condition is an infinite loop. If only min is specified then max = min. If only condition is specified then loop while it is true
- condition can be a Boolean expression or a plain text expression provided it is clear!
- When the loop is broken out of, the break fragment executes and the rest of the loop after the break does *not* execute



Normal flow: m1, (m2, m3, m2, m3…), m5
Break flow: m1, (m2, m3, m2, m4), m5

Loop idioms

Type of loop	Loop expression	Semantics
infinite loop	loop() or loop(*)	Keep looping forever (or until a break)
for i = 1 to ….	loop(n)	Loop exactly n times
while(condition) …	loop(condition)	Loop while condition is true

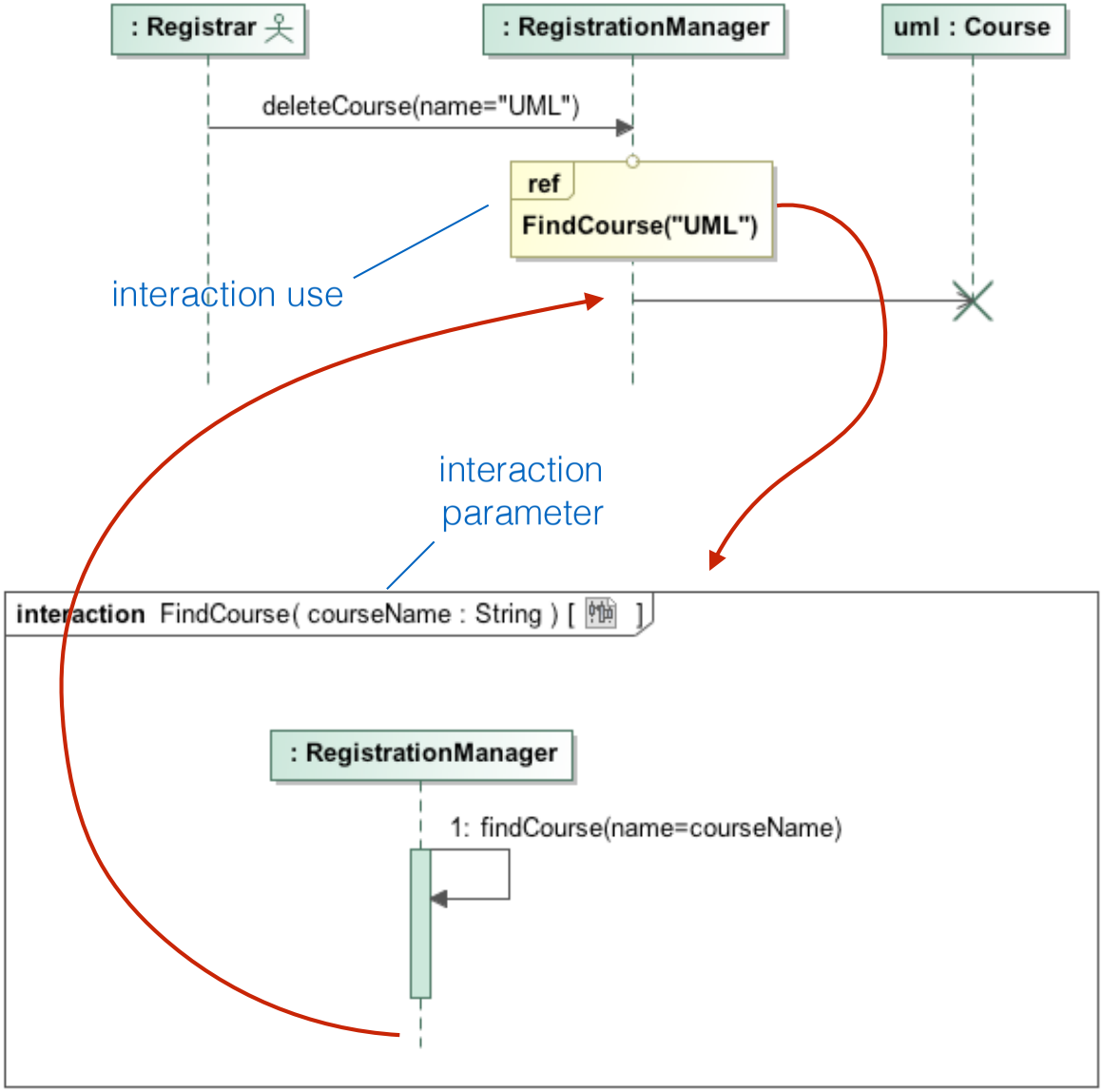
repeat
…
while(condition)
	loop(1, condition)	Loop once then loop while condition is true.

forEach object in collection	for(each object in collection)	Execute the loop once for each object in a collection
forEach object of type T	for(each object in in type :T)	Execute the loop once for each object of a particular type

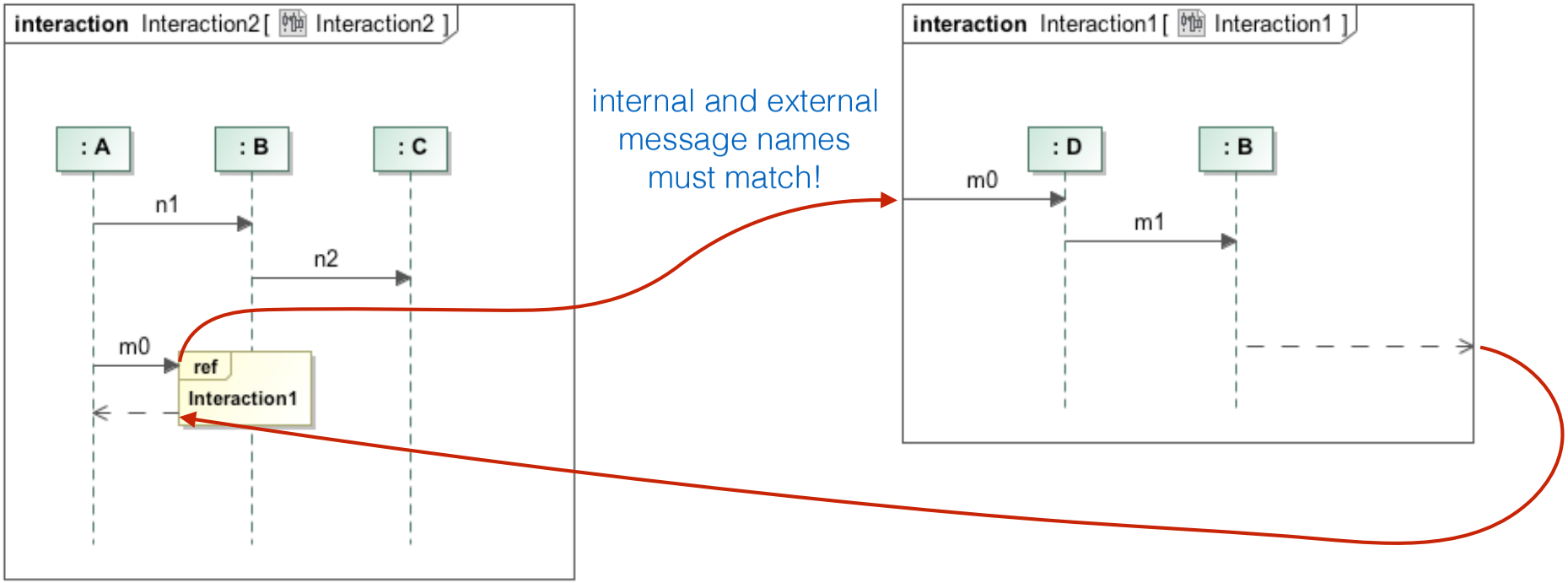
- We prefer loop(*) to loop(), because loop() is ambiguous. It may mean an infinite loop is intended, or just that the loop condition has not yet been specified

Reuse with the ref operator

- We have refactored FindCourse into a reusable interaction that takes a parameter called courseName
- Interactions with parameters use operation syntax, so they may have return types, etc.
- In the example, we call the interaction FindCourse(…) with the argument "UML"



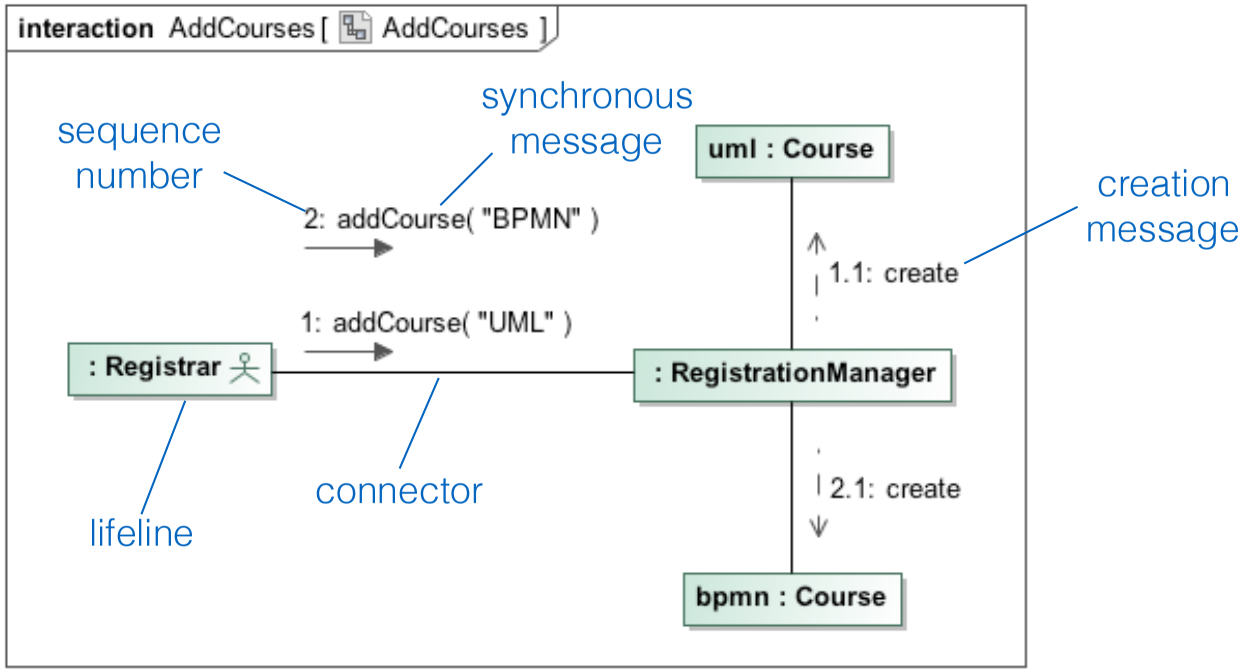
Gates



- Gates are inputs and outputs of interactions and combined fragments. They provide connection points that relate messages inside an interaction use or combined fragment to messages outside it

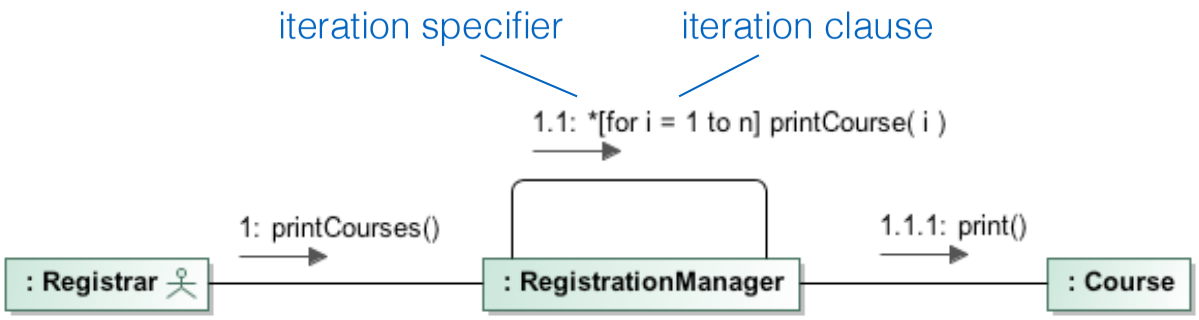
- Syntactically, they are just points on the diagram frame! They may be given an explicit name, but this not very useful

Communication diagram syntax



- Communication diagrams emphasize the structural aspects of an interaction. Compared to sequence diagrams they are semantically weak
- The connector relationship specifies that there must be a link between objects when they play the roles in the interaction specified by the lifelines on each end of it

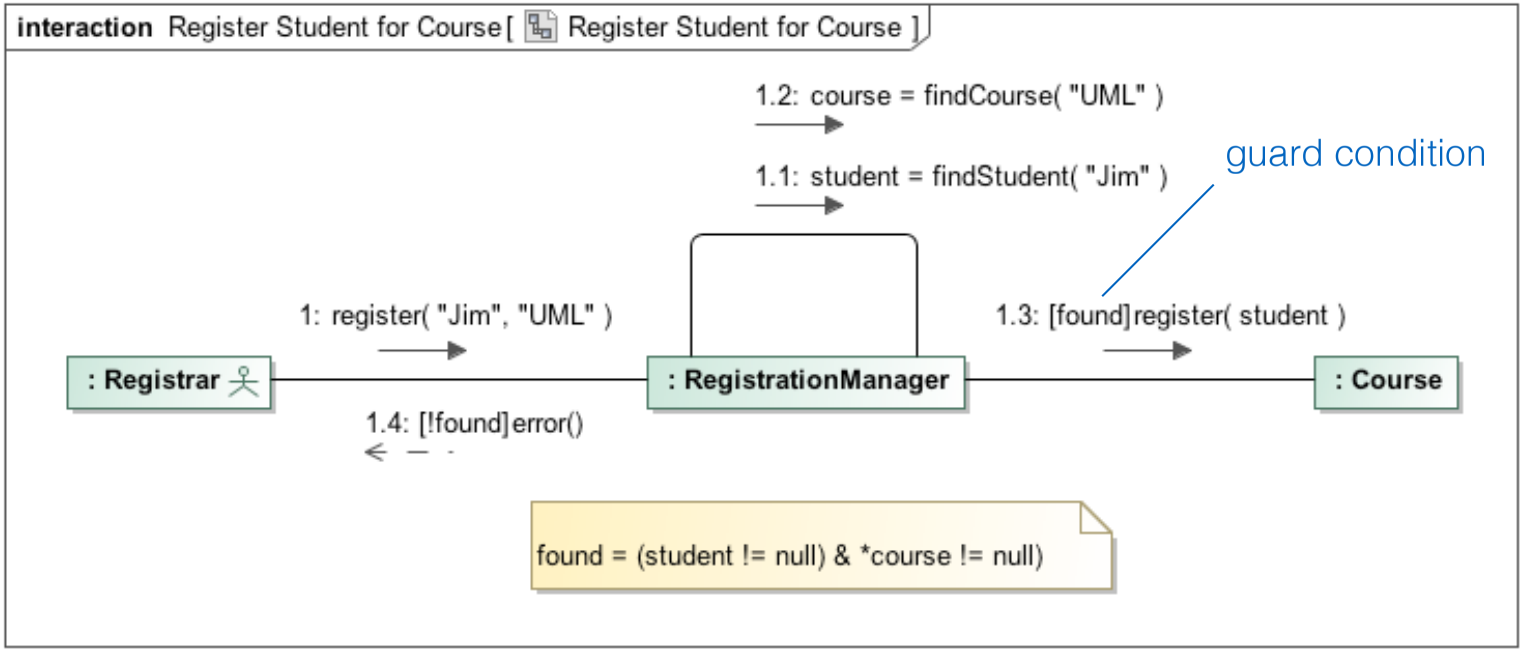
Iteration



- Iteration is shown by using the iteration specifier *, and an optional iteration clause e.g. [for i = 1 to n]. There is no prescribed UML syntax for iteration clauses, so use code or pseudo code

- To show that messages are sent in parallel use the parallel iteration specifier, *//

Branching



- Use guard conditions to create branches
- It is quite hard to show branching clearly on communication diagrams!

Summary

- In this section we have looked at use case realization using interaction diagrams. There are four types of interaction diagram:
  - Sequence diagrams – emphasize time-ordered sequence of message sends
  - Communication diagrams – emphasize the structural relationships between lifelines
  - Interaction overview diagrams – show how complex behavior is realized by a set of simpler interactions
  - Timing diagrams – emphasize the real-time aspects of an interaction
- We have looked at sequence diagrams and communication diagrams in this section - we will look at the other types of diagram later