Lecture 4: Models

CS 5150, Spring 2026

Administrative reminders

- Project Pitches are being reviewed (Response: Latest by Jan 30)
  - Approve/Refinement/Reject
  - Team Assignments
- **Next**: Meet with your clients, work on Project Plan
  - Feb 5: In class activity: Preparing requirements, milestones, etc.
- Course staff email: cs-5150-staff@cornell.edu
  - But Ed preffered
- TA office hours: Wed 1-2 PM, 441 Statler Hall

Requirements

… continued from last lecture

Previously …

- Agile models
- XP
- Requirements:
  - Heavyweight v Lightweight
  - Functional vs Non-Functional

Eliciting requirements

Interviews

- Difficult, but essential
- Tips:
  - Allow plenty of time
  - Prepare before meeting client
  - Keep full notes
  - Clarify what you do not understand
    - Define domain-specific terminology
  - Repeat what you hear
- Relevant for all client meetings
- Consider all stakeholders
- Ask questions
  - "Why do you do things this way?"
  - "Is this essential?"
    - Be wary – impact may not be obvious
  - "What are the alternatives?"

Negotiation and Prioritization

- Conflicts, and difficulties affecting cost and schedule, must be resolved with client
  - Help client understand the tradeoffs
  - Be open to suggestions
- Incremental delivery (e.g., Agile sprints) encourages regular prioritization

Stories & scenarios

- Don’t start with formal specifications
  - Most clients can't relate to them
  - Difficult to evaluate completeness
- **Stories** put devs, client on same wavelength
  - Describe actors and their goals
  - High-level, "big picture"
  - Lavish detail about context
    - Helps crystalize alternative viewpoints
    - Refocus by asking which details are relevant
- **Scenarios** detail interactions with system
  - Agile user stories - narrative scenarios with moderate detail
    - Often written on cards
    - Devs break into tasks to estimate effort
    - Prioritized by clients for inclusion in a sprint
    - Postponed stories may be revised with minimal rework
  - Structured scenarios provide more detail
    - Tool for clarifying requirements, checking completeness

Usage scenarios (or Stories)

- Illustrates some interaction with a proposed system
- Use specific examples from a user's point of view
- Clarifies many functional requirements
- Especially good for analyzing off-nominal behavior

- Must include:
  - Purpose
  - User or transaction being followed
  - Assumptions about equipment
  - Steps of scenario
- Should consider (corner cases)
  - What could go wrong
  - Concurrent activities
  - Changes to system state
- Avoid system details that pertain to design

Examples

- "*As a user, I want a password reset page*" – Incomplete
- "*As a locked-out user, I want **a secure way to reset my password**, so that I can **regain access** without contacting support*"
- Security perspective: "*As a **security administrator**, I want **password reset links to expire quickly**, so that **accounts are protected from unauthorized access**.*"
- Edge cases: "*If I upload oversized image (larger than 5MB), when I upload it, then I should see a helpful error message*"
- System state: "*Password resets should be disallowed during system upgrades*"

Developing scenarios with clients

- Choose a viewpoint
- Identify purpose,
  actors, equipment, procedure
- Ask clarifying questions

- Example: online exam system

Online exam system: Viewpoints?

Online exam system scenario: typical student

- Purpose: Describe how a typical student uses the system to take an exam.

- User:

- Equipment:

- Steps:

Requirements Modeling

- Need to bridge requirements and design
  - Leverage abstraction
  - Exploit patterns
  - Identify invariants
  - Improve precision

- UML
- Use cases
- Activity and flow diagrams
- State charts
- …

- Future lecture

Read Ian Sommerville's book: Chapter 4 and 5

Requirements steps

1. Elicitation & Analysis
2. Modeling
3. Specification

- Heavyweight
  - Document formal specification before beginning design
- Lightweight
  - Relevant requirements developed during sprints
    - But work out system-level requirements upfront
  - Avoid specification unless necessary
    - Models, prototypes clearer to client
    - Sometimes details are important

Models

Lecture goals: Modeling

- Select appropriate models to improve communication during multiple process steps (requirements, architecture, program design)
- Visualize models using UML (Unified Modeling Language)

Purpose of models

- Simplification of reality
- Facilitates communication during process steps
  - Requirements
  - Architecture (system design)
  - Program design
- Need multiple models
  - Different perspectives
  - Different levels of completeness, formality
- Larger, more complex projects benefit from more formality
- Most models are consumed by *humans*

Representing models

- UML: Unified Modeling Language
  - Models consist of diagrams and specifications
  - Many different diagram types
  - Particularly well suited to object-oriented design
- Can serve many purposes
  - Facilitate discussion
  - Provide documentation
  - Generate code

- Why not code?
  - Can have multiple models with simplifications serving different perspectives
  - Code usually must pick a single abstraction; can't manifestly show correctness for other perspectives
  - Code can introduce syntactic distractions, platform details
  - Sometimes, (pseudo)code is the clearest specification

Modeling perspectives

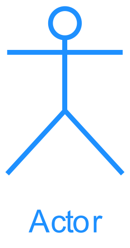
- **External**
  - Represent the (simplified) context of the system wrt environment
- **Interaction**
  - How do user and component interactions proceed?
  - E.g., Use Cases, Sequence Diagrams
- **Structural**
  - How are system components organized?
  - How is data represented? E.g., Class Diagrams
- **Behavioral**
  - How system responds to events, changes over time
  - E.g., Data flow Diagram, State/Transtion Diagrams

Interaction models

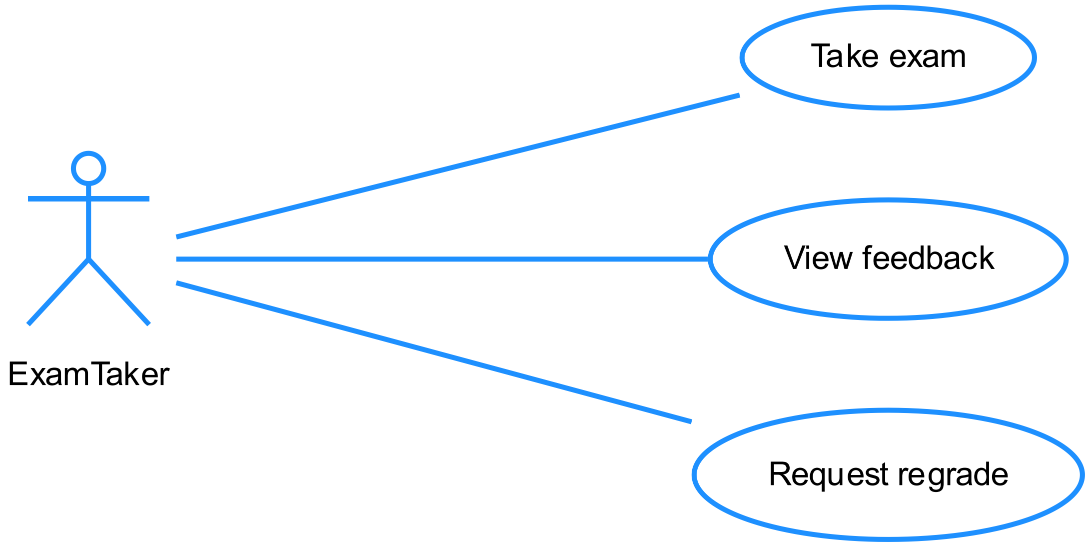
- Modeling user interactions helps catalog functional requirements
  - Use case diagrams
- Modeling inter-system interaction helps highlight potential communication problems
  - Sequence diagrams

Use cases

- Discrete task involving external interaction with the system
- Actor
  - A role, not an individual
  - Beneficiary or instigator
  - May be other systems
  - Use specific, not generic names



- Use case



Pair with textual description

- Metadata
  - Name of use case
  - Goal of use case
  - Actor(s)
  - Trigger
  - Preconditions
  - Postconditions
- Flow of events
  - Basic flow
  - Alternate flows
  - Exceptions

- Name: Take exam
- Goal: Enables a student to take an exam online with a web browser
- Actor(s): ExamTaker
- Trigger: ExamTaker is notified that the exam is ready to be taken
- Preconditions: ExamTaker is registered for course; ExamTaker has authentication credentials
- Postconditions: Completed exam is ready to be graded

Basic flow ("Take exam" use case)

1. ExamTaker connects to server via web browser
2. Server checks whether ExamTaker is already authenticated; if not, triggers authentication process
3. ExamTaker **selects** an exam from list
4. ExamTaker repeatedly selects a question and either types in a new solution, edits an existing solution, or uploads a file with a solution
5. ExamTaker either **submits** exam or **saves** current state
6. When exam is **submitted**, server checks that all questions have been attempted and sends acknowledgement to ExamTaker

Discuss

- What could be some alternate or erroneous scenarios for the “Take Exam” use case?

Alternative flows

Alternate flow

- Alternative path to successful completion of use case
- Example: Take exam
  - Resuming exam from saved state
  - Solution file format not accepted
  - Submission is incomplete

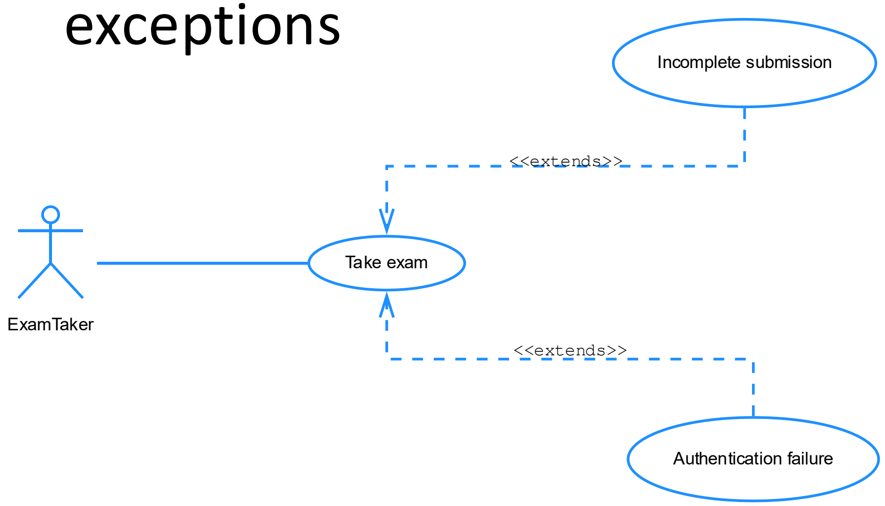
Exceptions

- Lead to failure of use case
- Example: Take exam
  - Authentication failure

Relationships

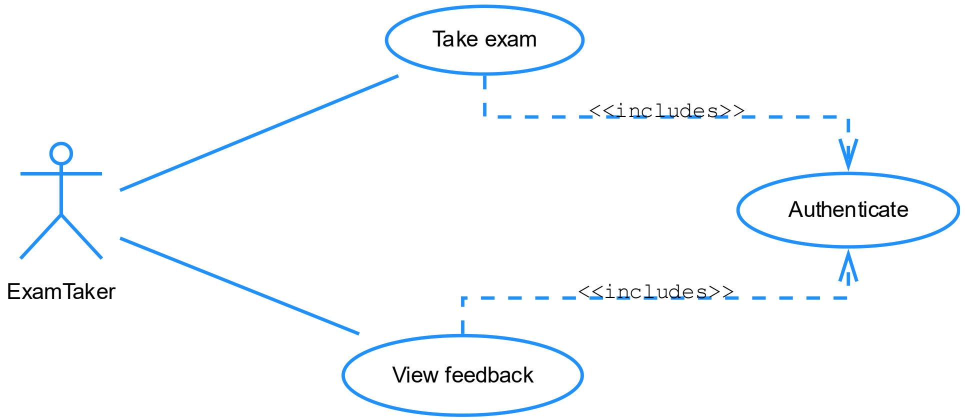
<<extends>>

- Defer extra detail to other use cases
  - Useful for alternate flows and exceptions



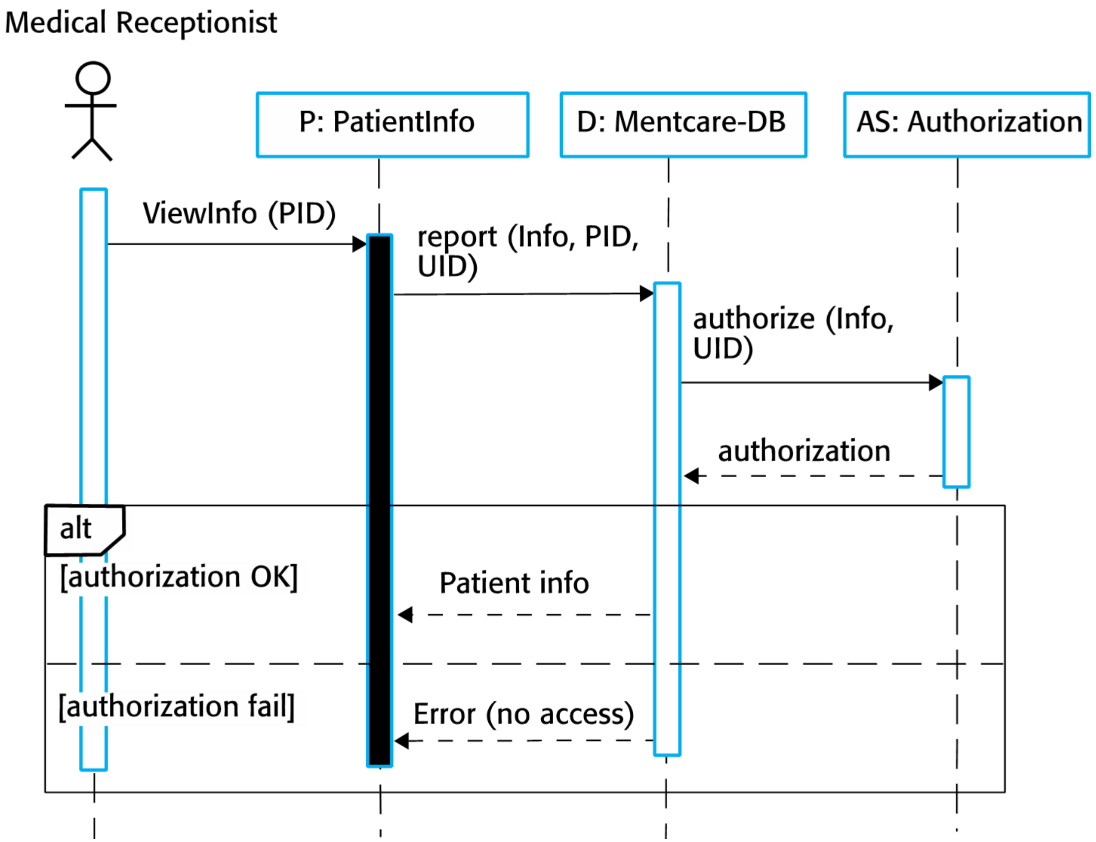
<<includes>>

- Include steps from another use case
  - Useful when common procedure is required in multiple contexts



Sequence Diagrams

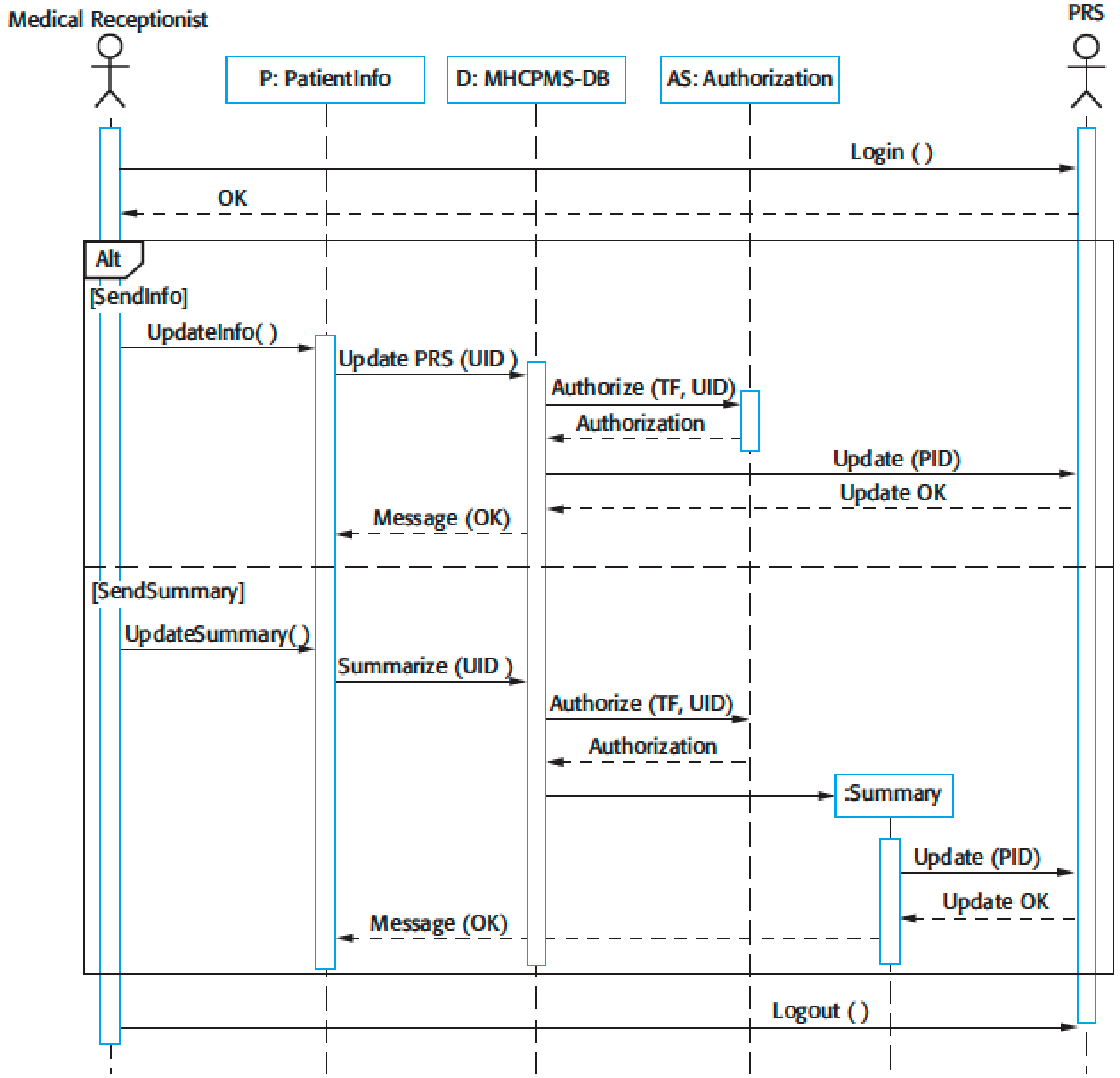
- Show sequence of interactions (ordering, causal relationships) between actors and objects
  - Excellent for documenting communication protocols
- Networking examples: https://www.eventhelix.com/networking



Sommerville, *Software Engineering*

Sequence Diagrams

- A more complex example
- Can be used for code generation



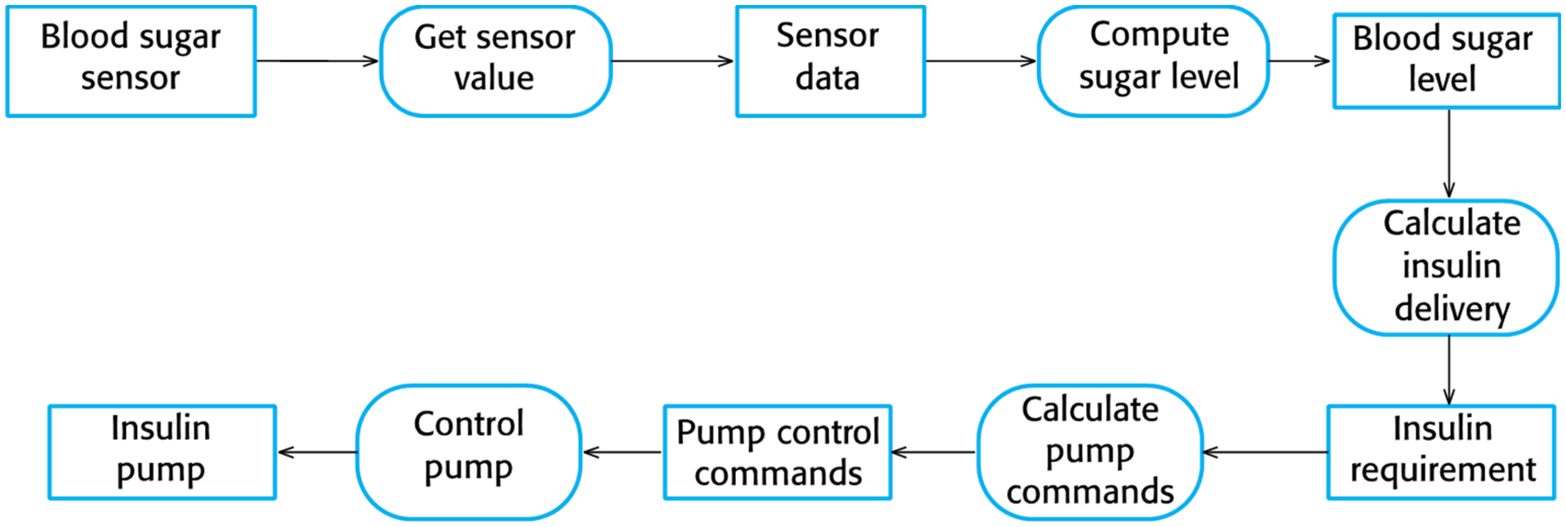
Behavioral Models

- Model dynamic behavior of system during execution
- How does system process data or respond to events?

- Data-driven models
  - Show sequence of processing steps from input to output
- Event-driven models
  - How does system respond to events? (internal and external)
  - Assumes finite number of application states
  - Great for embedded, real-time systems

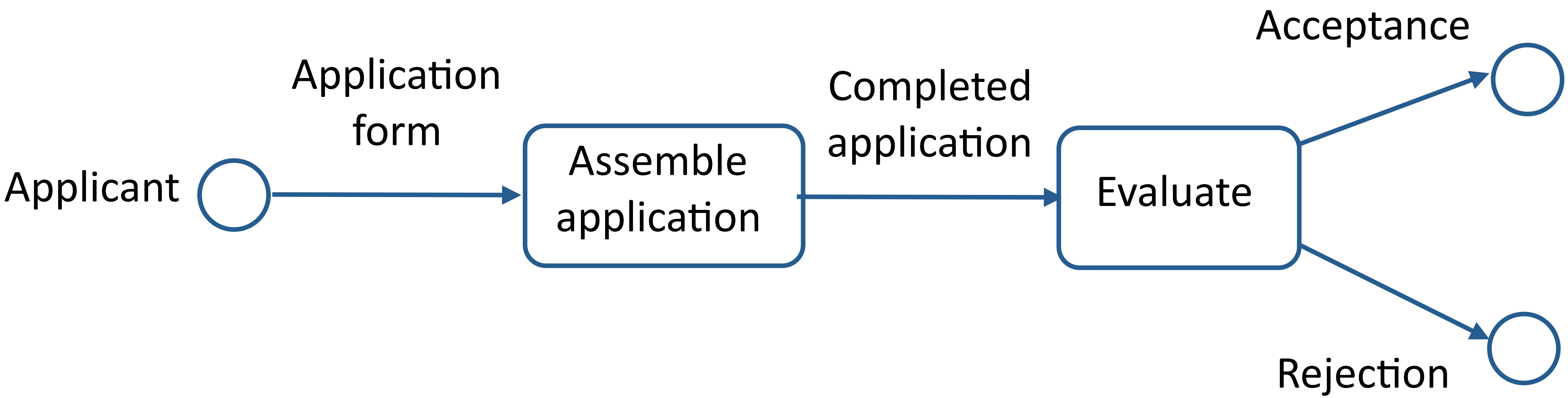
Data flow (activity) diagrams

- **Example Task**: Chain of Processing in insulin pump software
- **Activity**: rounded rectangle
- **Data**: rectangle or labeled edge
- **Data source/sink**: rectangle
- **Beginning/end**: circle

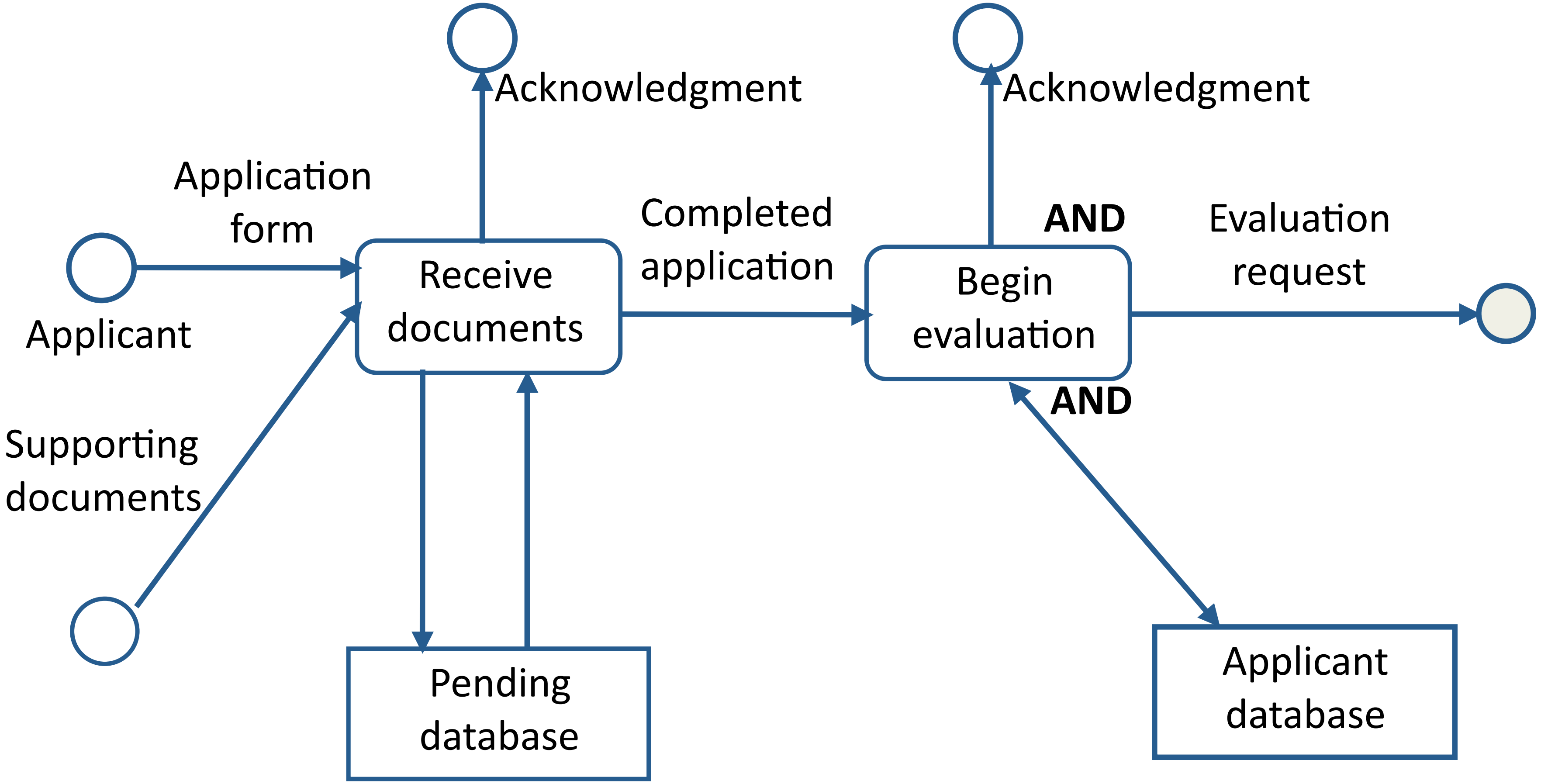


Sommerville, *Software Engineering*

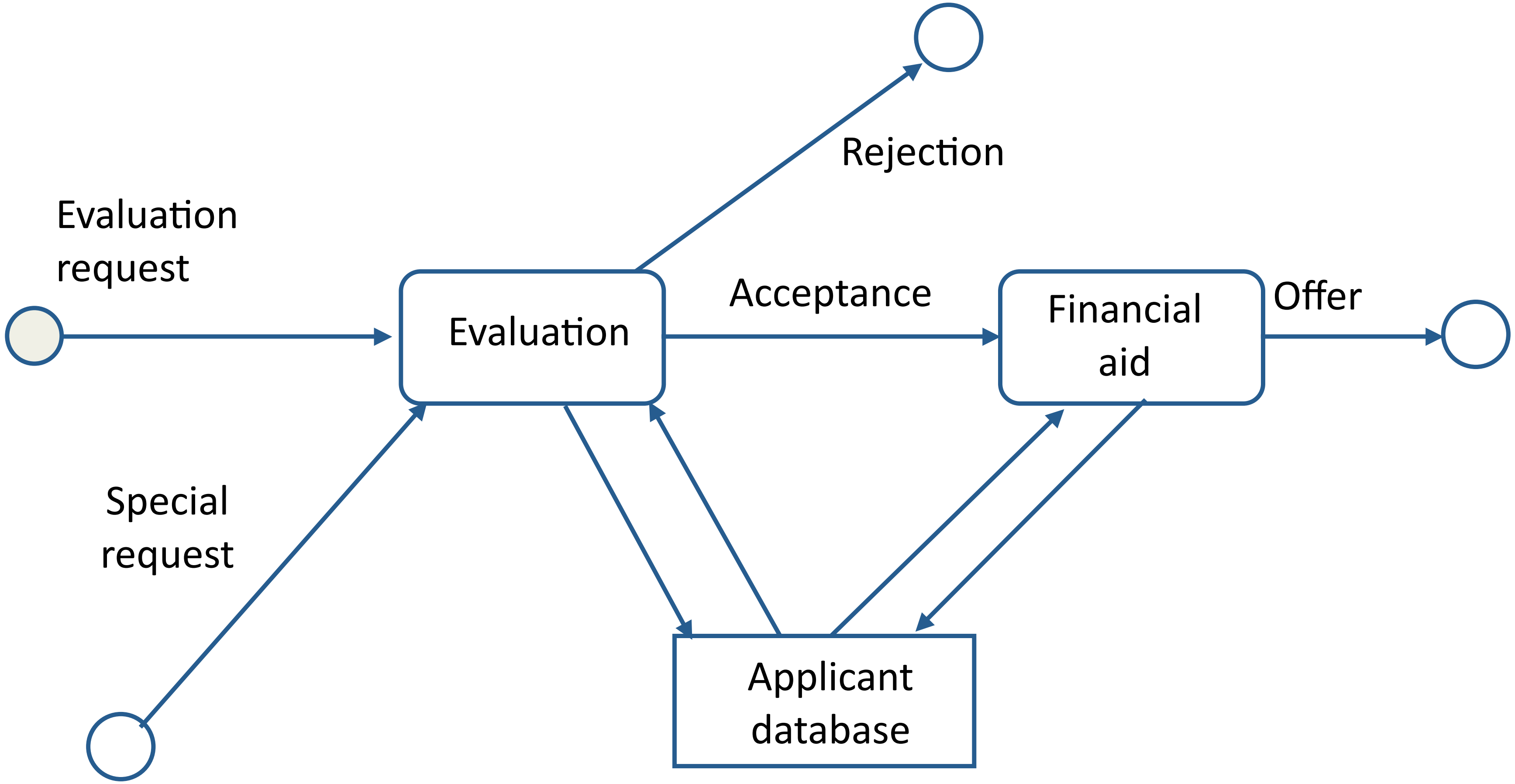
Example: University Admissions



Refined example



Refined example, continued



How to specify logic?

- Data flow & sequence diagrams show high-level flow; must be augmented by specifications for low-level behavior

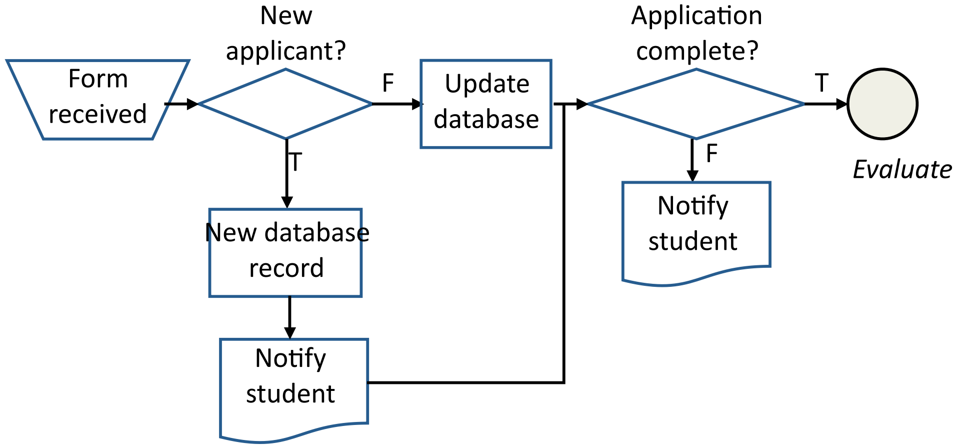
- Decision table
  - Process columns from left to right
  - Rules are specific and testable
  - Can be clearer to clients than code

SAT > S1	T	F	F	F	F	F
GPA > G1	-	T	F	F	F	F
SAT between S1 and S2	-	-	T	T	F	F
GPA between G1 and G2	-	-	T	F	T	F
Accept	*X*	*X*	*X*			
Reject				*X*	*X*	*X*

Flowcharts and pseudocode

Flowchart

- Shows logic (not just flow)
- Used to specify computer programs before modern programming languages



Pseudocode

- Compact and precise
- Composable
- Easy to implement
- Harder to see flow

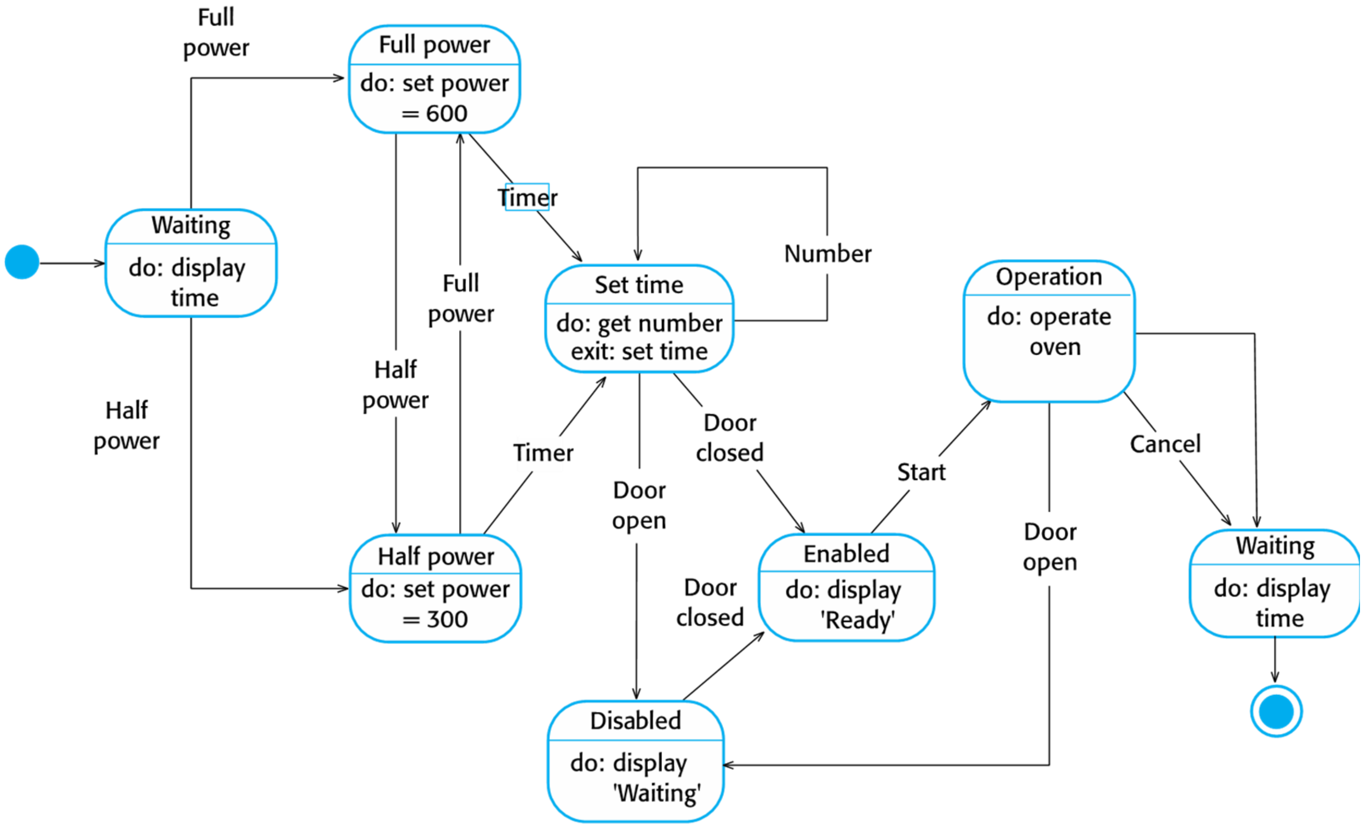
admin_decision (application)
    if application.SAT == null **then** error (incomplete)
    if application.SAT > S1 **then** accept(application)
    else if application.GPA > G1 **then** accept(application)
    else if application.SAT > S2 **and** application.GPA > G2
        then accept(application)
    else reject(application)

Mathematics

- Many systems are well-described by mathematical models
  - Differential equations
  - Probability distributions
  - Integrals
  - Filters
  - Interpolation
  - Curve fits
- Document progression of approximations and domain transformations
  - Frequency vs. time domain
  - Continuous vs. discrete
    - Differential vs. difference equations
    - Integration vs. quadrature
    - Root solve vs. Iteration
- Higher-level specifications give developers more flexibility, can improve maintainability

State charts / Transition diagrams (Event Driven Modeling)

- Model system as a finite set of states
- A transition moves the system from one state to another
  - Triggered by a condition
  - Mathematically, a function from $S \times C \rightarrow S$
- Can be hierarchical
- Also useful for user interface navigation



Sommerville, *Software Engineering*

Transition tables

- Specify state transitions in textual form
  - Useful when transitions are "dense" (most conditions are applicable in most states)
  - Example: physical buttons on embedded device
- Can visually check for completeness

State	Next State				
Action>	**Half Power**	**Full Power**	**Timer**	**Door Open**	**Door Close**
Waiting	Half Power	Full Power			
Full Power	Half Power		Set Time		
Half Power			Set Time		
…					