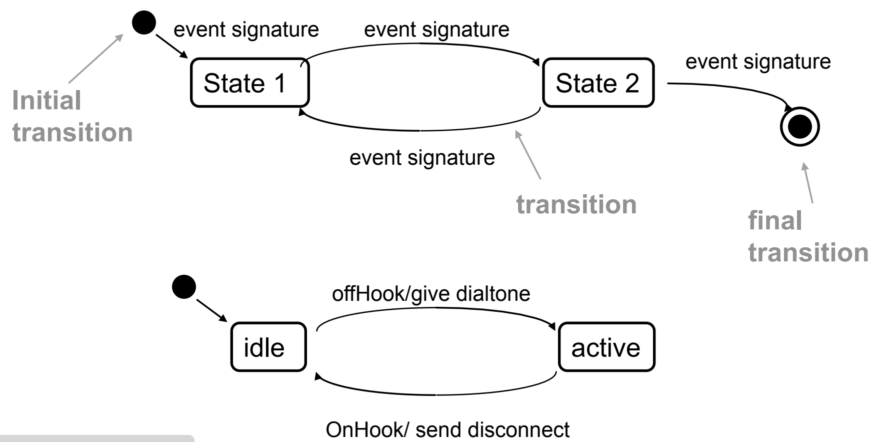
# About UML's Statecharts




## Finite State Machines

- Finite State machines (FSMs) describe behavior in terms of <u>states</u>, <u>events</u>, and <u>transitions</u>:
  - They have their transitions triggered by events.
  - They are equivalently called state (transition) diagrams.
  - They are useful in automating the generation code and tests.
  - Extended FSMs (eFSMs) allow the use of state variables.
  - Douglass has suggested patterns for FSMs of real-time systems
- For OO Development, an FSM may be developed for each class:
  - Each object is in exactly one state at any point in time.
  - Events correspond to the messages sent by other objects.
  - Many classes do not have sophisticated state behavior - often just one state.
  - Contrary to Structural Programming, we do not develop an FSM for the system: behavior is distributed across objects:
    - » semantically we can think of communicating eFSMs

## Basic UML Notation for State Machine




Event signature:

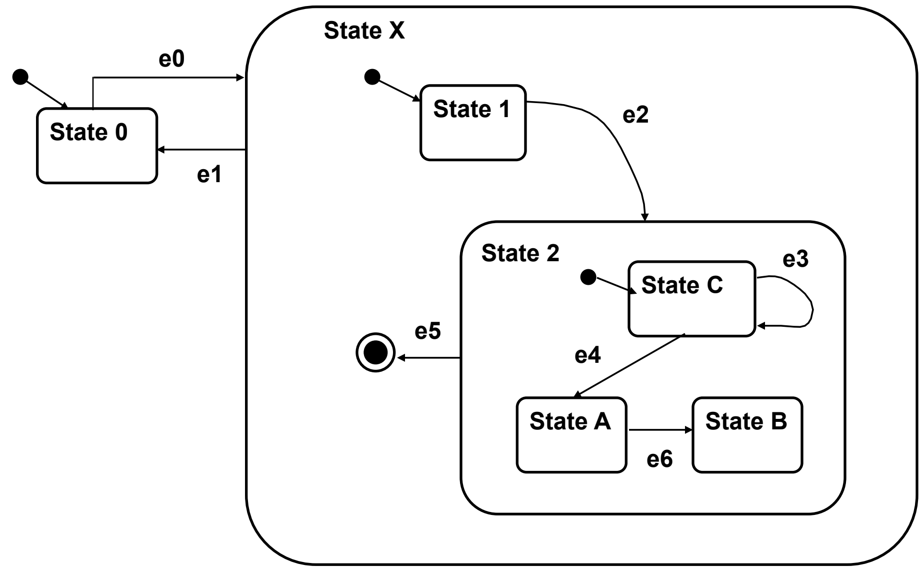
**event(parameters: type) [guard condition] / action ^sends**




## Statecharts = Hierarchical eFSMs

- **The notation used in UML is taken almost directly from Harel Statecharts:**
  - **Statecharts embellish traditional state machines by providing notation for nesting and concurrency.**
  - **The embellishments help simplify visually state machines, which can, otherwise, become quite complex.**
    - » **But the embellishments introduce semantic difficulties…**
- **Hierarchical eFSMs lend themselves to iterative development: (the usual stub idea…)**
  - **but remember that a single transition may make an FSM non-deterministic!**
  - **and non-determinism is not the only problem of communicating state machines: deadlocks and livelocks must be detected…**
  - **and Binder insists on statecharts being flattened if tests are to be extracted from them…**

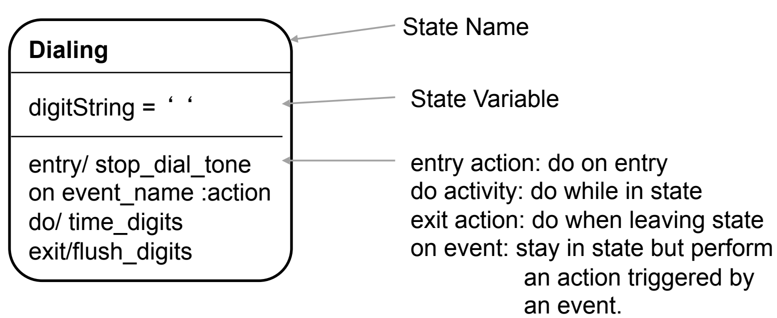
# Nested States





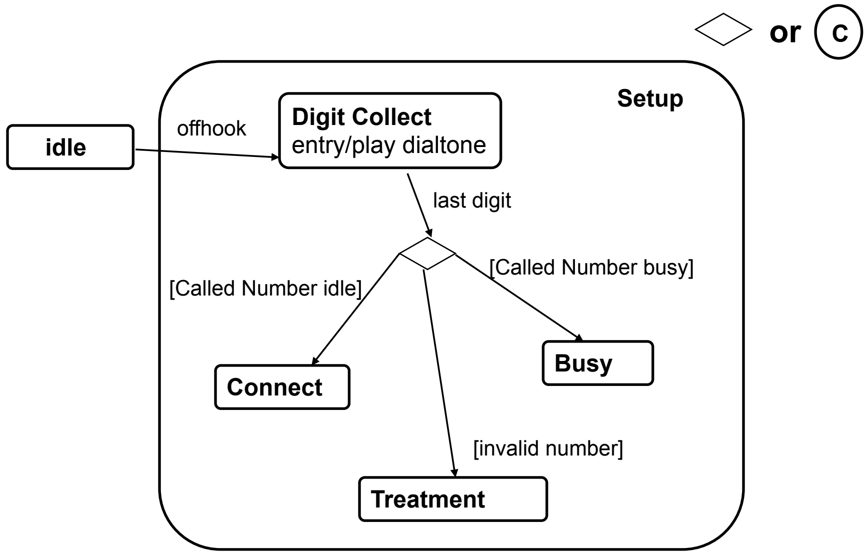


# UML State Structure




**Such rich semantics give modeling flexibility, but also create headaches for testers…**
**And UML-RT does NOT have the exact same semantics…**

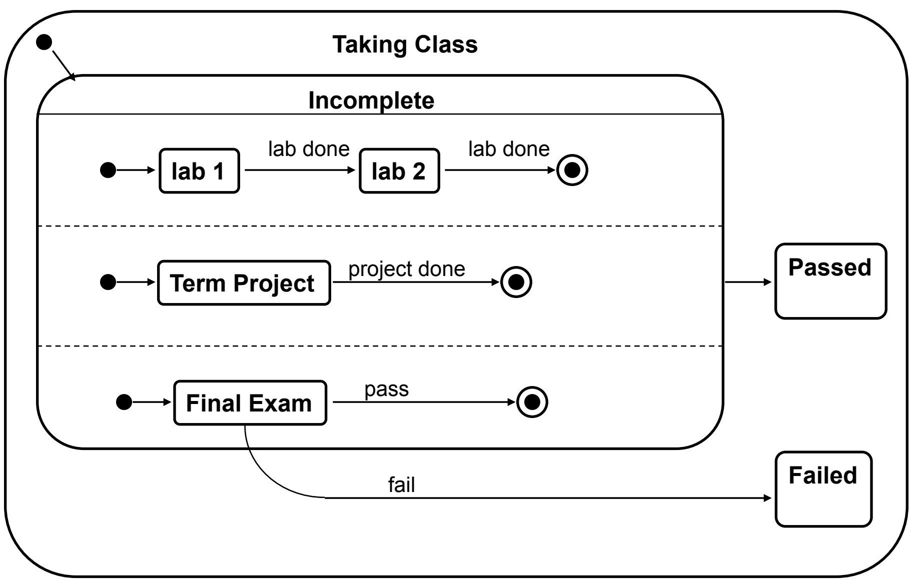
## Conditional Connector



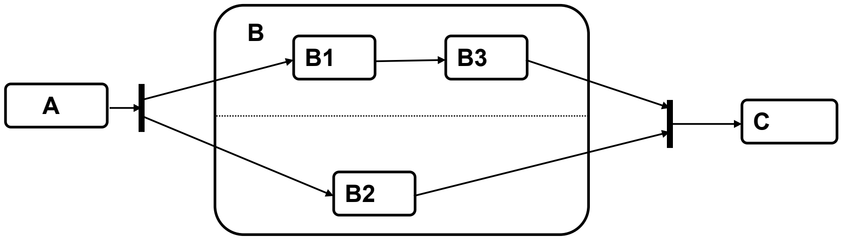




## And Concurrency



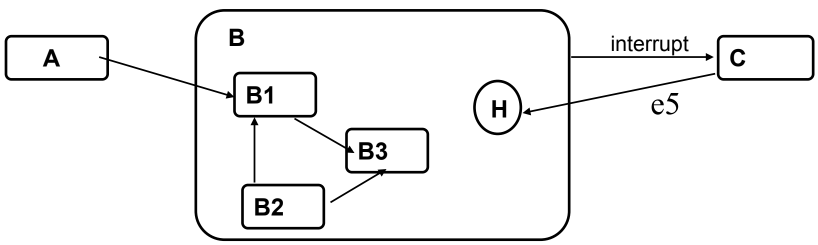
## Explicit Fork and Join



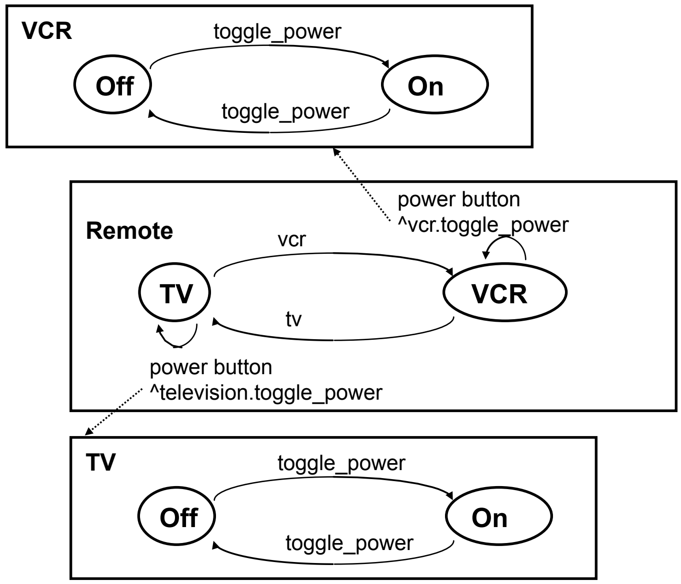




## History Indicator



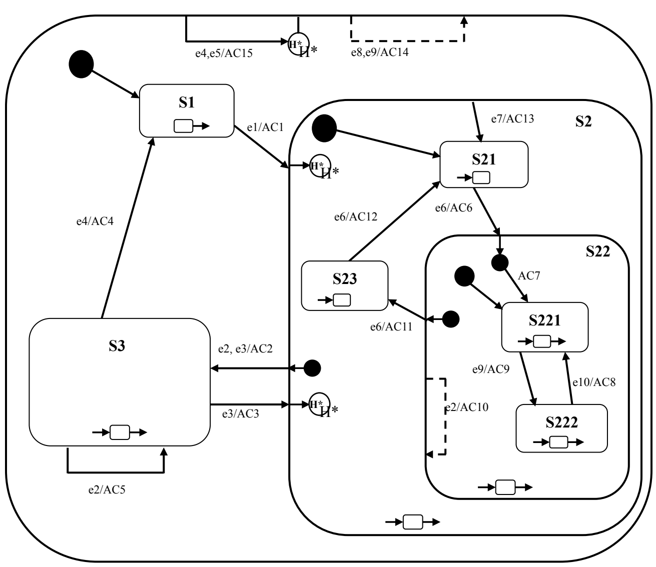
## Sending Events to Other Objects







## Understanding Hierarchical State Machines in ROSE-RT



# From Problem Statement To Statecharts





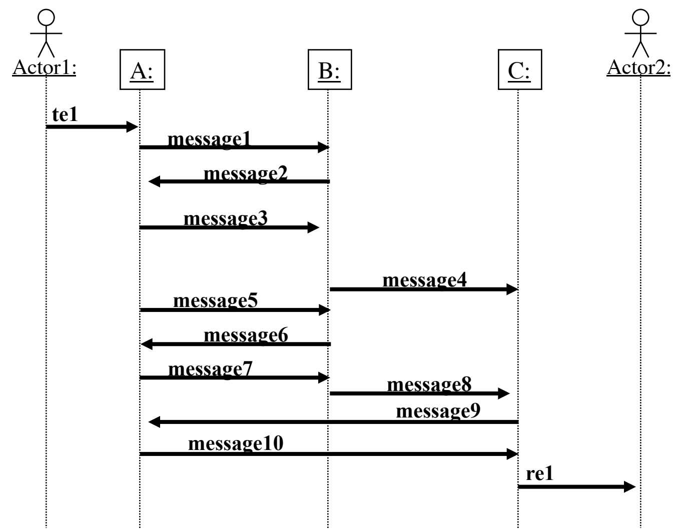
## The Scenario-Driven Recipe

**In UML the transition from a set of UCs to a set of a sequences of messages and to the relevant statecharts can be conducted in 4 steps:**

1. **Definition of pre and post conditions associated with a use case**
2. **In each instance of the *corresponding* interaction diagram, introduction of states before every incoming message**
3. **Naming of the states**
4. **Generation of statecharts**

**A glitch: this recipe downplays completely inter-UC relationships…**

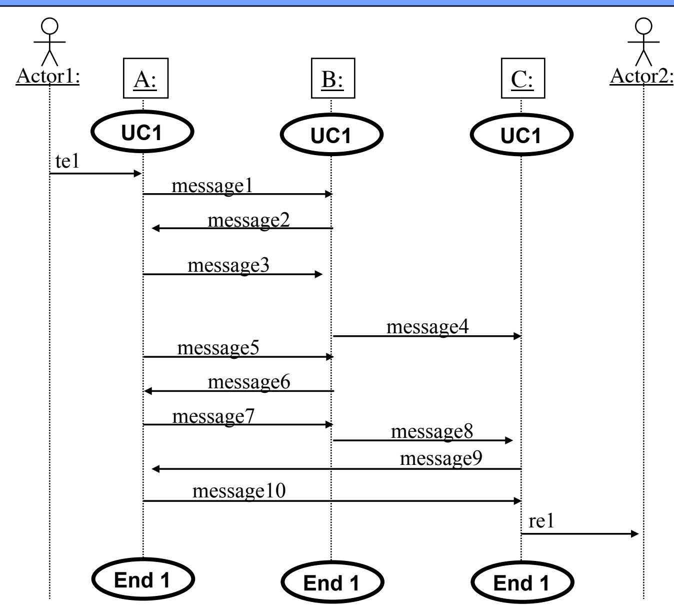
## A Simple Example




© J.-Pierre Corriveau, 1997- present

3004 T4e - 15

## Step1: Definition of Pre and Post Conditions



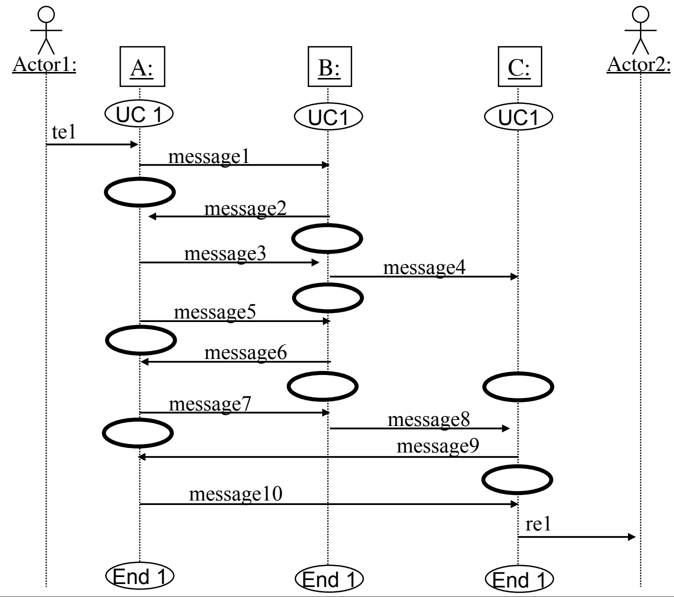

**Warning:**
**Though the names may be the same, each instance has its own states!**

**These states are used to enforce the necessary pre- and post-conditions…**

© J.-Pierre Corriveau, 1997- present

3004 T4e - 16

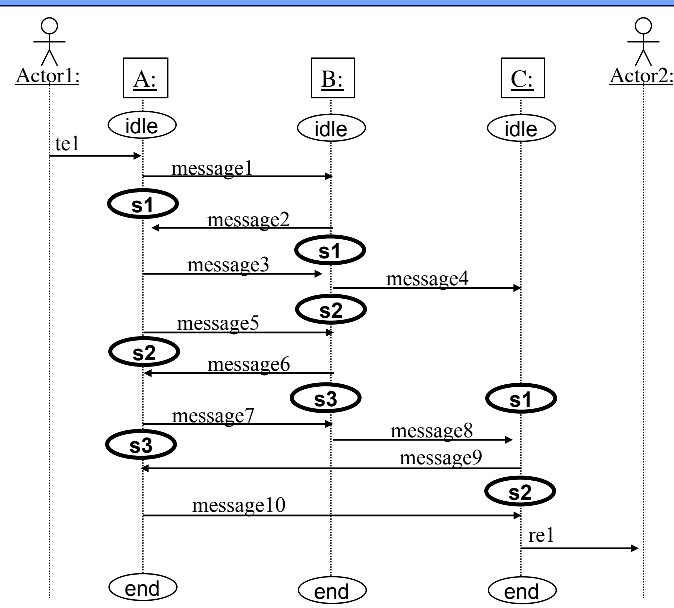
## Step2: Introduction of States




**Step 2:** Introduce a nameless state before every incoming asynchronous message.




## Step3: Naming the States




**Again!** Though the names may be the same, each instance has its own states!

**challenge:** integration of states across several such diagrams

## Step4: Generation of Statecharts

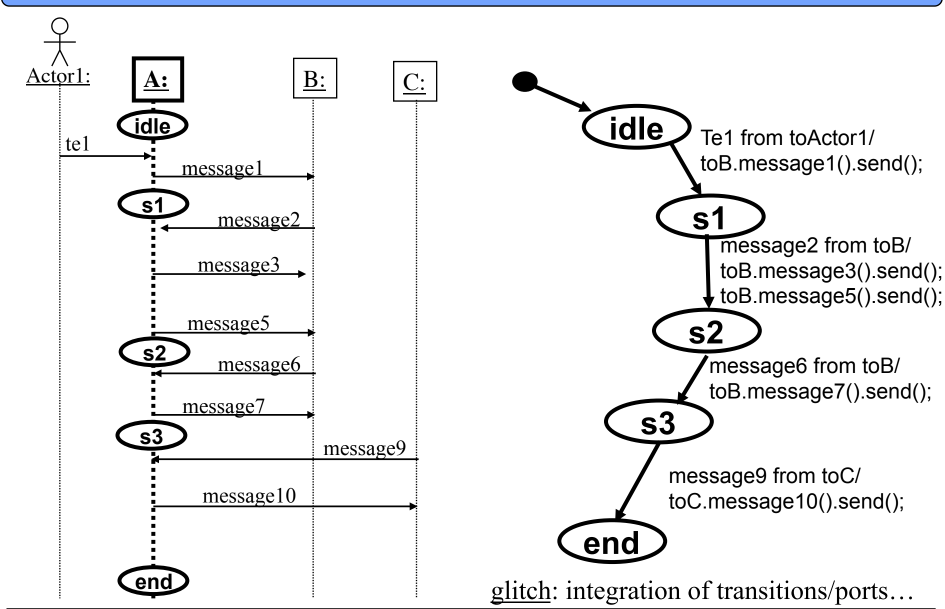
**Task:**

**For each instance of the sequence define a statechart as follows:**

- **Define a state for each state defined in the sequence diagram**
- **Define transitions between the states**
- **Define each transition in terms of a triggering event (i.e., an incoming message) and transition actions (i.e., sending outgoing messages)**

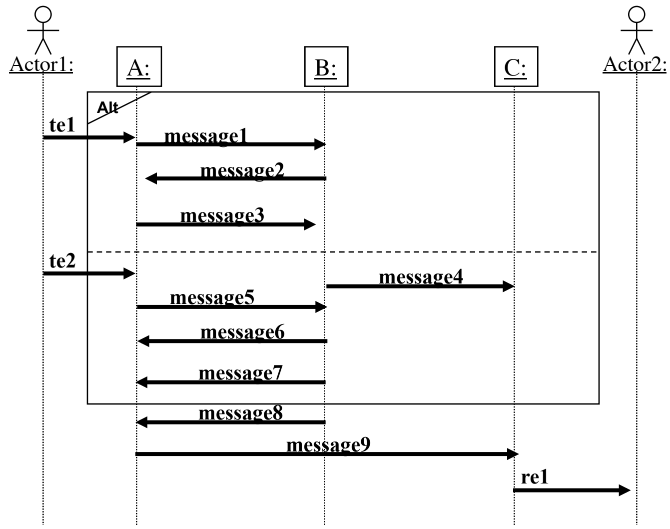



## Step4: Generation of Statecharts



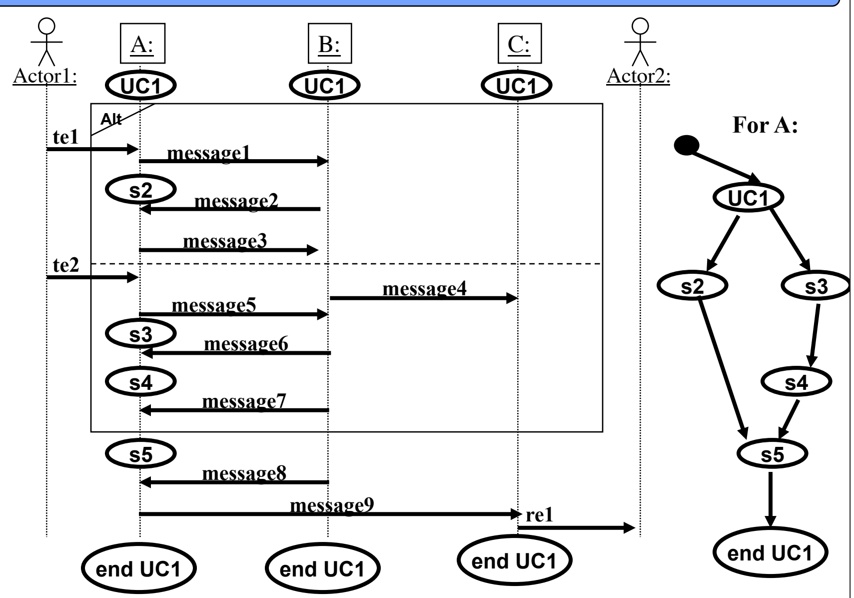

glitch: integration of transitions/ports…

# A Second Example (1)

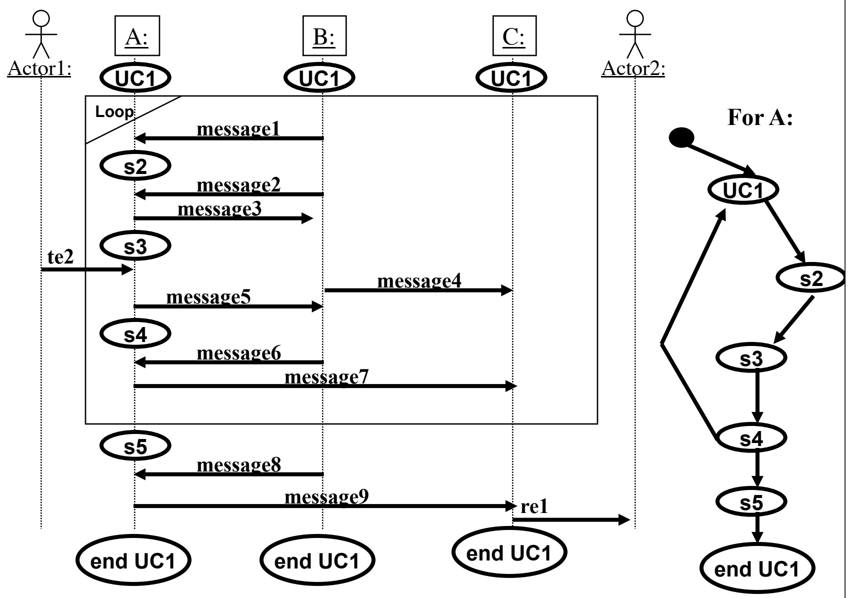





# A Second Example (2)



## A Third Example








## Looks Simple?

- The ultimate success of the extraction of a role state machine depends:
  - on the exact semantics of the notation you use.
    - » UML's statecharts are one of several possible semantics.
    - » Other models exist: eg., Douglass
  - on the complexity of the interaction diagram to start with:
    - » UML 2.0 sequence diagrams have much more complicated syntax and semantics than the interaction diagrams currently in ROSE-RT. This does complicate role state machine extraction.
- Role state machines??
  - We obtain a state machine for each instance participating in a single use case.
  - Other instances of the same class may participate in other use cases!
    - » We will say that instances of a class may play different *roles* in different use cases.
    - » Once we have role state machines, we will need to *consolidate* them (to use Gomaa's terminology).