Hugbúnaðarverkefni 1 / Software Project 1

11. Behavior Models

HBV501G – Fall 2018

Matthias Book

HÁSKÓLI ÍSLANDS

VERKFRÆÐI- OG NÁTTÚRUVÍSINDASVIÐ

IÐNAÐARVERKFRÆÐI-, VÉLAVERKFRÆÐI-
OG TÖLVUNARFRÆÐIDEILD

In-Class Quiz Prep

- Please prepare a scrap of paper with the following information:

- ID: _____@hi.is   Date: ________
- a) ______________ e) ______________
- b) ______________ f) ______________
- c) ______________ g) ______________
- d) ______________ h) ______________

- During class, I’ll show you questions that you can answer very briefly
  - No elaboration necessary
- Hand in your scrap at the end of class

- All questions in a quiz weigh same
- All quizzes (ca. 10 throughout semester) have the same weight
  - Your worst 2 quizzes will be disregarded
- Overall quiz grade counts as optional question worth 7.5% on final exam

Recap: Assignment 4: Code Review – Schedule

- By **Sun 11 Nov,** make your **project artifacts** available to your partner team:
  - Your project vision and design model from Assignments 1 & 3 (incl. fixes of severe issues)
  - A current snapshot of your source code (does not need to be the finished product)
- Take **one week** to **review** the other team's code and **document** your findings:
  - Comment on clarity of design, quality of implementation, readability of code, tech choices
  - State what you like and make suggestions for improvements
- By **Sun 18 Nov,** submit your **review report** to Ugla
  - 1-2 pages in PDF
- On **Thu 22 Nov, discuss** your findings with the other team and your tutor.

- **Grading criteria:**
  - Quality of constructive feedback on other team's design and code (80%)
  - Design and technology issues identified in your own system (10%)
  - Coding style / clarity issues identified in your own system (10%)

Recap: Assignment 4: Code Review – Considerations

- **When reviewing your partner team’s code, consider the following things:**
    - Clarity of the design
        - Is the code consistent with the design model?
        - Is it easy to trace the control flow through the code as a request is processed?
    - Quality of the implementation
        - [Is the system readily executable?]
        - Are there obvious bugs in areas that don’t seem to be under construction anymore?
    - Readability of the code
        - Is the naming of classes, methods, attributes etc. descriptive and helpful?
        - Do the comments help you understand the code?
        - Can you tell which parts are done and which are still under construction?
    - Technology choices
        - Are the features of the programming language and application framework used appropriately?
- **What do you like? What would you improve?**

Recap: Assignment 4: Code Review – Partner Teams

(‘→’ = ‘**receives** and **reviews** code of’)

- **08:30-09:30 Timeslot**
  - **Andri’s teams**
    - 1 → 4
    - 4 → 1
    - 7 → 10
    - 10 → 7
  - **Daníel’s teams**
    - 2 → 5
    - 5 → 8
    - 8 → 2
  - **Matthias’ teams**
    - 3 → 6
    - 6 → 9
    - 9 → 3
- **09:30-10:30 Timeslot**
  - **Andri’s teams**
    - 13 → 20
    - 20 → 13
  - **Daníel’s teams**
    - 11 → 16
    - 16 → 11
  - **Matthias’ teams**
    - 12 → 15
    - 15 → 19
    - 19 → 12
- **10:30-11:30 Timeslot**
  - **Andri’s teams**
    - 23 → 26
    - 26 → 23
  - **Daníel’s teams**
    - 24 → 27
    - 27 → 29
    - 29 → 24
  - **Matthias’ teams**
    - 25 → 28
    - 28 → 30
    - 30 → 25

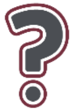
Quiz #8 Solution: Interaction Room Concepts

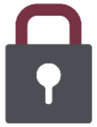
- **What is the purpose (1-4) of the following canvases (a-d)?**

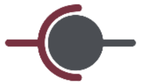
a) Integration canvas **(2)**
b) Interaction canvas **(3)**
c) Object canvas **(1)**
d) Process canvas **(4)**

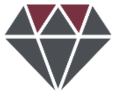
1. Describes relationships between business and technical data structures
2. Illustrates interfaces and dependencies with external systems
3. Sketches dialog flow and look & feel of key dialogs
4. Visualizes business processes that the system needs to support

- **What is the meaning (5-8) of the following annotation symbols (e-h)?**

e)  **(8)**

f)  **(7)**

g)  **(6)**

h)  **(5)**

5. Business value
6. External resource
7. Security
8. Uncertainty

UML Behavior Diagrams

see also:

- Larman: Applying UML and Patterns, Ch. 28 & 29
- Miles, Hamilton: Learning UML 2.0, Ch. 3 & 14

Modeling Static and Dynamic Aspects of a System

- Class diagrams allow us to describe only the **static, structural aspects** of a system (in a design model) or an application domain (in a domain model)

- Equally important is to understand a system’s behavior or a domain’s processes, i.e. its **dynamic, behavioral aspects**.

- The Unified Modeling Language (UML) provides several diagram types to model both static and dynamic aspects of a system or domain.

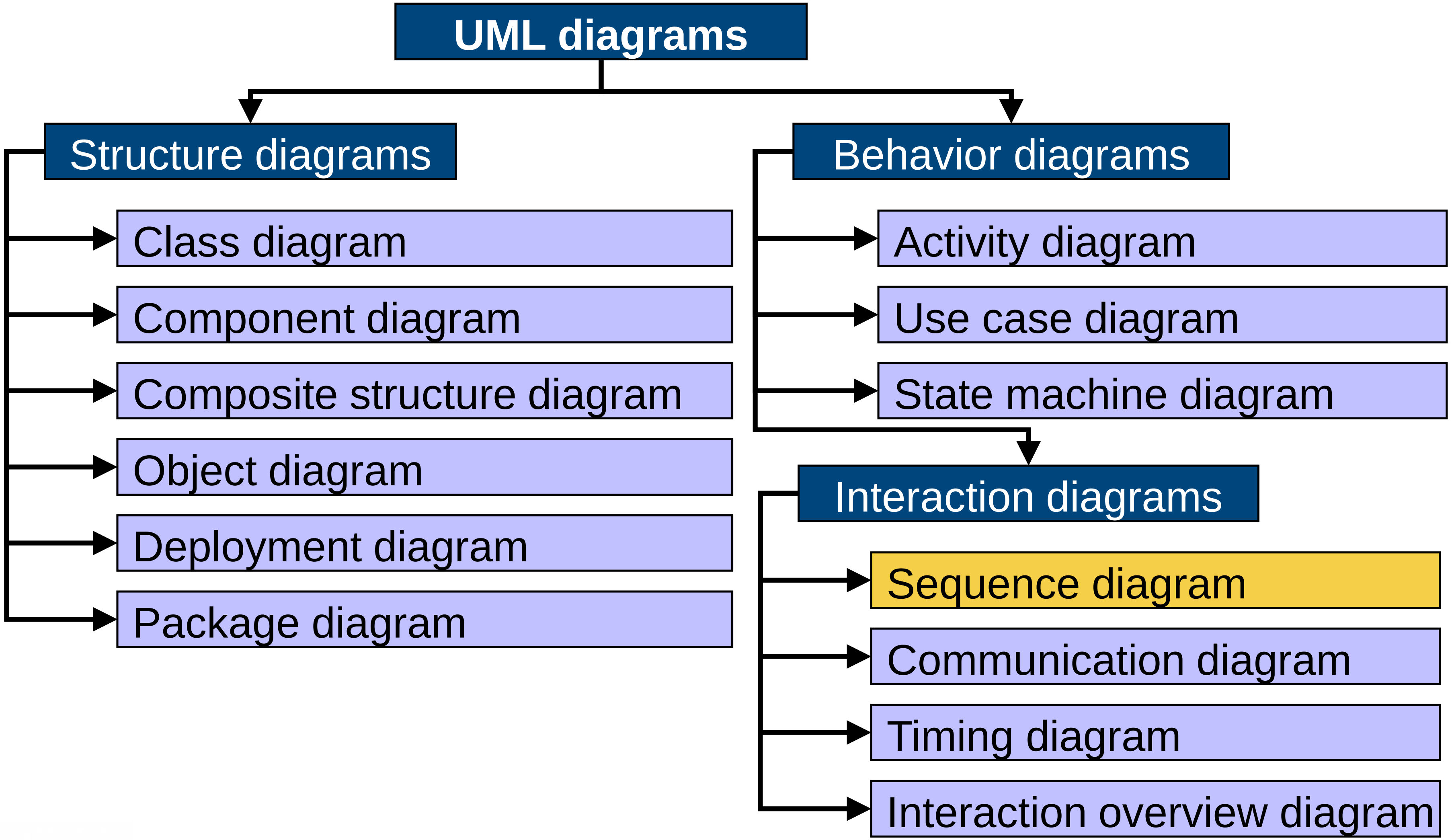
UML Diagram Types

- **Logical view**
  - What is the system made up of?
  - How do its parts interact with each other?
    - ➢ UML Class diagram
    - ➢ UML Object diagram
    - ➢ UML State machine diagram
    - ➢ UML Interaction diagram
- **Process view**
  - What processes exist in the domain?
  - What must happen within the system?
    - ➢ UML Activity diagram

(Structure following Philippe Kruchten's
4+1 View Model of Software Architecture)

- **Development view**
  - How are the system's parts and layers organized?
    - ➢ UML Package diagram
    - ➢ UML Component diagram
- **Physical view**
  - How do the abstract parts map to the deployed system?
    - ➢ UML Deployment diagram
- **Use case view**
  - What is the system supposed to do?
    - ➢ UML Use case diagram
    - ➢ UML Interaction overview diagram

UML Diagram Types



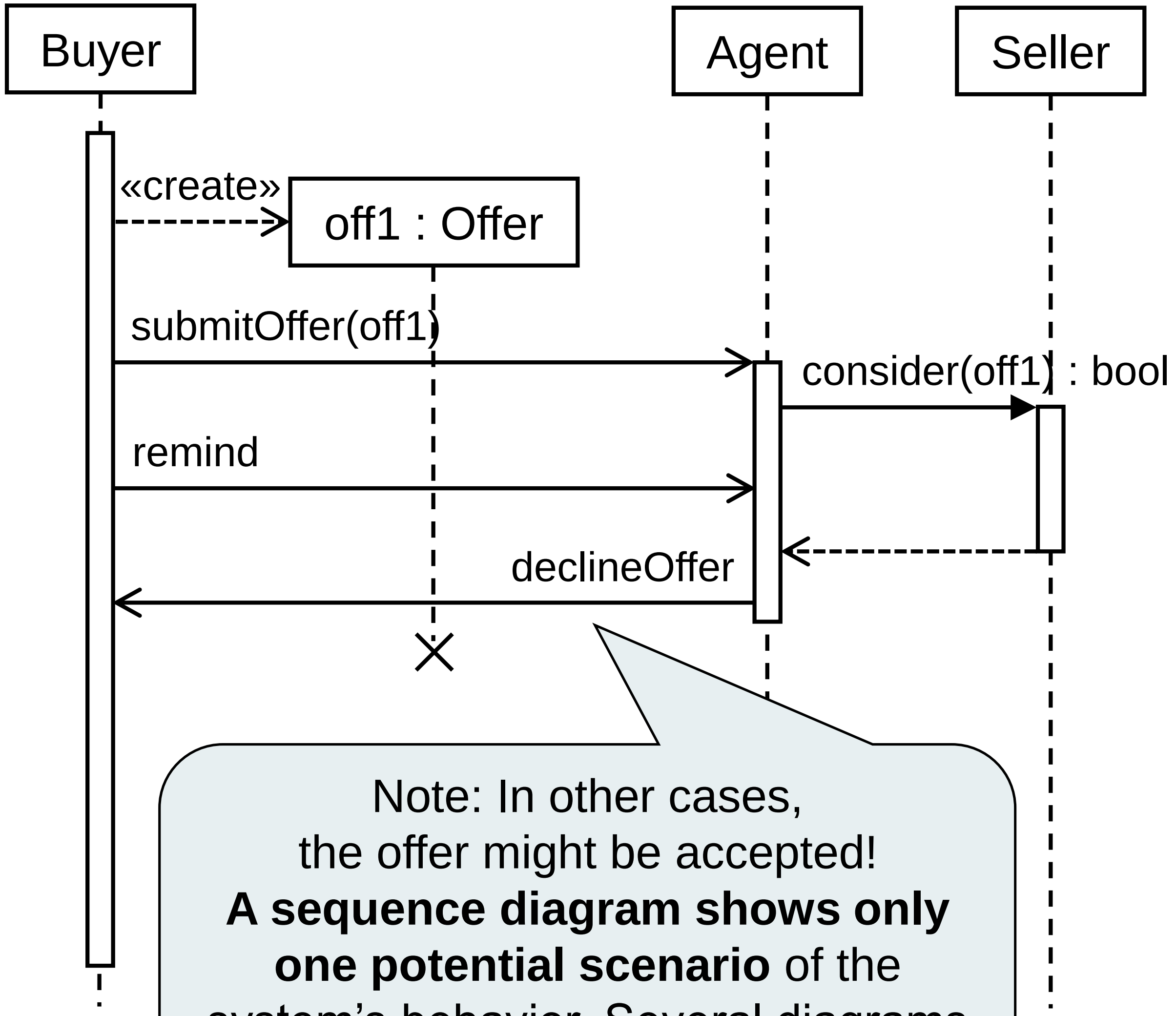
UML Sequence Diagrams

- **Purpose**
  - Visualize how several participants interact through the exchange of messages over time
  - Domain model: “What messages are exchanged by which actors?”
  - Design model: “What methods are called on which objects?”

- **Features**
  - Visualizing interaction as message exchange / method calls
  - Makes sequence (order) of messages explicit
  - Clearly showing the messages is more important than clearly showing the process
    - For clear visualization of process rather than messages, choose UML Activity Diagram
    - For precise definition of timing, choose UML Timing Diagram

Recap: UML Sequence Diagrams *(→ HBV401G, Ch. 9)*

- **Participants** can be objects, □ packages, components, other actors
- **Synchronous message** ──▶
  - The message sender waits until the message receiver responds to the invocation.
- **Return message** ←‑‑‑‑‑‑‑‑‑‑
  - Control flow returns after invocation of synchronous message
- **Asynchronous message** ──→
  - The message sender does not wait for the receiver's response but remains in control after sending.

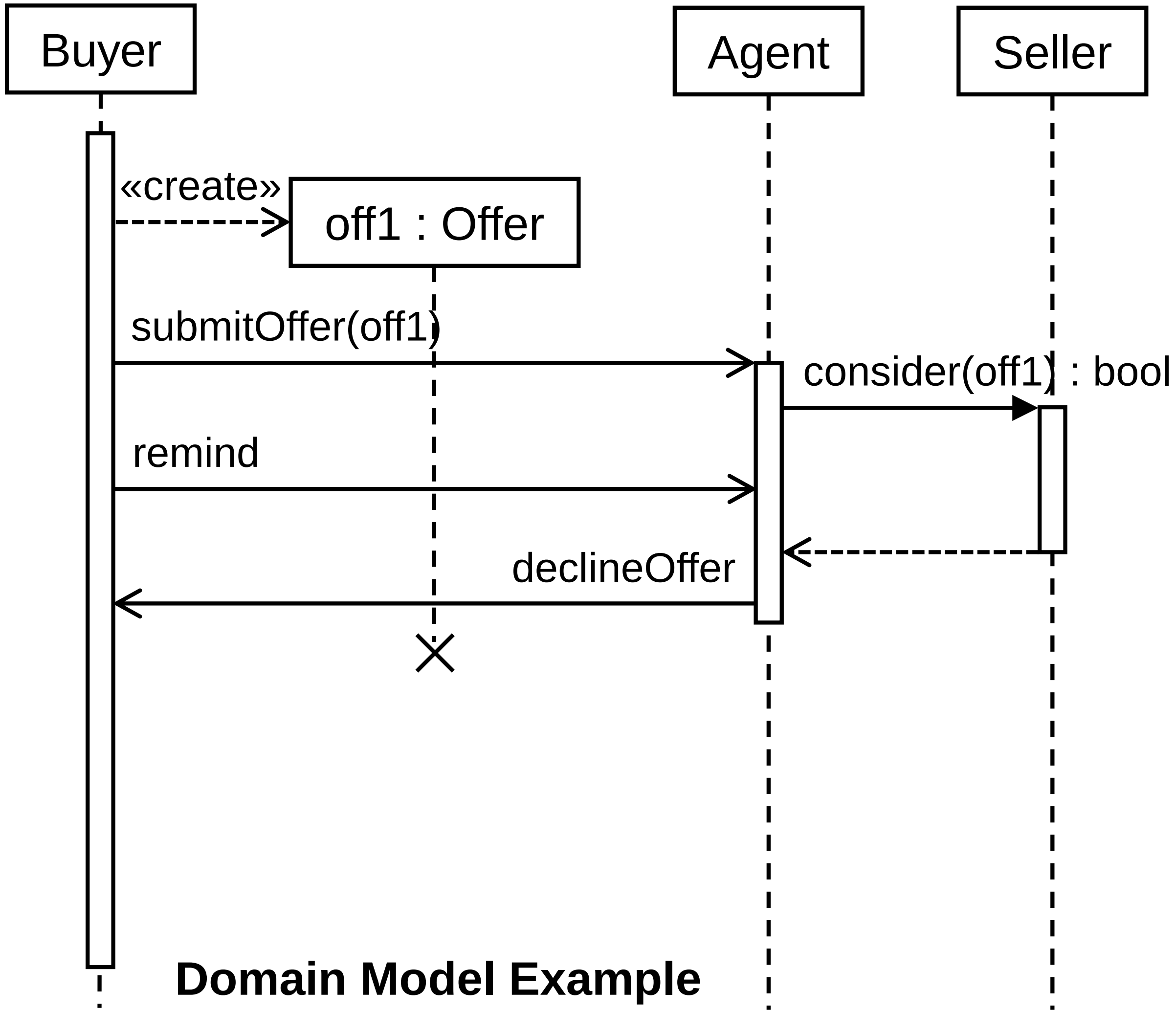


Note: In other cases, the offer might be accepted! **A sequence diagram shows only one potential scenario** of the system's behavior. Several diagrams may be needed to show all variants.

Recap: UML Sequence Diagrams in Domain Models

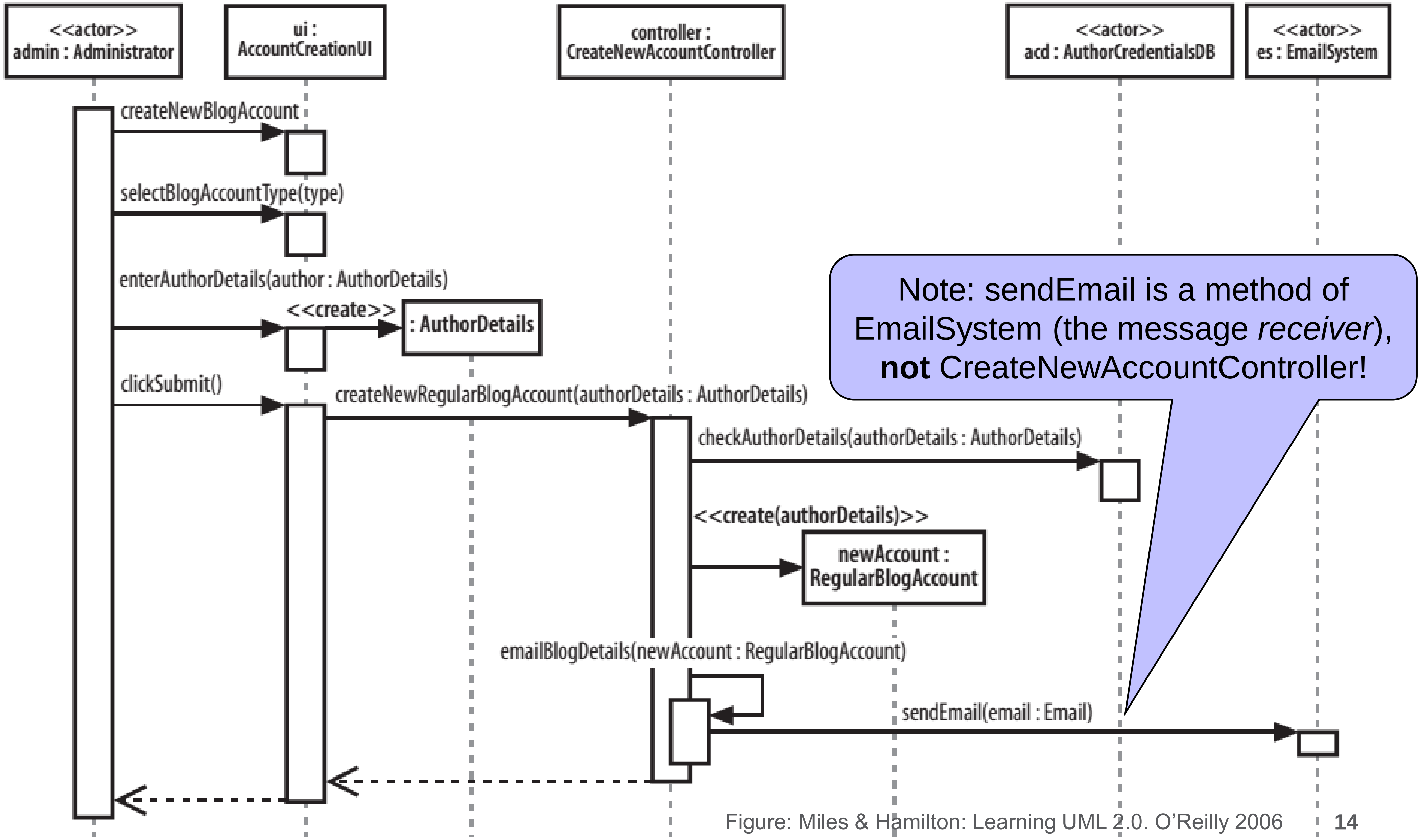
- Participants exchange messages
- Messages are symbolized as arrows
- **Caution:**
  - In a **domain model**, arrows are often labeled according to content of message
    - Note: This is often some activity written from the message *sender's* perspective
  - In a **design model**, arrows are always labeled with the called method's name
    - Note: This is necessarily written from the message *receiver's* perspective!

➢ Be aware of model type (domain vs. design model) when drawing and interpreting a sequence diagram!



Domain Model Example

Example: UML Sequence Diagram in Design Model



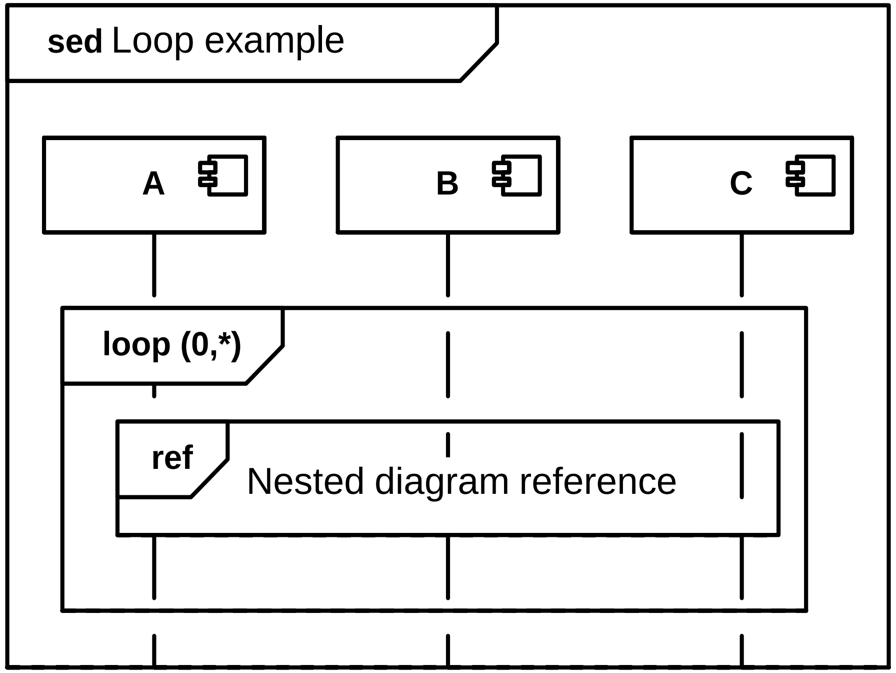
<<actor>> admin : Administrator
ui : AccountCreationUI
controller : CreateNewAccountController
<<actor>> acd : AuthorCredentialsDB
<<actor>> es : EmailSystem
createNewBlogAccount
selectBlogAccountType(type)
enterAuthorDetails(author : AuthorDetails)
<<create>>
: AuthorDetails
clickSubmit()
createNewRegularBlogAccount(authorDetails : AuthorDetails)
checkAuthorDetails(authorDetails : AuthorDetails)
<<create(authorDetails)>>
newAccount : RegularBlogAccount
emailBlogDetails(newAccount : RegularBlogAccount)
sendEmail(email : Email)

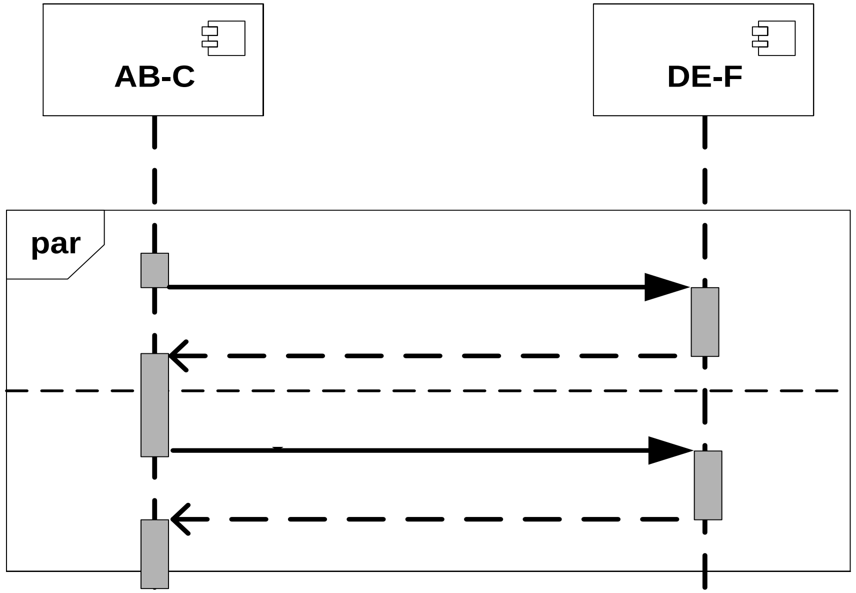
Note: sendEmail is a method of EmailSystem (the message *receiver*), **not** CreateNewAccountController!

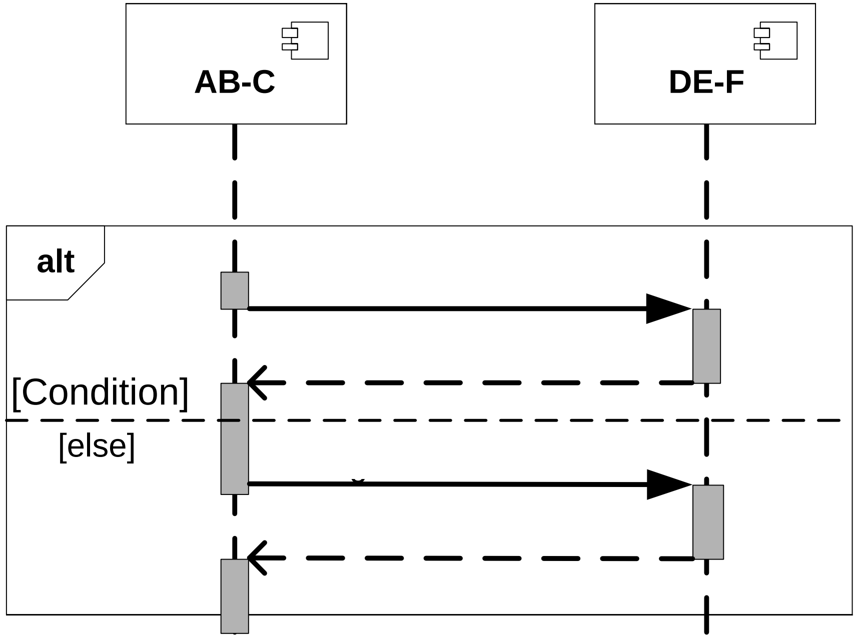
Figure: Miles & Hamilton: Learning UML 2.0. O'Reilly 2006

UML Sequence Diagram Fragments

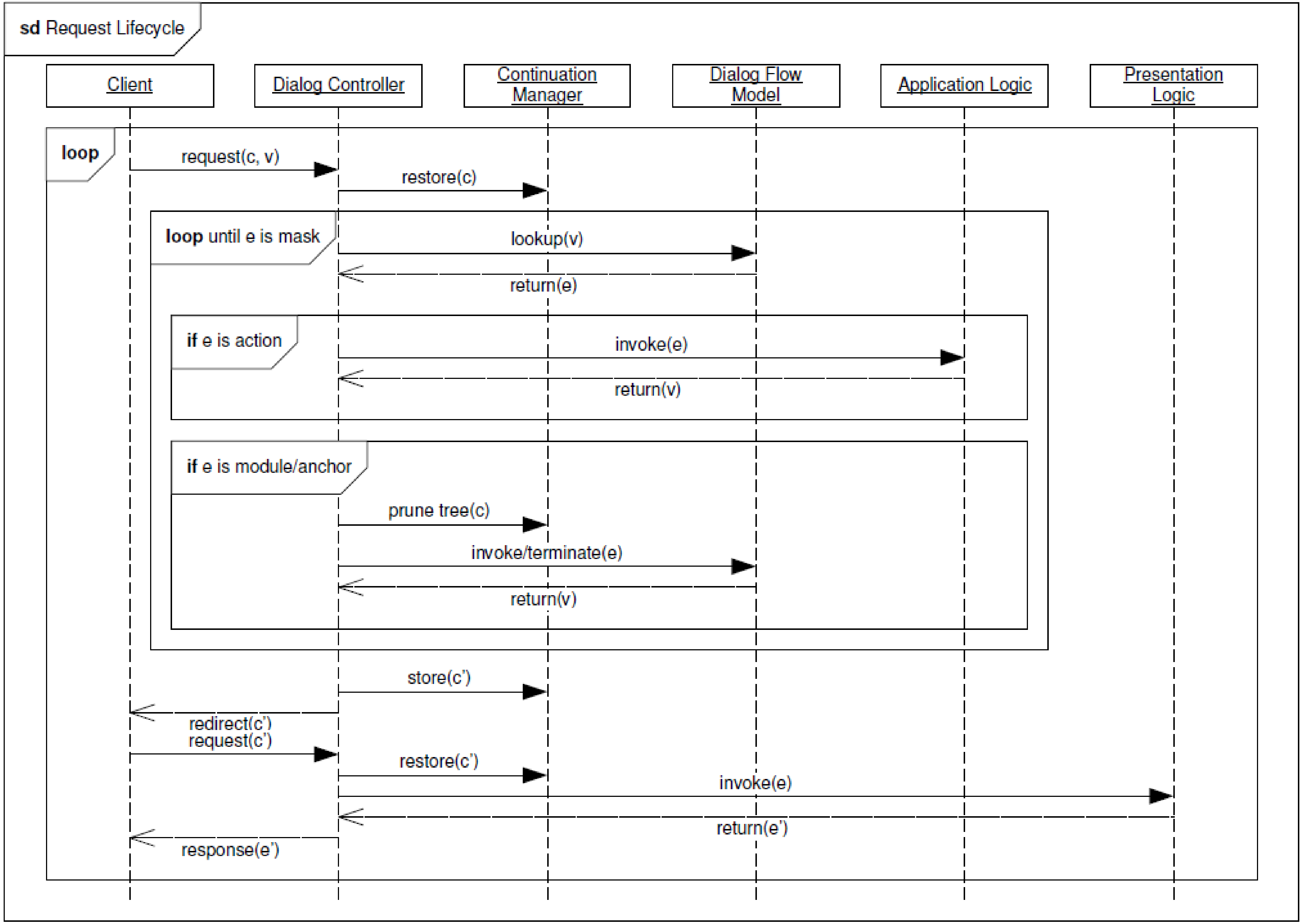
- **par:** Parallel execution
- **alt:** Alternative execution
- **loop:** Looped execution
- **ref:** Nested execution





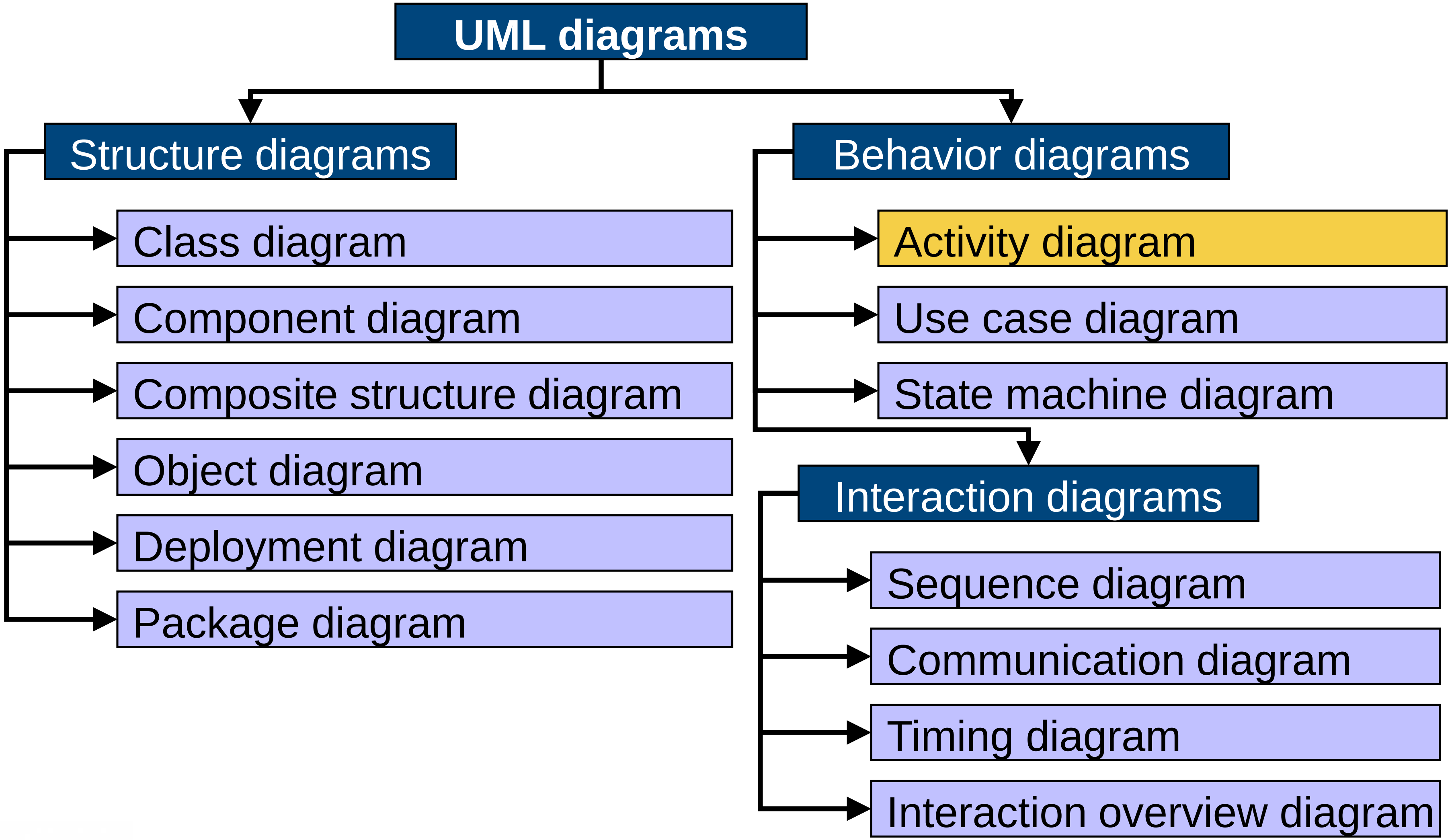


Case Study: A More Complex Sequence Diagram



HÁSKÓLI ÍSLANDS

UML Diagram Types



UML diagrams

Structure diagrams

- Class diagram
- Component diagram
- Composite structure diagram
- Object diagram
- Deployment diagram
- Package diagram

Behavior diagrams

- Activity diagram
- Use case diagram
- State machine diagram

Interaction diagrams

- Sequence diagram
- Communication diagram
- Timing diagram
- Interaction overview diagram

UML Activity Diagrams

- **Purpose**
  - Visualize the individual steps/actions that constitute a particular process/activity
  - Domain model: “How does a process work?”
  - Design model: “How does our system implement certain behavior?”

- **Features**
  - Visualizing processes with conditions, branches and loops
  - Expressing concurrency and synchronization
  - Visualizing data flows
  - Hierarchical nesting

Example: Blog Account Creation

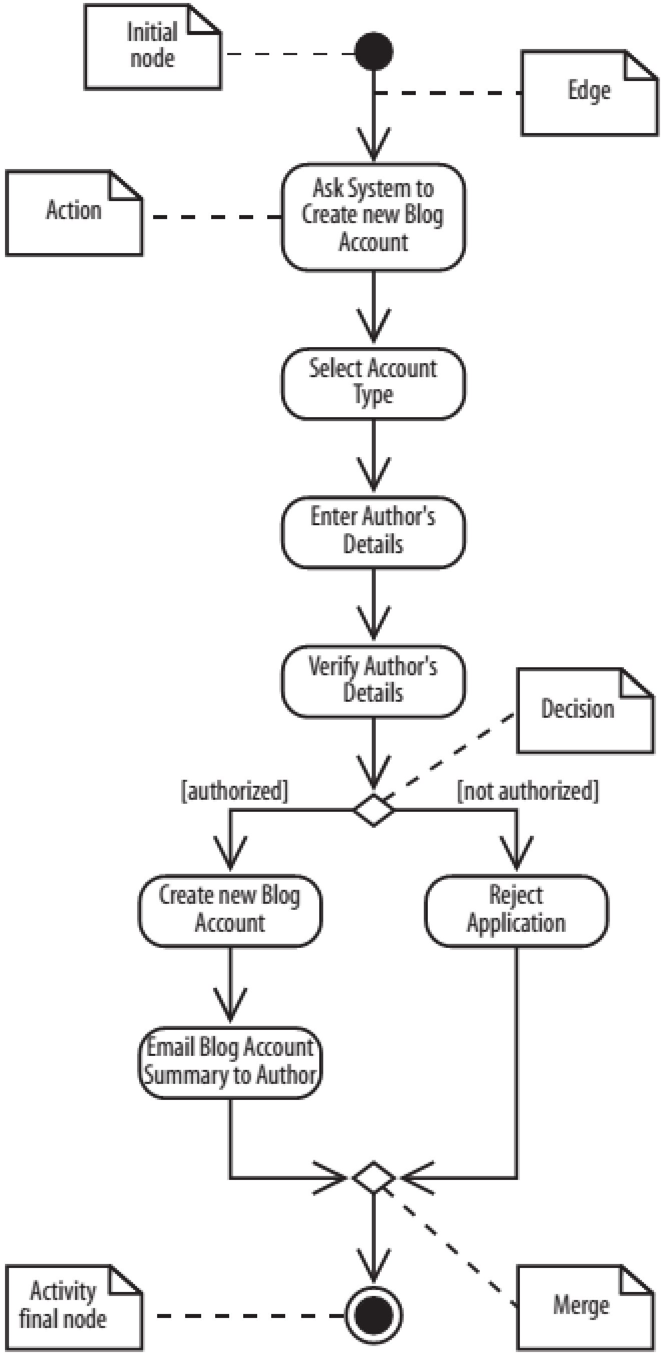
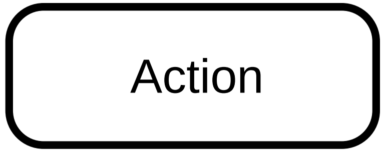
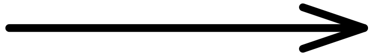


Figure: Miles & Hamilton: Learning UML 2.0. O'Reilly 2006

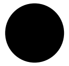
Basic Elements



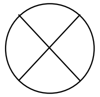
- An **action** is one step of the activity
    - Sum of all actions constitutes the **activity**



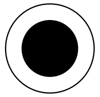
- Control flow



- Initial node



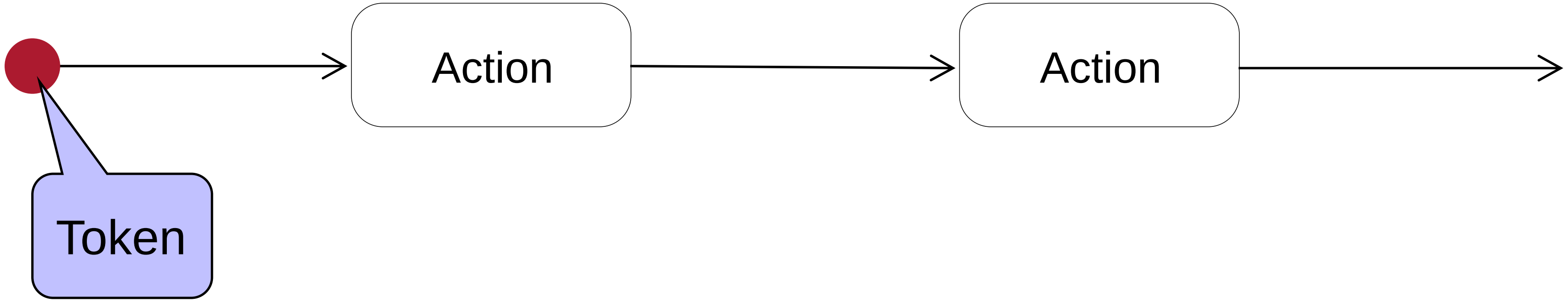
- Final node of a control flow branch



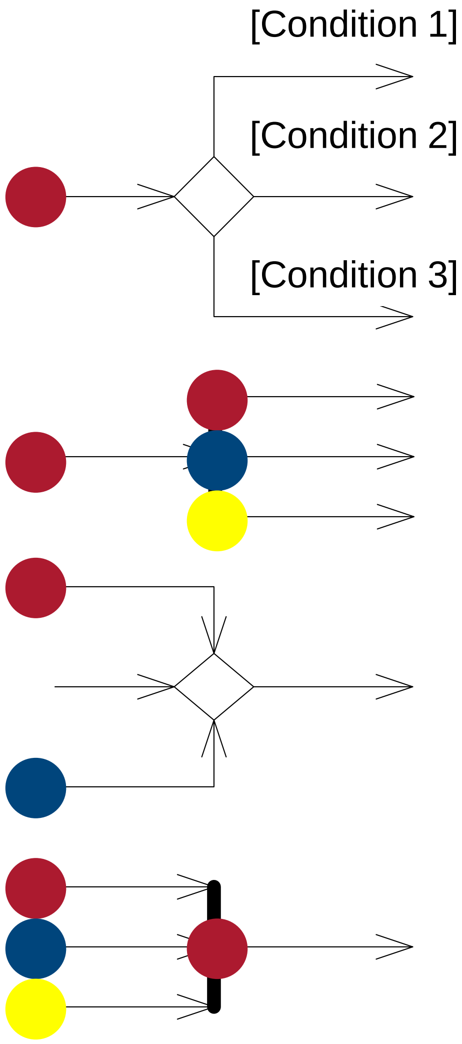
- Final node of activity

Token Concept

- Based on the concept of Petri nets (place/transition nets, P/T net)
- Control flow is simulated by tokens
- An action is executed when a token reaches its inbound edge
- When the action has been executed, a token leaves its outbound edge(s)

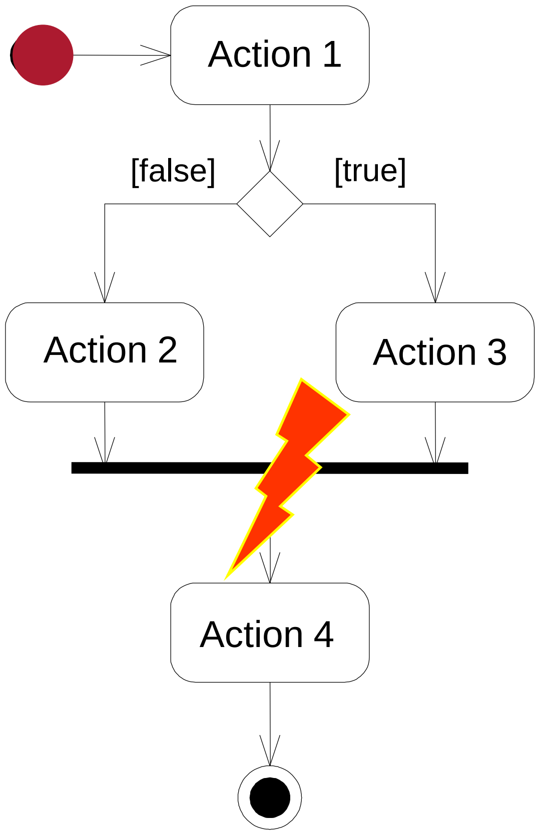


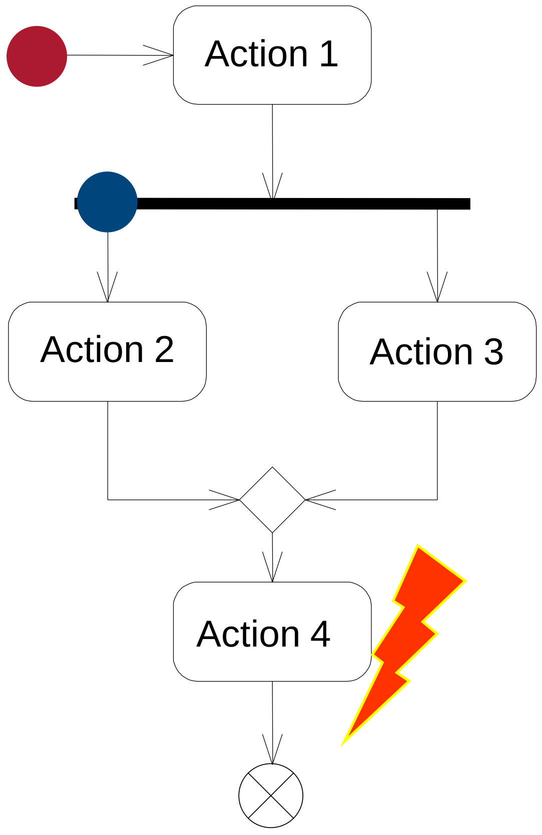
Token Concept for Control Flow



- **Decision nodes** create a token only on one outbound edge, depending on which condition is satisfied.
  - If several conditions are satisfied, the choice of outbound edge is undetermined.
- **Fork nodes** create a token on each outbound edge.
- **Merge nodes** create a token on the outbound edge whenever a token reaches the inbound edge.
- **Join nodes** create a token on the outbound edge only when a token has arrived on each inbound edge.

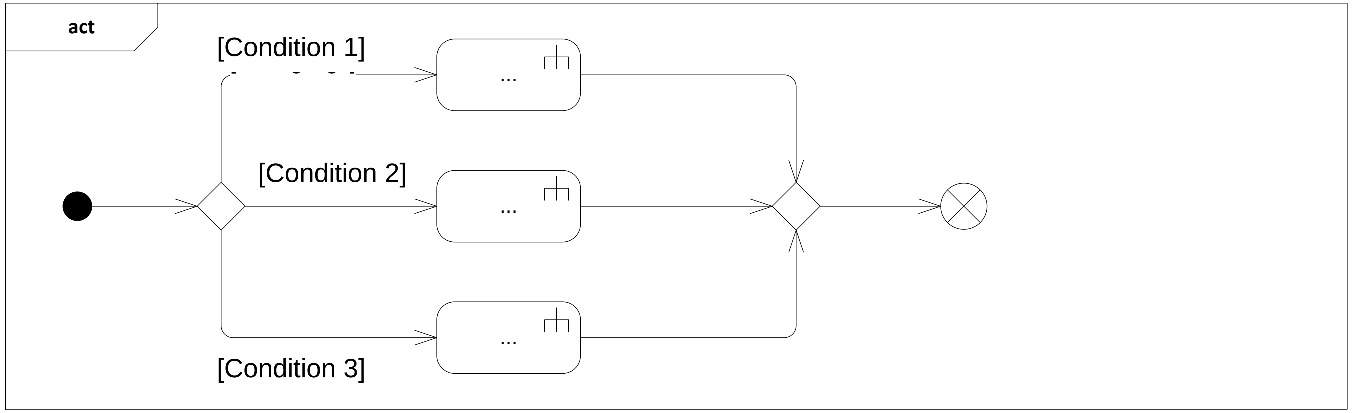
Modeling Errors

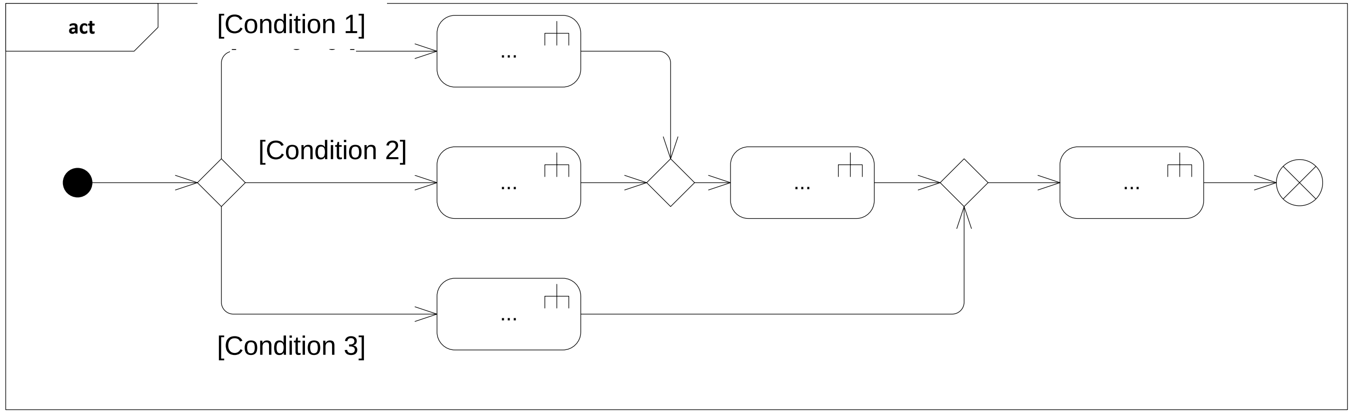




Modeling Rules

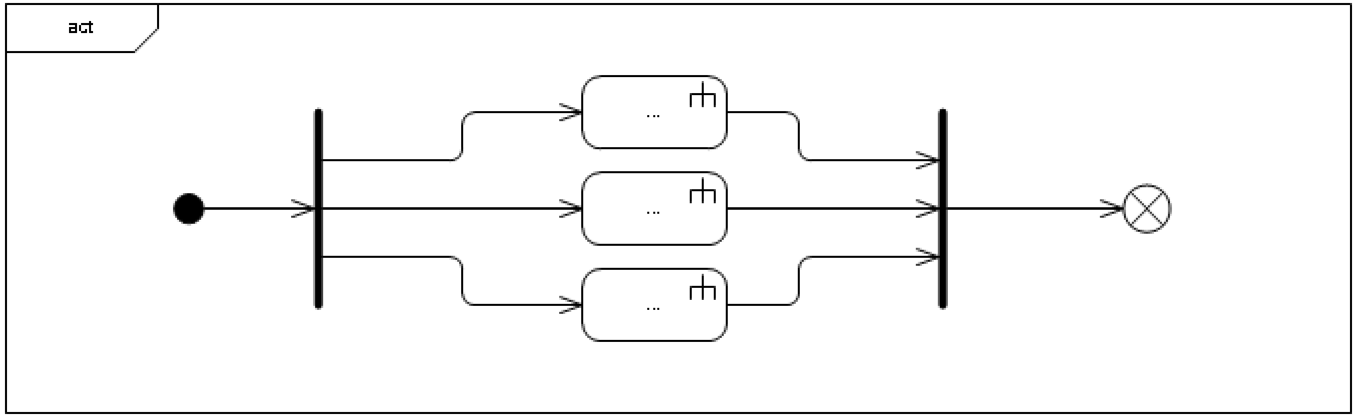
- Control flows separated by decision nodes must be recombined by merge nodes



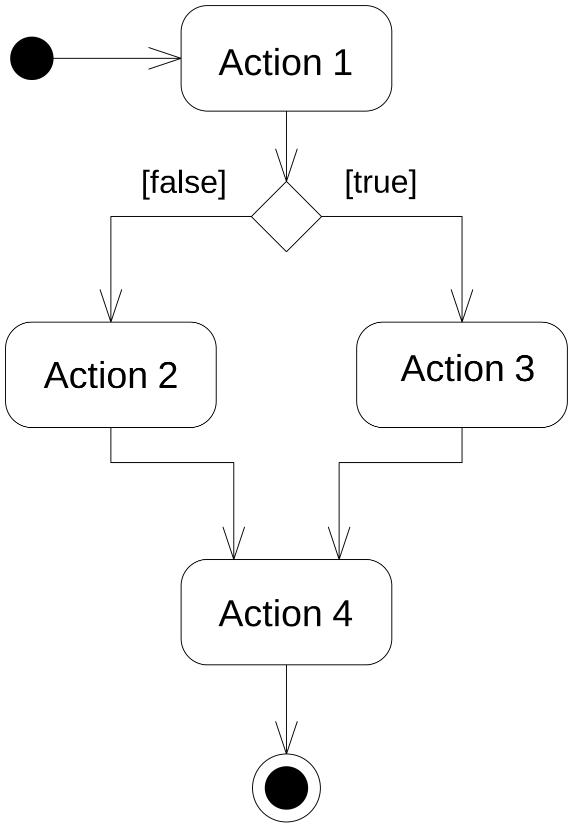


Modeling Rules

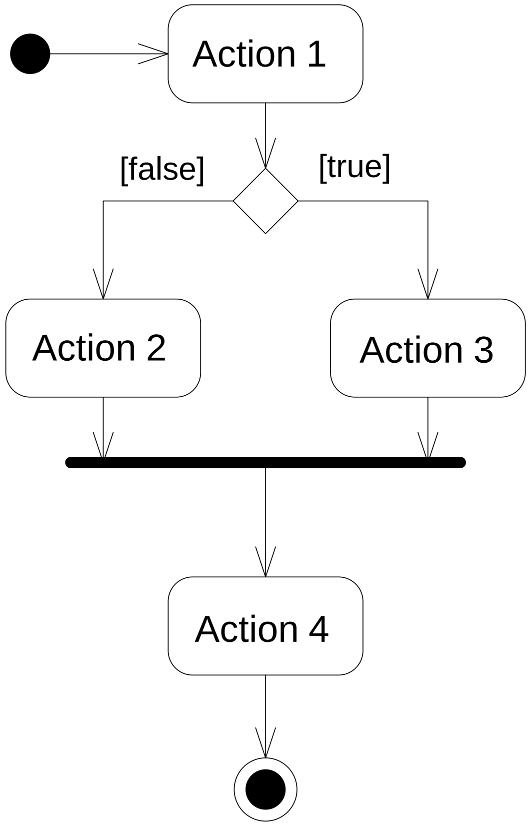
- Control flows split by fork nodes must be recombined by join nodes



Modeling Errors



≙



UML1.x vs. UML 2.0

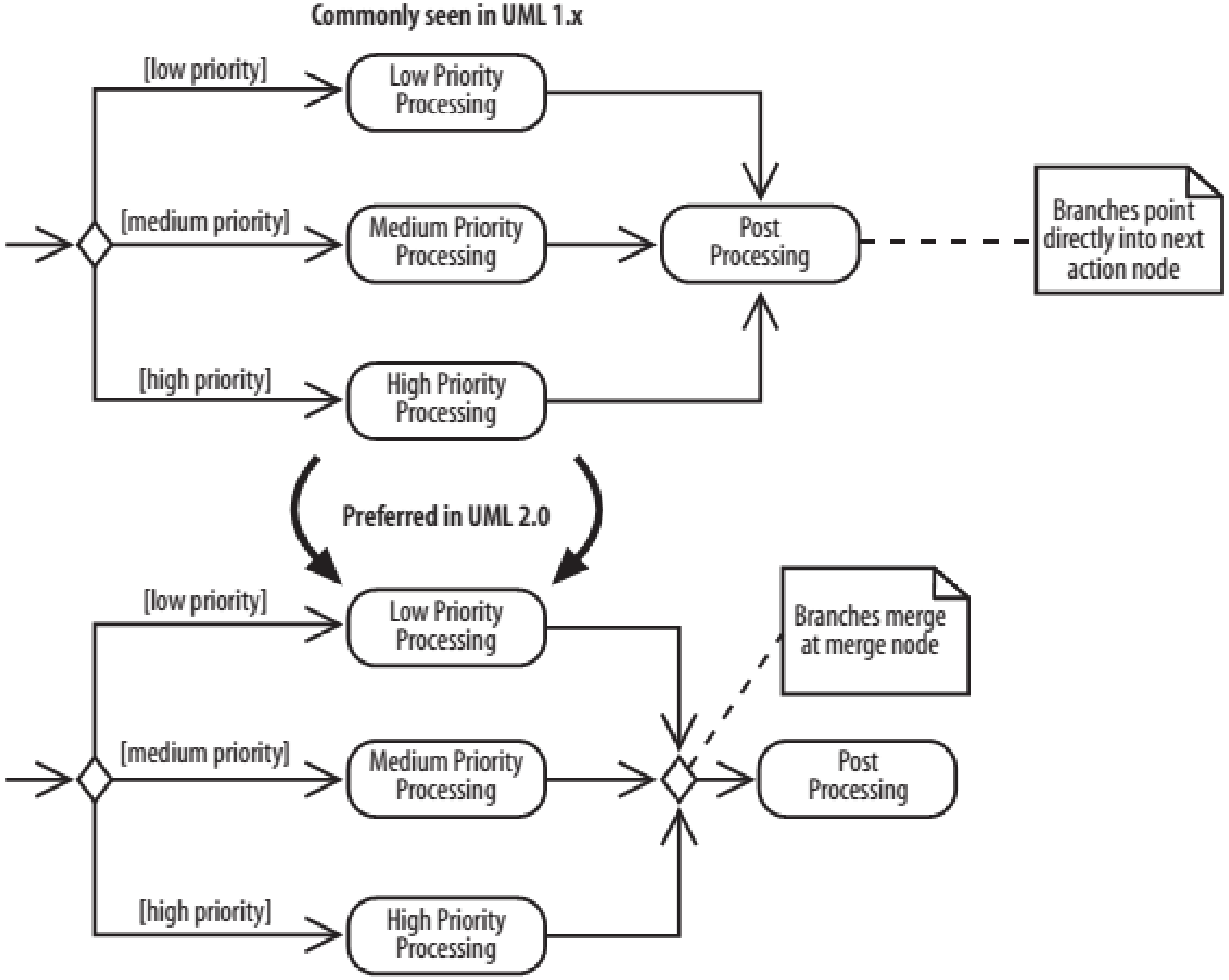
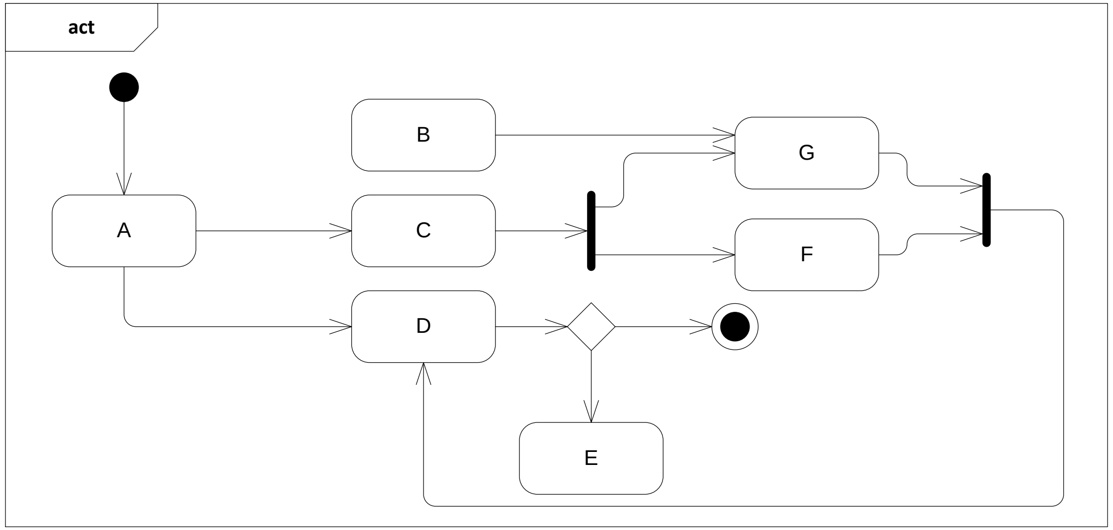


Figure: Miles & Hamilton: Learning UML 2.0. O’Reilly 2006

Modeling Rules

- An action must have exactly one inbound and one outbound node.
- An activity diagram must have exactly one initial node and at least one final node.

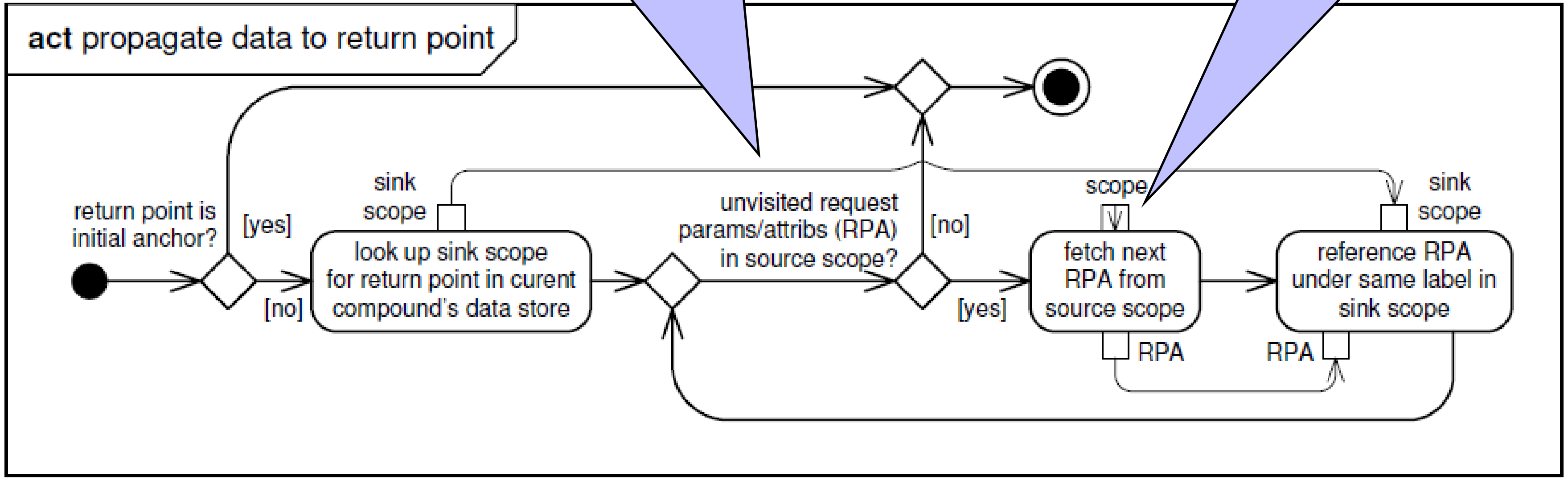


Modeling Data Flows in Activity Diagrams

- Pin notation

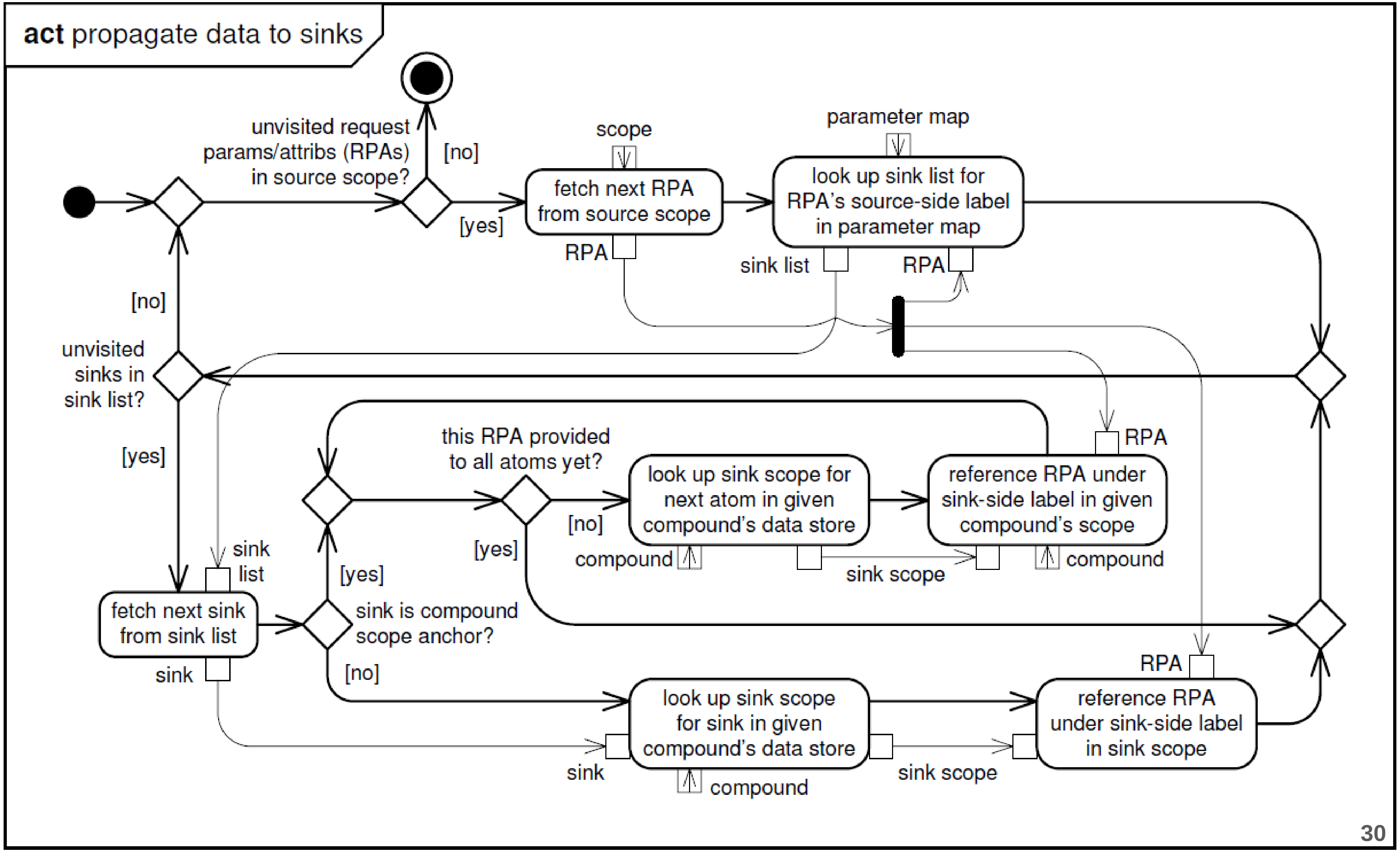
Flow of **data** “sink scope” from one action to another

Data “scope” coming in from this activity’s invocation context



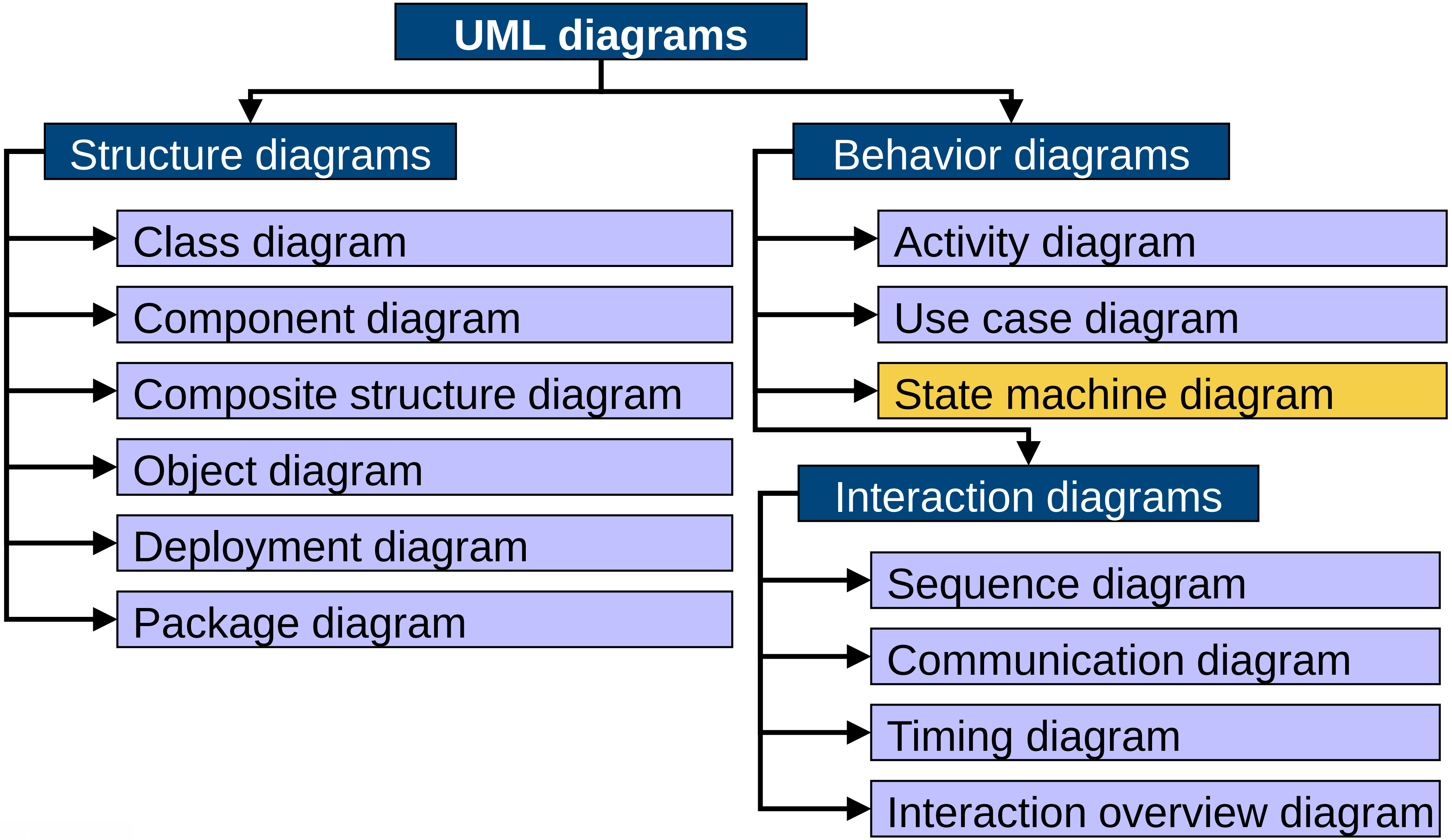
act propagate data to return point
return point is initial anchor?
[yes]
[no]
sink scope
look up sink scope for return point in curent compound’s data store
unvisited request params/attribs (RPA) in source scope?
[no]
[yes]
scope
fetch next RPA from source scope
RPA
sink scope
reference RPA under same label in sink scope
RPA

Case Study: A More Complex Activity Diagram



act propagate data to sinks
unvisited request params/attribs (RPAs) in source scope?
[no]
[yes]
scope
fetch next RPA from source scope
RPA
parameter map
look up sink list for RPA's source-side label in parameter map
sink list
RPA
[no]
unvisited sinks in sink list?
[yes]
this RPA provided to all atoms yet?
[no]
[yes]
look up sink scope for next atom in given compound's data store
compound
reference RPA under sink-side label in given compound's scope
RPA
sink scope
compound
sink list
fetch next sink from sink list
sink
[yes]
sink is compound scope anchor?
[no]
look up sink scope for sink in given compound's data store
sink
compound
sink scope
RPA
reference RPA under sink-side label in sink scope

UML Diagram Types

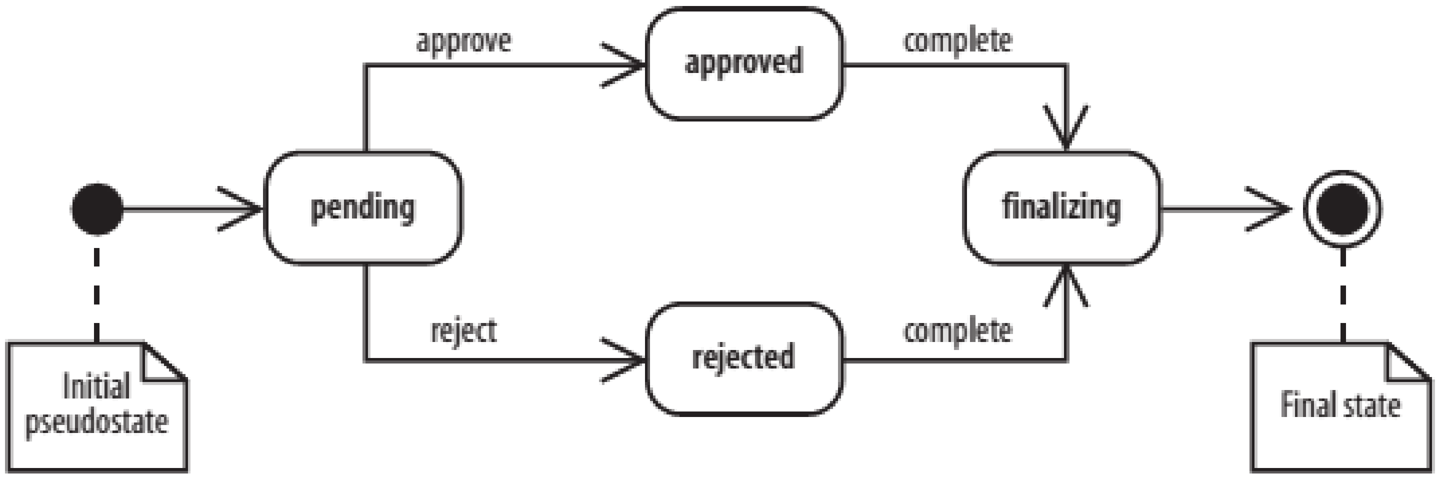


UML State Machine Diagrams

- **Purpose**
  - Visualizing the states that an object, interface, component etc. can adopt, depending on certain events

- **Features**
  - Precise modeling of states, events, concurrency, conditions, input and output operations
  - Hierarchical nesting

Example: Blog Account Creation

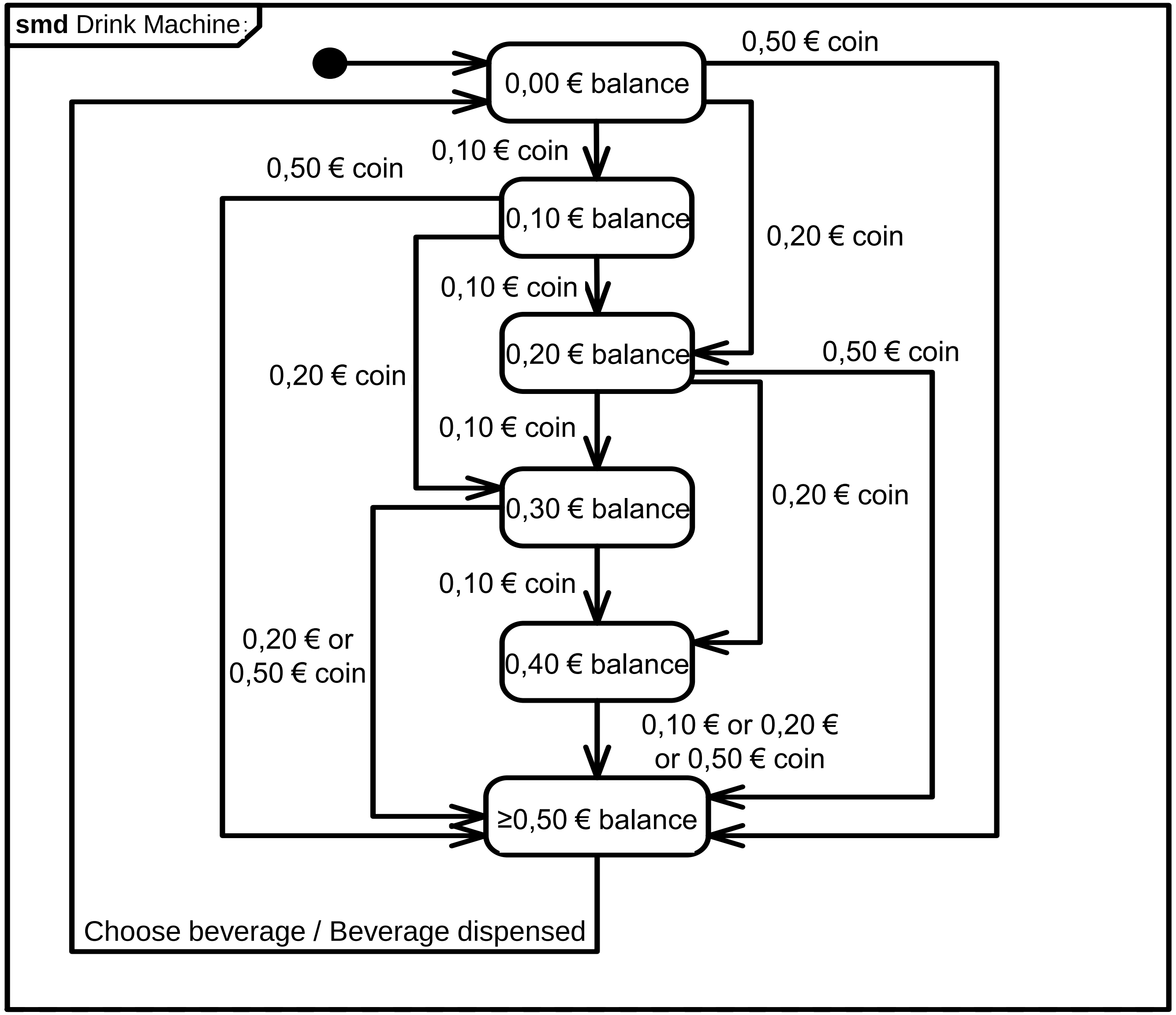


- At any point in time, the system is in exactly one state.
- A transition from the current to the next state occurs instantaneously
    - i.e. it does not take time

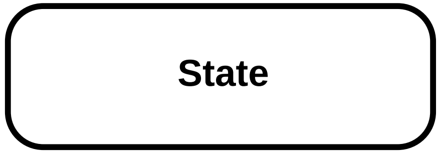
Figure: Miles & Hamilton: Learning UML 2.0. O’Reilly 2006

Example: Automatic Beverage Dispenser

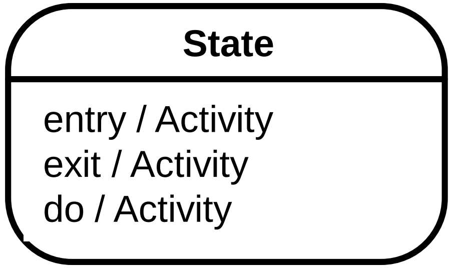
- Alle beverages cost 0,50 €
- The machine accepts 0,10 €, 0,20 € and 0,50 € coins
- No change is given
- After paying at least 0,50 €, the user can choose a drink and receives it



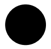
Notation Elements



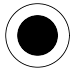
- A **state** that the system can be it at one time



- Activities can be performed upon certain events:
  - *entry* → activity performed upon entering state
  - *exit* → activity performed upon leaving state
  - *do* → activity performed while in this state



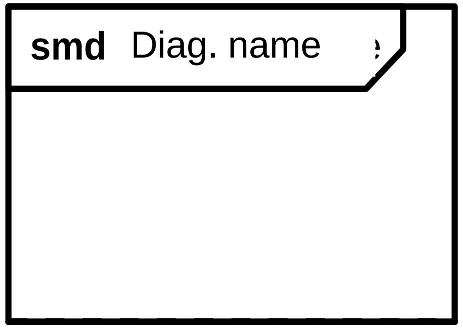
- Initial pseudo state



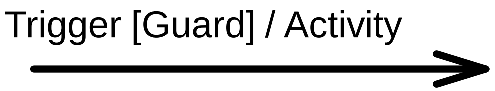
- Final state

- Only one initial state allowed per diagram!

Notation Elements



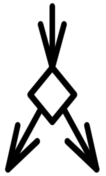
- State machine diagrams should be placed in a named frame
  - (smd / sm = state machine diagram)



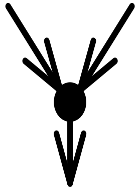
- **Triggers** are events prompting a state transition
- The specified **guard** is a condition that must be true for the transition to occur.
- A specified **activity** can be performed when the transition occurs.

Notation Elements

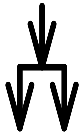
- **Pseudo states** are used to show complex relationships between states
- The system cannot be in a pseudo state
  - (i.e. passage through pseude states occurs instantaneously)



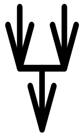
- **Choice pseudo state:** Next transition depends on guard condition



- **Merge pseudo state:** Combining inbound transitions from alternate states



- **Fork pseudo state:** Transition into several concurrent states.



- **Join pseudo state:** Combining incoming transitions from concurrent states

Composite States

- Forks beg the question: How can a system be in two states at once?
- Solution: **Composite states** define regions in each of which the system can only be in one state at a time.

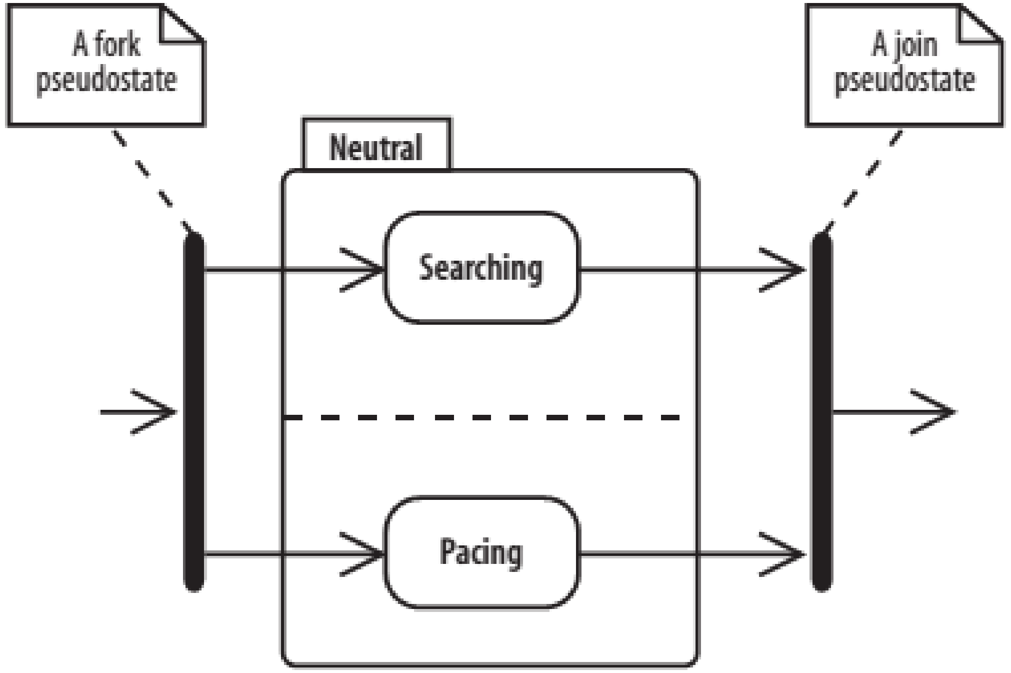
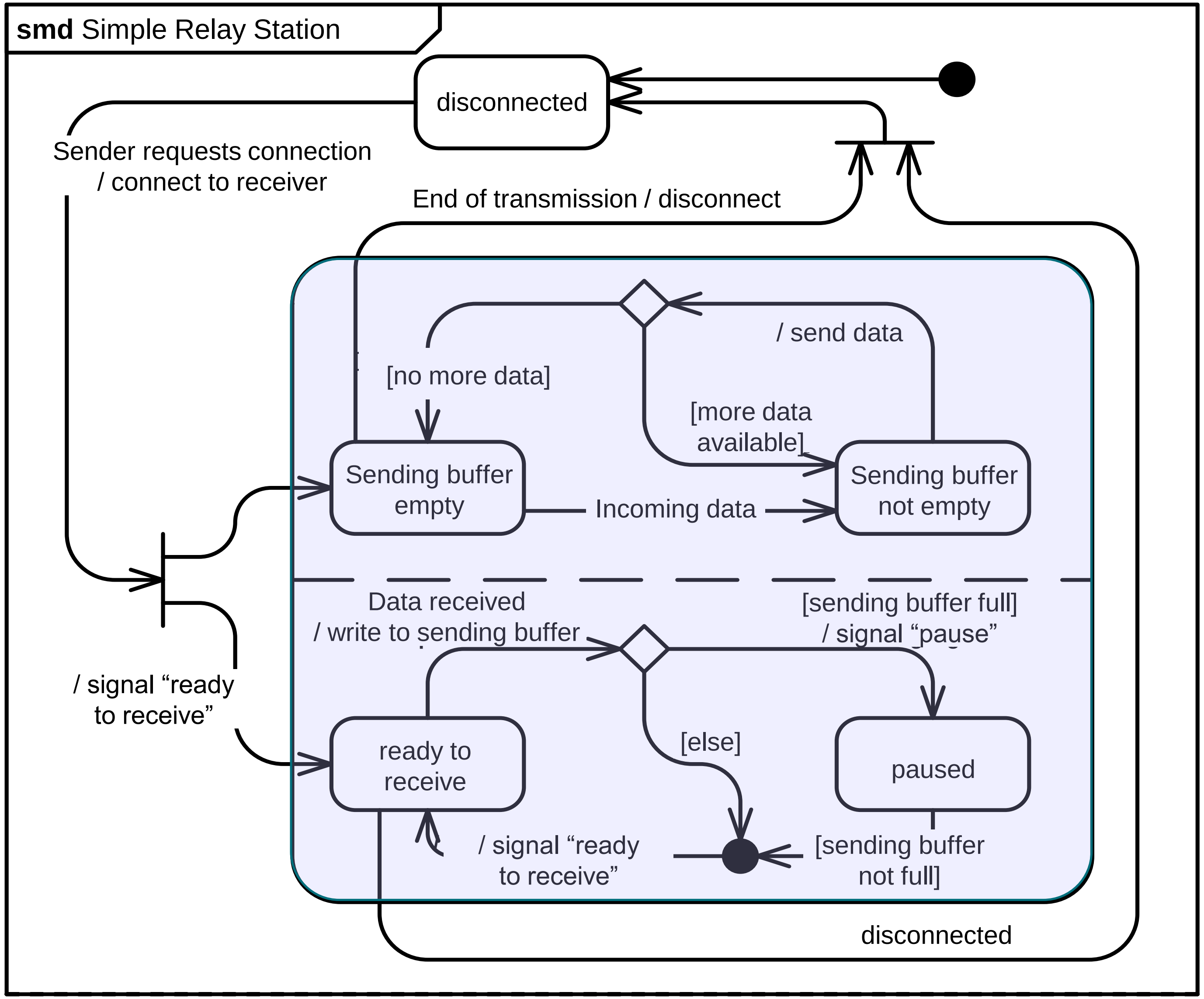


Figure: Miles & Hamilton: Learning UML 2.0. O'Reilly 2006

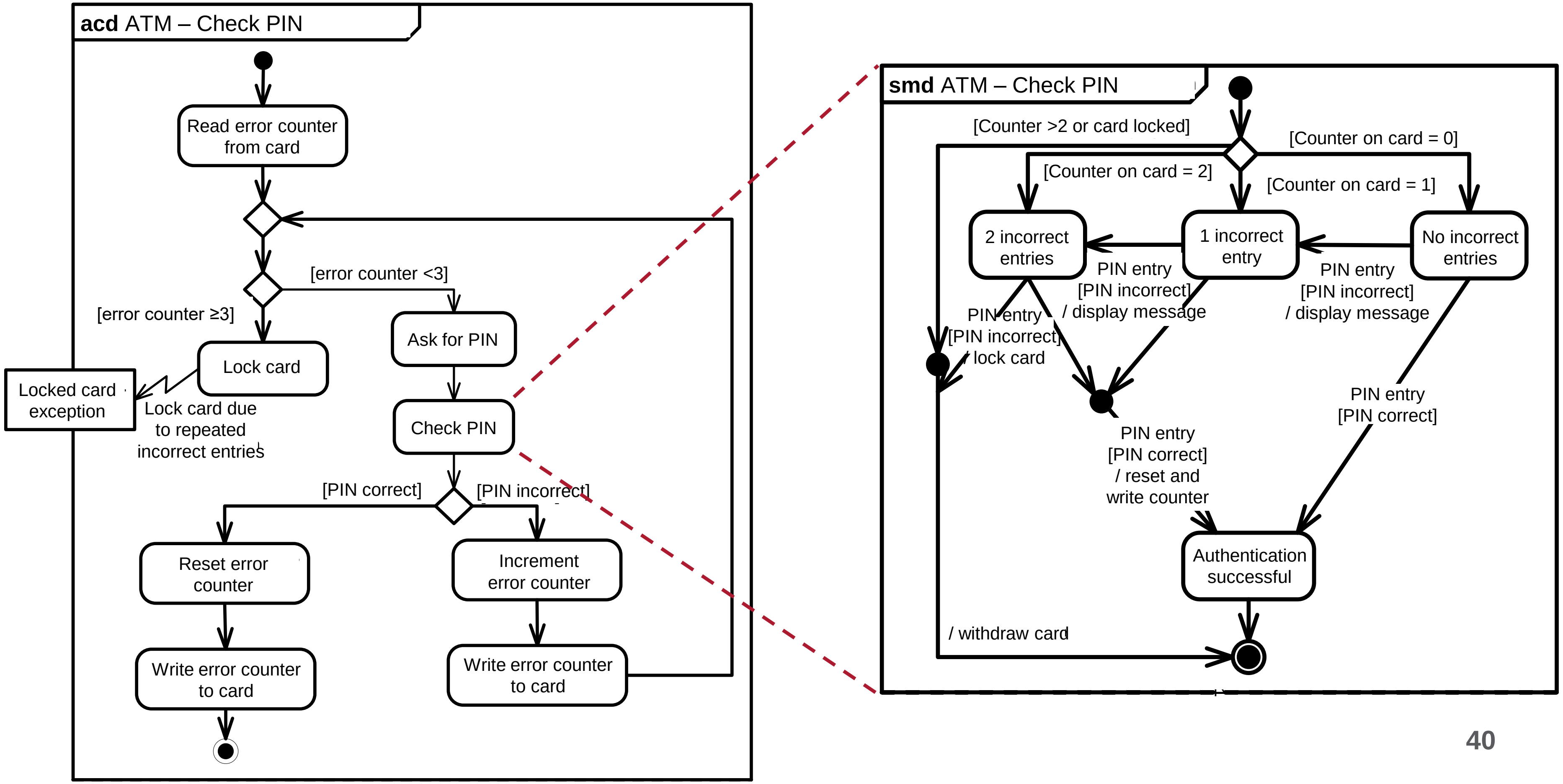
More Complex Example: A Relay Station



Activity Diagrams vs. State Machine Diagrams

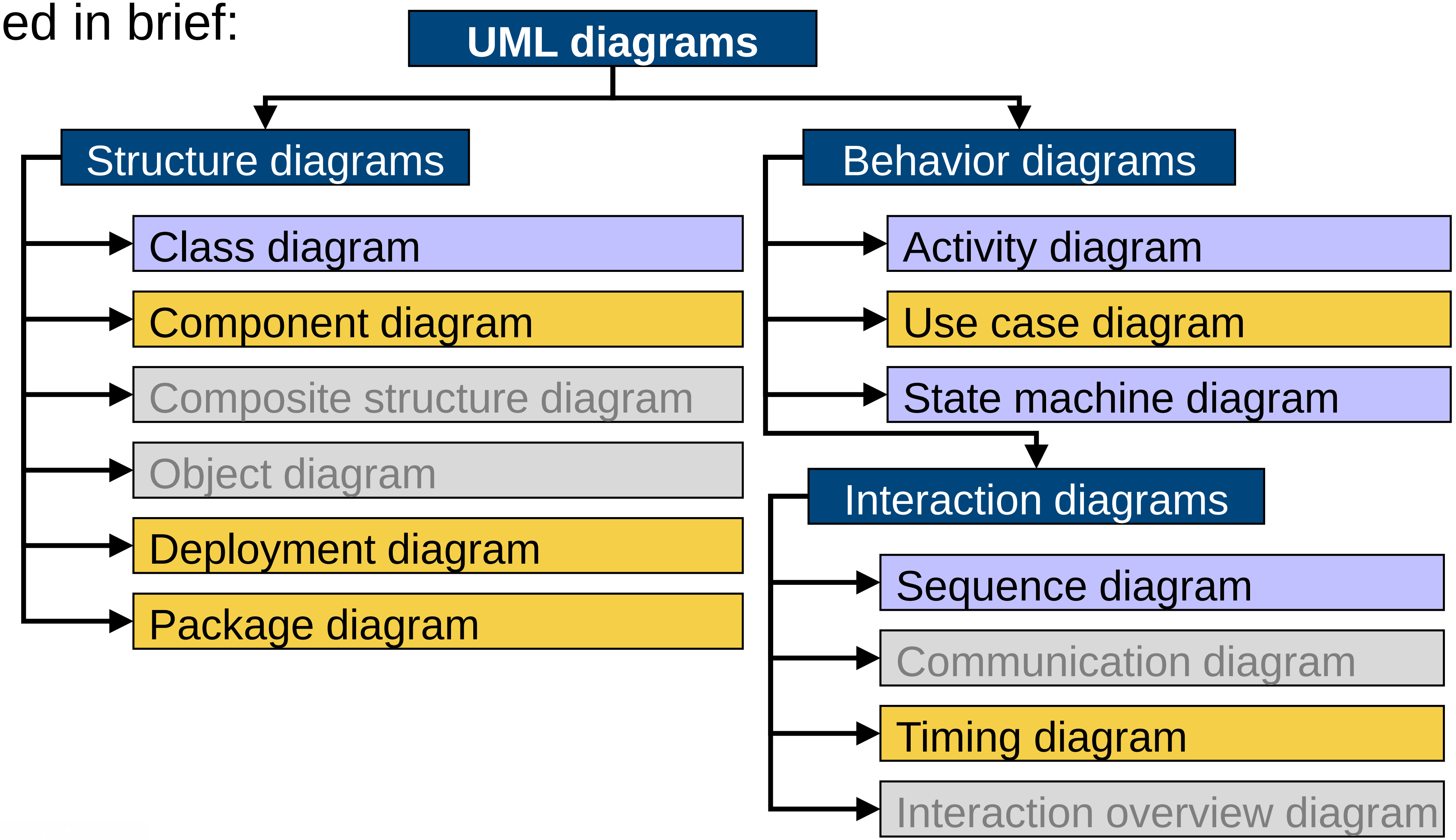
- **Activity diagrams describe what a component does**
- **State machine diagrams describe how a component changes**

- Example: ATM checking a card holder’s PIN
  - **acd:** Overall PIN checking process
  - **smd:** Keeping track of invalid entries



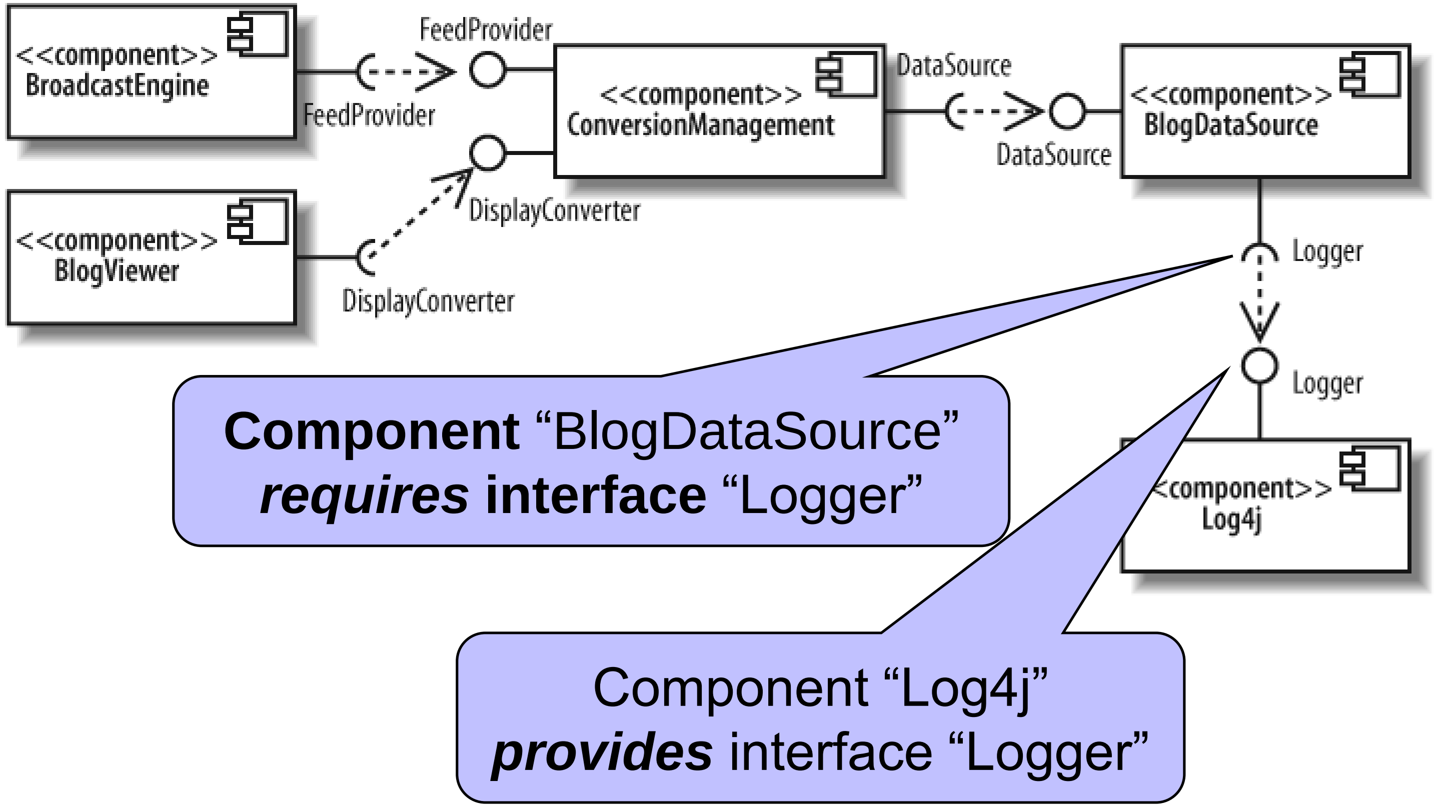
UML Diagram Types

Additional types introduced in brief:



UML Component Diagram

- Modeling high-level relationships / interfaces / dependencies between system components or with external components



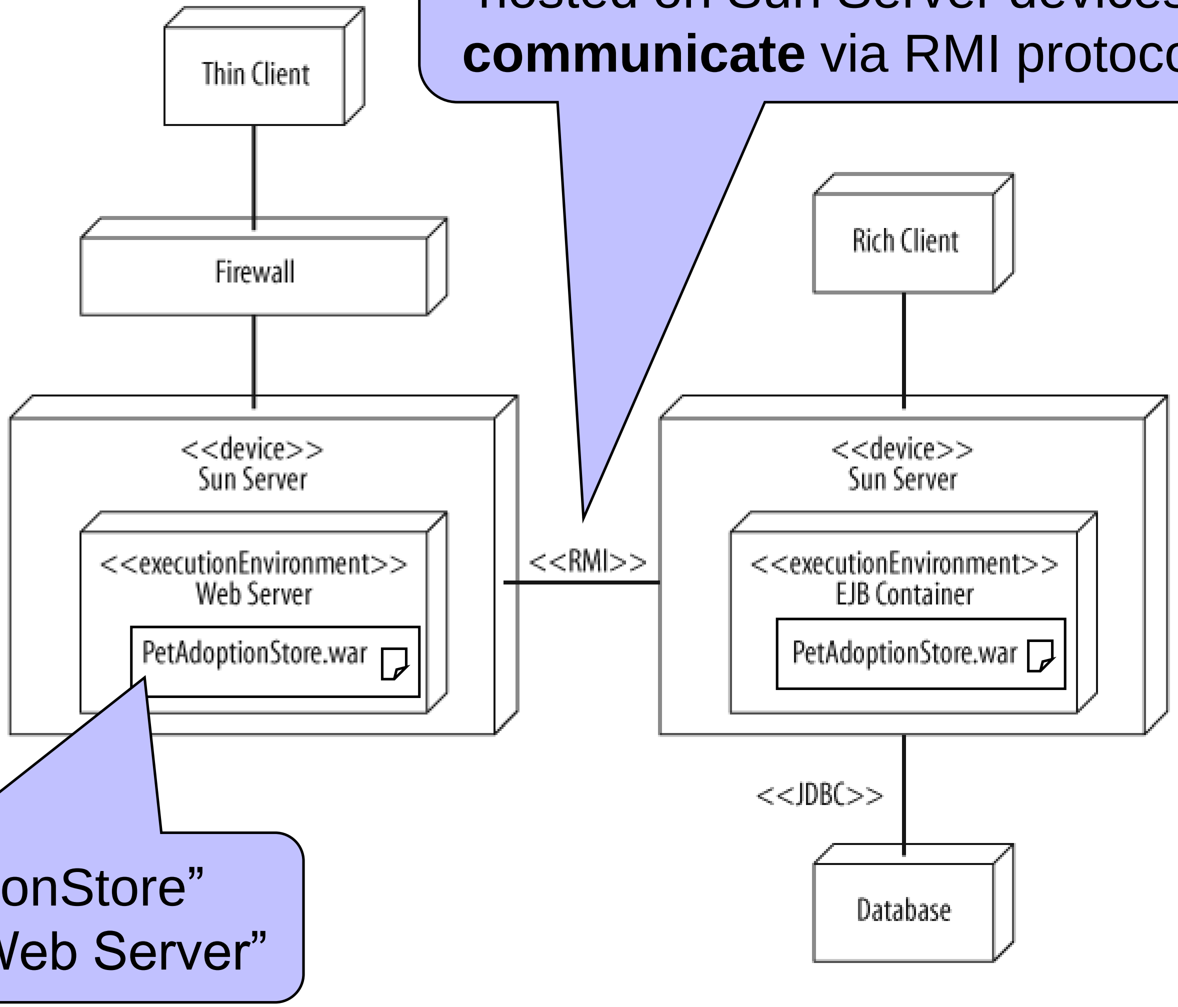
Component “BlogDataSource” ***requires*** **interface** “Logger”

Component “Log4j” ***provides*** interface “Logger”

Figure: Miles & Hamilton: Learning UML 2.0. O’Reilly 2006

UML Deployment Diagram

- Modeling deployment of software components on hardware nodes



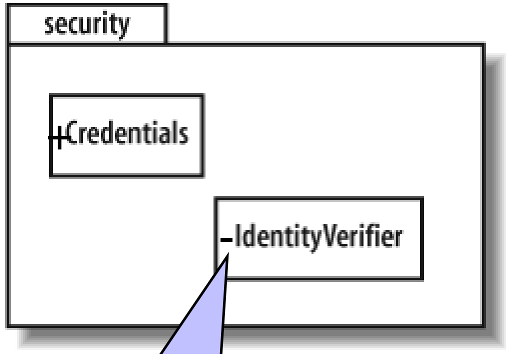
Web Server and EJB Container hosted on Sun Server devices **communicate** via RMI protocol

Artifact “PetAdoptionStore” **deployed** on **node** “Web Server”

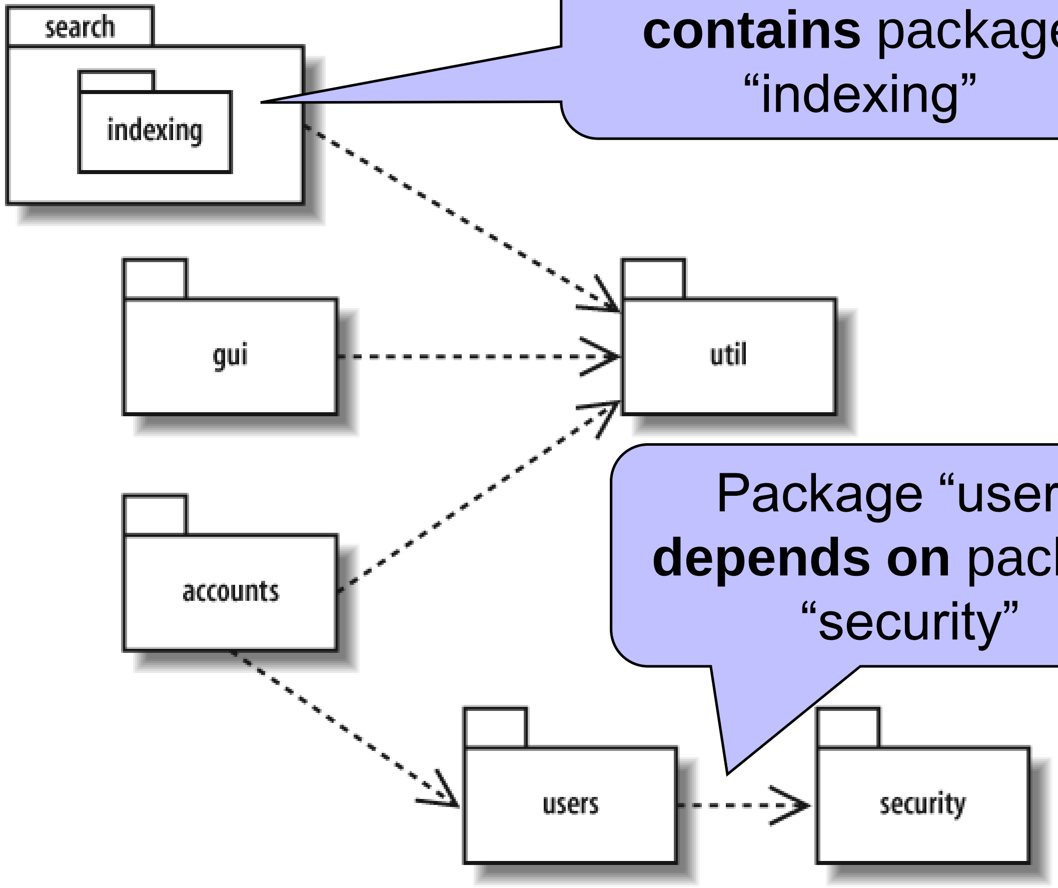
Figure: Miles & Hamilton: Learning UML 2.0. O’Reilly 2006

UML Package Diagram

- Modeling hierarchical structure (→ namespaces) of classes
- Modeling visibility of classes



Class “IdentityVerifier” not accessible outside its **package** “security”

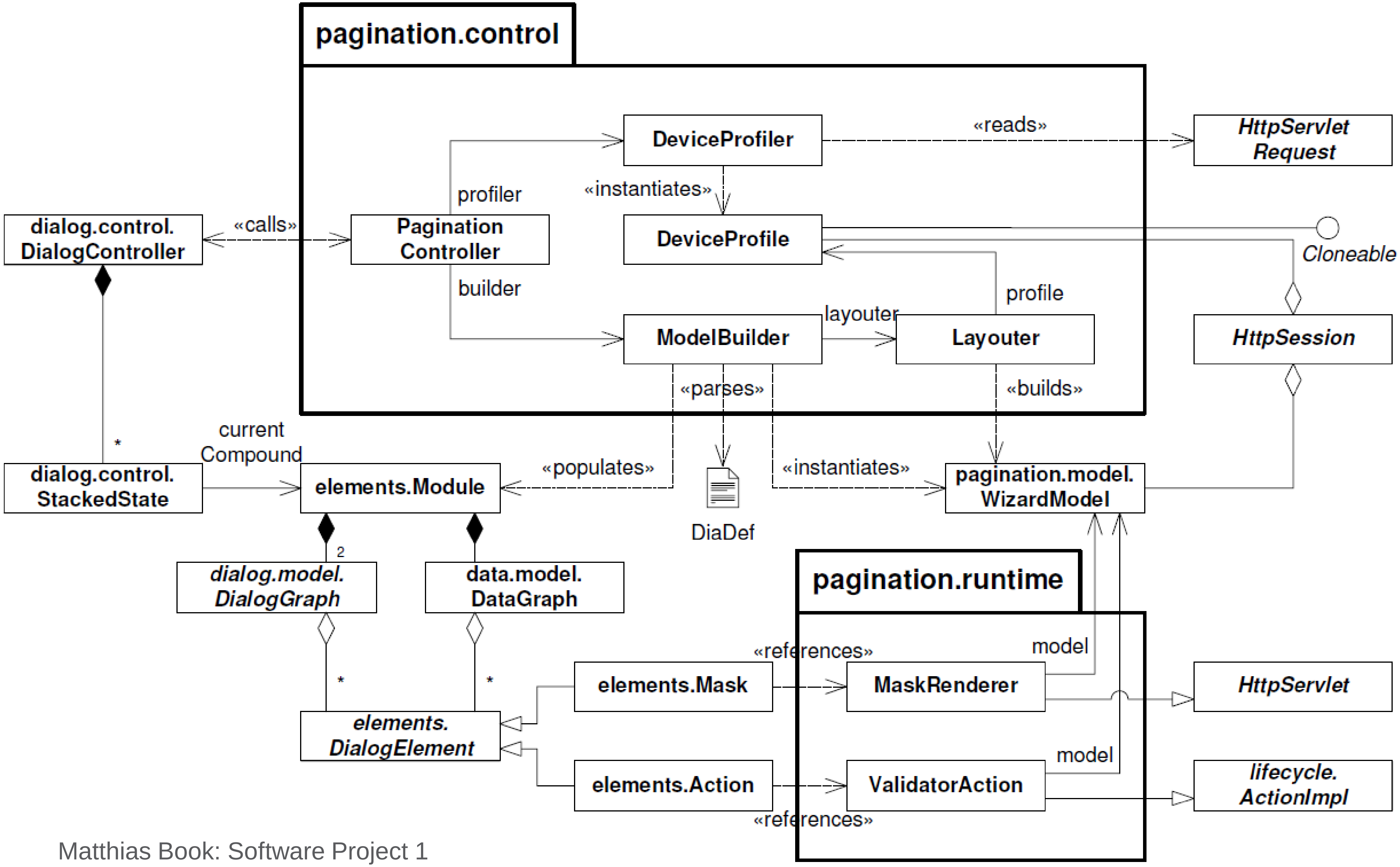


Package “search” **contains** package “indexing”

Package “users” **depends on** package “security”

Figure: Miles & Hamilton: Learning UML 2.0. O’Reilly 2006

Case Study: A More Complex Package Diagram

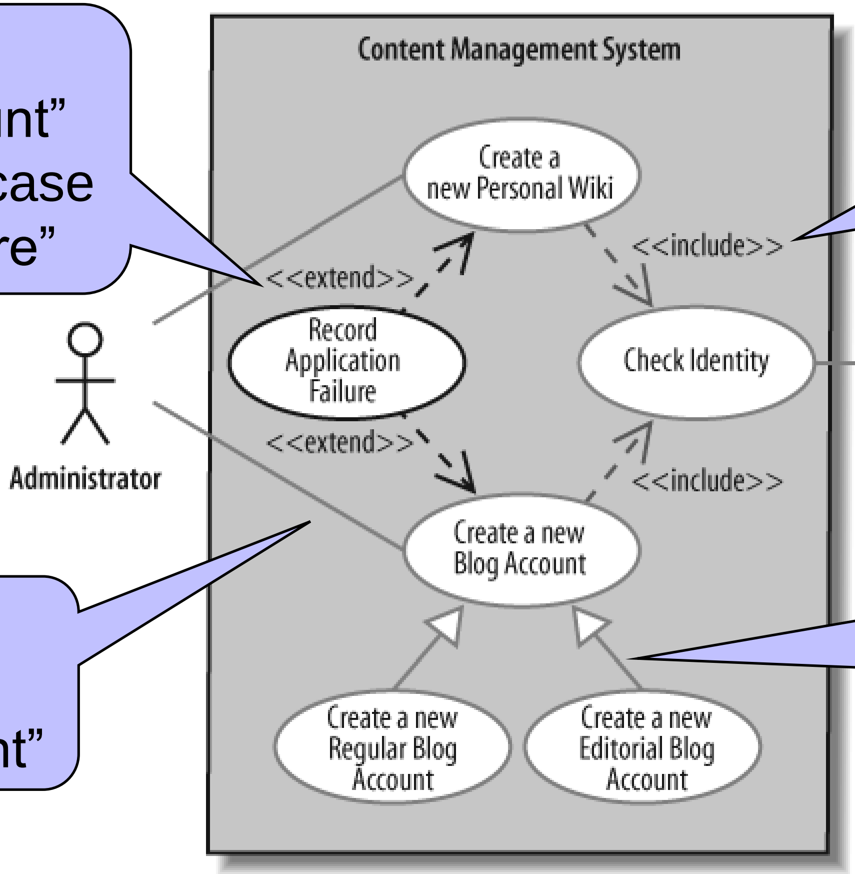


UML Use Case Diagram

- Modeling relationships / dependencies between use cases
  - *NOT "defining" use cases! (frequent misunderstanding)*

Performing use case "Create a new Blog Account" **sometimes involves** use case "Record Application Failure"

Note: "extend" has nothing to do with inheritance here!



Content Management System
Create a new Personal Wiki
<<extend>>
<<include>>
Record Application Failure
Check Identity
<<extend>>
<<include>>
Administrator
Author Credentials Database
Create a new Blog Account
Create a new Regular Blog Account
Create a new Editorial Blog Account

Performing use case "Create a new Personal Wiki" **always includes** use case "Check Identity"

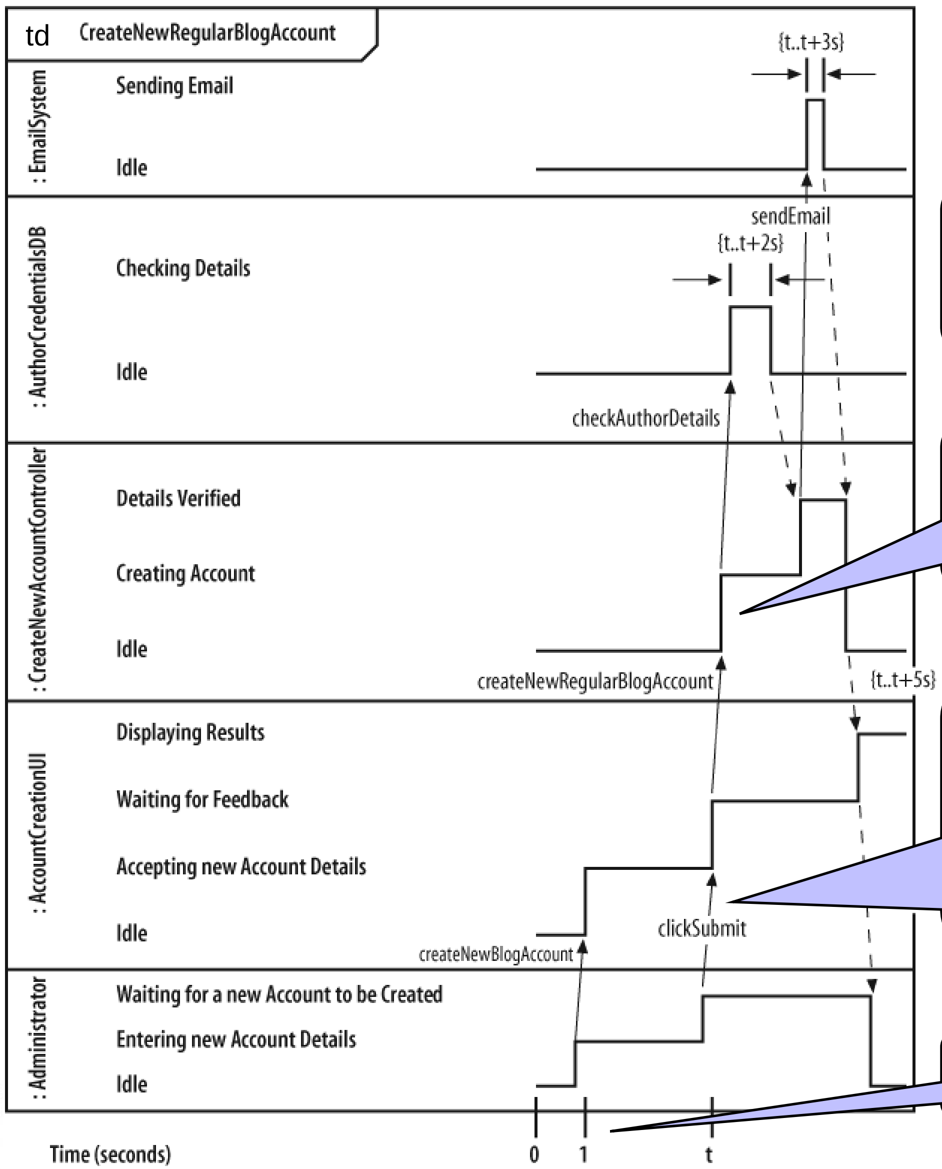
Actor "Administrator" can perform **use case** "Create new Blog Account"

Use case "Create a new Editorial Blog Account" is a **specialization** of "Create a new Blog Account"

Figure: Miles & Hamilton: Learning UML 2.0. O'Reilly 2006

UML Timing Diagram

- Modeling timing of message exchanges and state changes



Two options for visualizing **states**

Allowed **time interval** for state “Checking Details”

State change from “Idle” to “Creating Account”

Message “clickSubmit” sent from **participant** “Administrator” to “AccountCreationUI”

Units of time

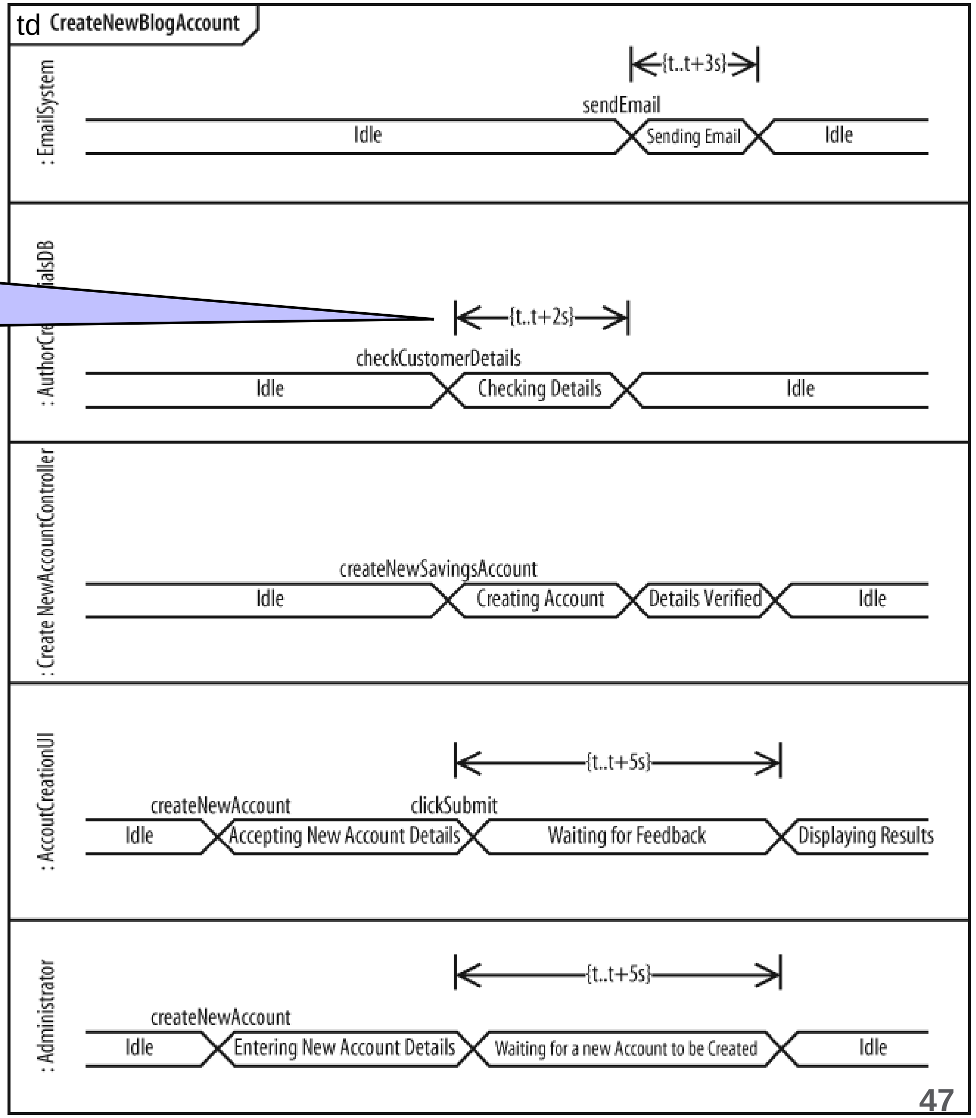
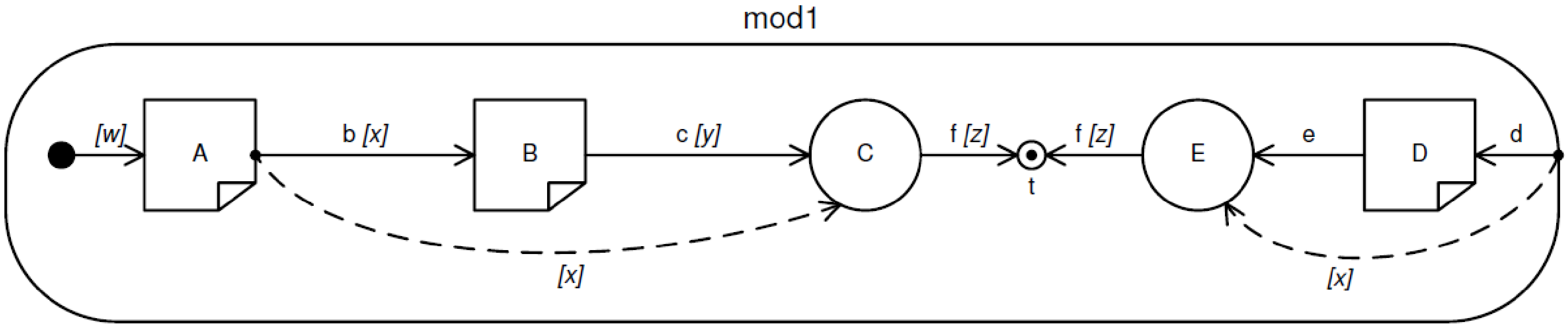


Figure: Miles & Hamilton: Learning UML 2.0. O'Reilly 2006

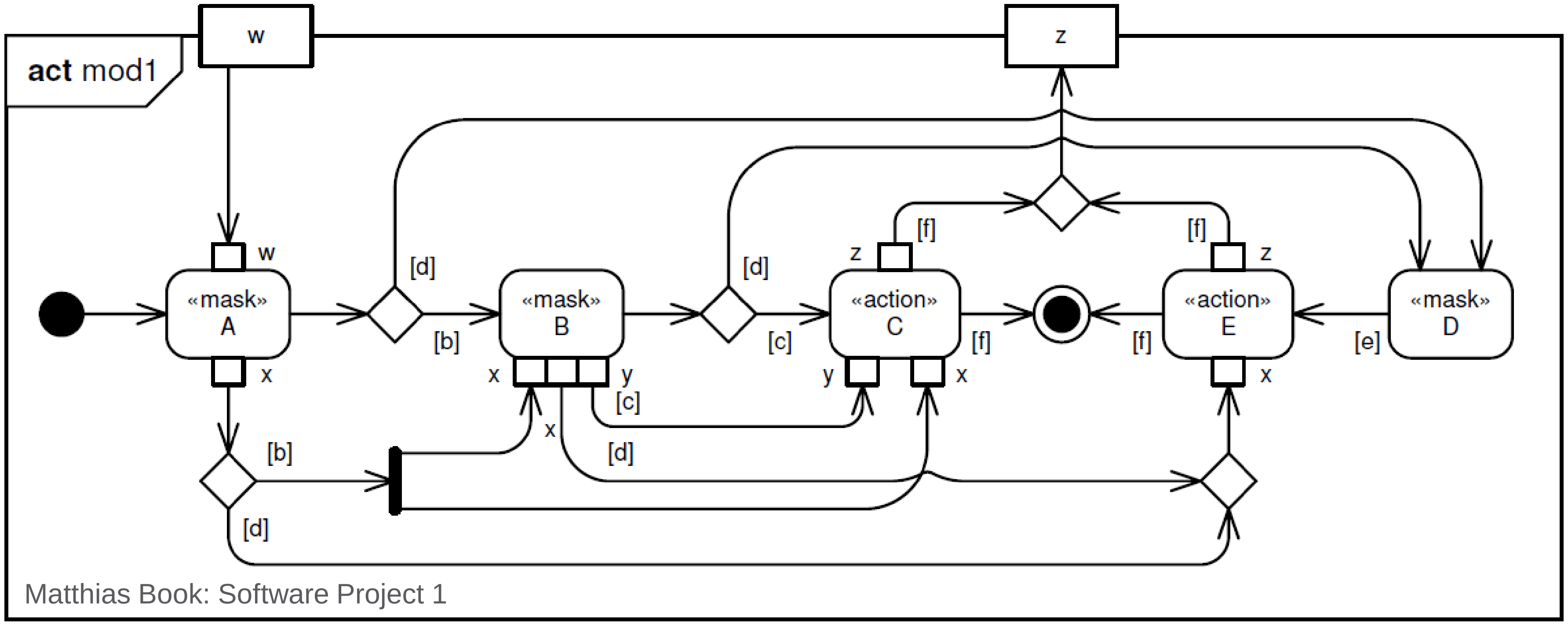
Modeling Pragmatism

- Sometimes, an informal or specialized notation is more efficient than a UML diagram.
- Example: Modeling dialog flows in web applications



Dialog Flow Notation

UML Activity Diagram

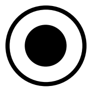


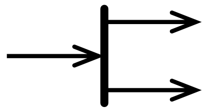
In-Class Quiz #9: UML Diagrams

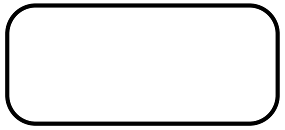
- **Decide which of the following UML diagram types (a-d) are (1) <u>structure</u> and which are (2) <u>behavior/interaction</u> diagrams:**

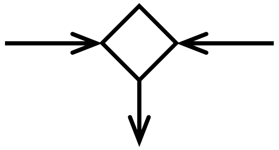
a) UML state machine diagram
b) UML package diagram
c) UML sequence diagram
d) UML object diagram

- **What is the meaning (5-8) of the following UML symbols (e-h)?**

e) 

f) 

g) 

h) 

5. Action
6. Final state
7. Merge node
8. Fork pseudo state