# UML

- Graphical notations
- Meta-model
- UML as sketch, blueprint, programming language
- MDA – PIM, PSM
- History – three amigos
- Tools – Visio, Dia, Violet

# Class Diagrams

- Classes, Interfaces
- Attributes, operations
  - Visibility (+,-,#,~), multiplicity
- Associations: uni, bi-directional
- Dependencies
- Generalization, realization
- Constraints {} (DbC)

# More on Attributes

- Attribute Syntax:
  - [visibility] name [multiplicity] [: type]
    [= initial-value] [{property-string}]
  - Examples

    password

    - password

    password : String

    # password : String = “changeme”

    + password [1] : String

    password : String {frozen}

# Visibility

- visibility:
  + for public, # for protected, - for private
- Public: Anything that can access the class can access this attribute or operation
- Protected: Any descendant of this class can access this attribute or operation
- Private: Only operations in this class can access this attribute or operation

# More on Attributes

- property-string options:
  - changeable (default if left off)
    - No restrictions on modifying the attribute's value.
  - addOnly
    - If multiplicity is > 1, additional values may be added, but once created, a value may not be removed or altered.
  - frozen
    - Attribute's value cannot be changed after initialization.

# More on Operations

- Operation Syntax:
  - [visibility] name [(parameter-list)]
    [: return type] [{property-string}]
  - Examples

    connect

    + connect : Boolean

    connect(name : String, password : String)

    isConnected() : Boolean {isQuery}

    getName (in ID : Integer, out name : String)

# More on Operations

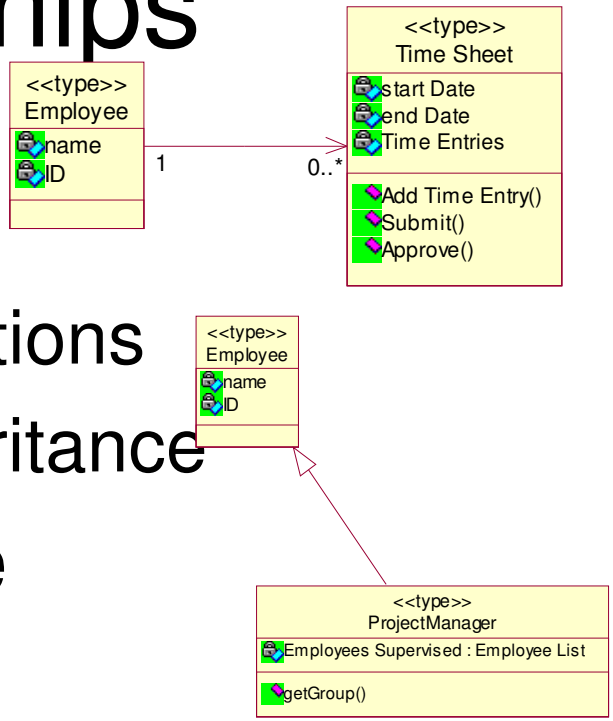
- Operation Parameter Syntax:
  - [direction] name : type [= default-value]
  - direction: in, out, inout
  - Example:
    - # getPasswd(in ID : Integer, out passwd : String = “changeme”)

# More on Operations

- property-string options:
  - leaf
    - non-polymorphic; may not be overridden.
  - isQuery
    - causes no changes or side-effects to the system.
  - sequential (valid only with active classes)
  - guarded (valid only with active classes)
  - concurrent (valid only with active classes)

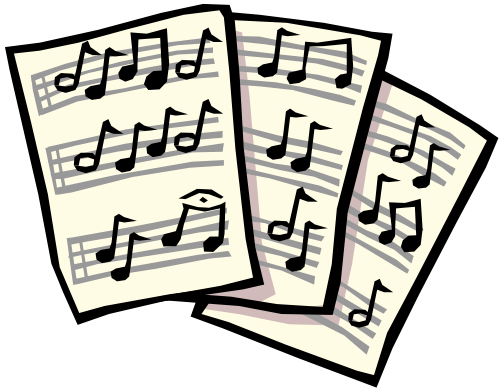
# Class Relationships

- We have seen:
  - uni- and bi-directional associations
  - generalizations/subtypes/inheritance
- Now we will look at two more forms of associations:
  - aggregation
  - composition



# Aggregation

- whole-part relationship
- the “has-a” relationship
  - Examples:
    - A course has students.
    - A song has notes.



# Aggregation

- The whole (aggregate) contains parts, but the parts may be in multiple aggregate classes
  - Examples:
    - A course has students.
      - Students may be enrolled in several courses simultaneously.
    - A song has notes.
      - Several songs may use the same notes.

# Aggregation

- The whole (aggregate) contains parts, but destroying the whole does not destroy the parts and removing the parts does not have to destroy the whole
  - Examples:
    - A course has students.
      - Deleting a course from the schedule does not delete the students.
    - A song has notes.
      - A song may have no notes. (John Cage 4’33’’)

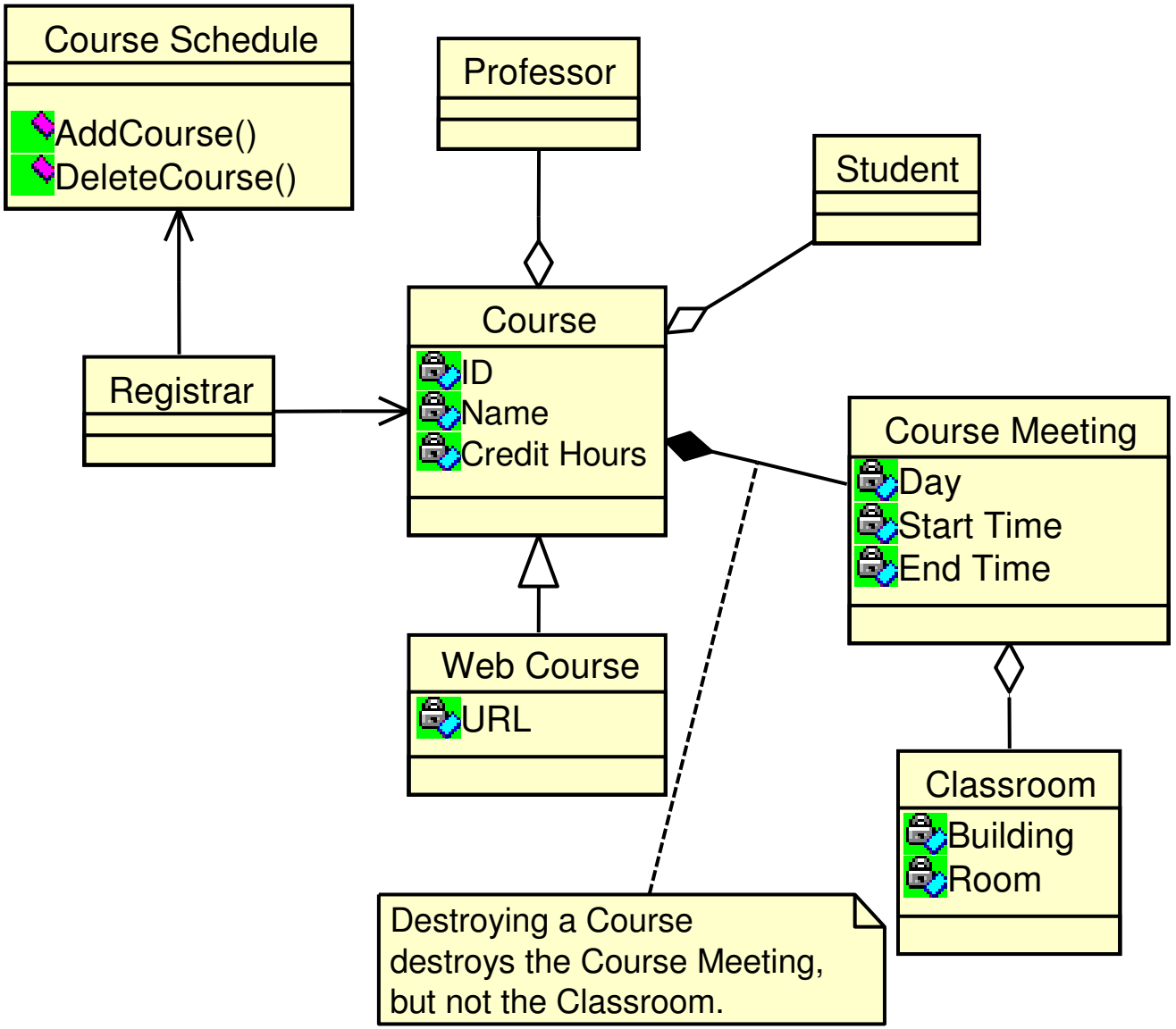
# Aggregation

- What is the difference between an association and an aggregation?
  - Officially there is no significant difference.
  - An association may simply mean that one class knows about another class.
  - An aggregation implies that one class is made up of other objects.

# Composition

- A stronger form of aggregation where
  - the parts have the same lifetime as the whole.
    - Or at least they die at the same time.
  - a part can exist in only one whole.
  - Examples:
    - An OS process contains allocated memory.
    - A document contains a signature.

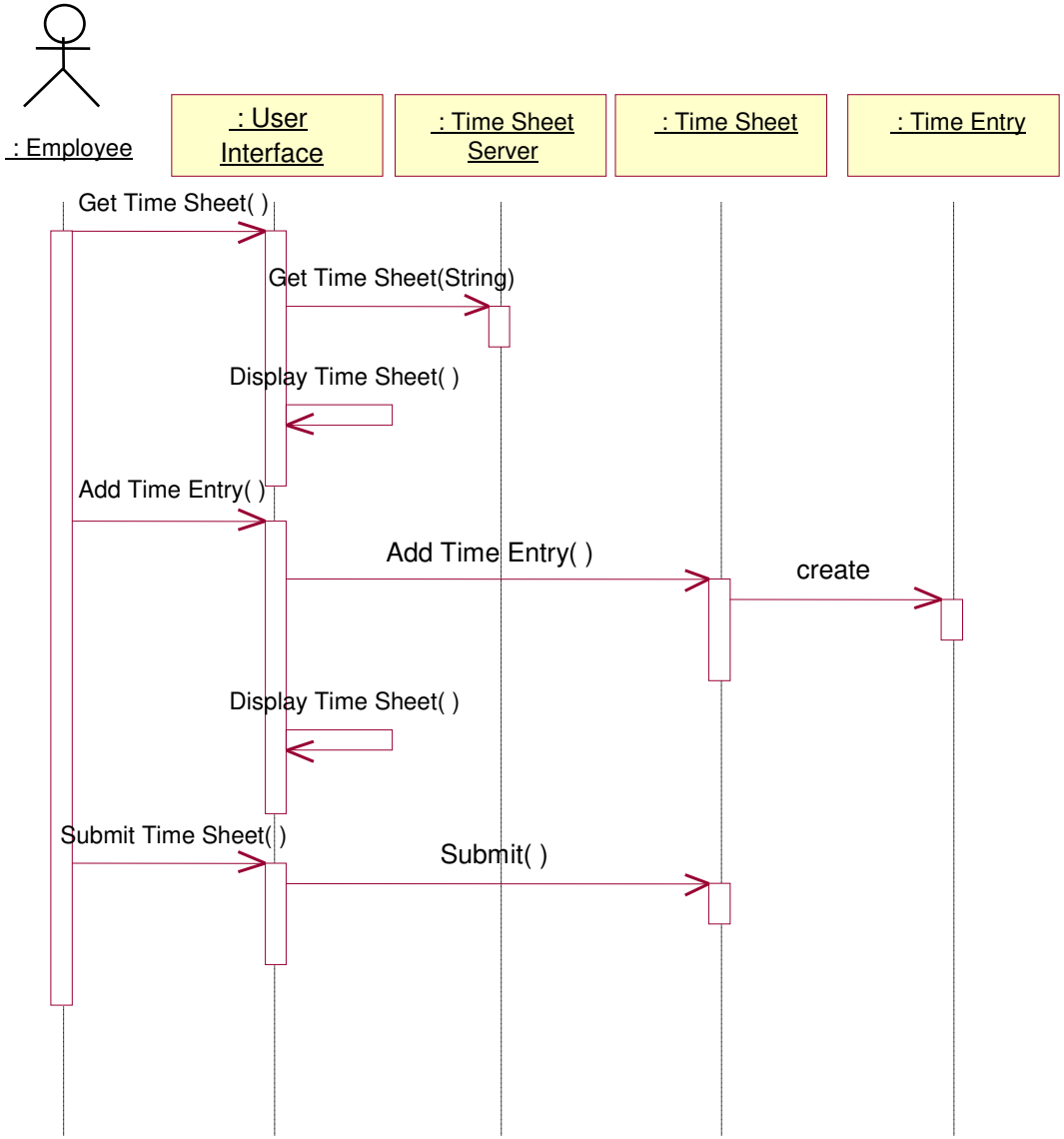
# Example


Course Schedule
AddCourse()
DeleteCourse()
Professor
Student
Course
ID
Name
Credit Hours
Registrar
Course Meeting
Day
Start Time
End Time
Web Course
URL
Classroom
Building
Room
Destroying a Course
destroys the Course Meeting,
but not the Classroom.

# Class Relationships

- Assume classes A and B are related as follows:
  - Subtype (Generalization/Inheritance):
    - A is a kind of B
  - Instance (Classification):
    - A is an example of B
  - Association:
    - A knows about B
  - Aggregation:
    - A has a B
  - Composition:
    - A contains a B

# Sequence Diagrams



# Sequence Diagrams

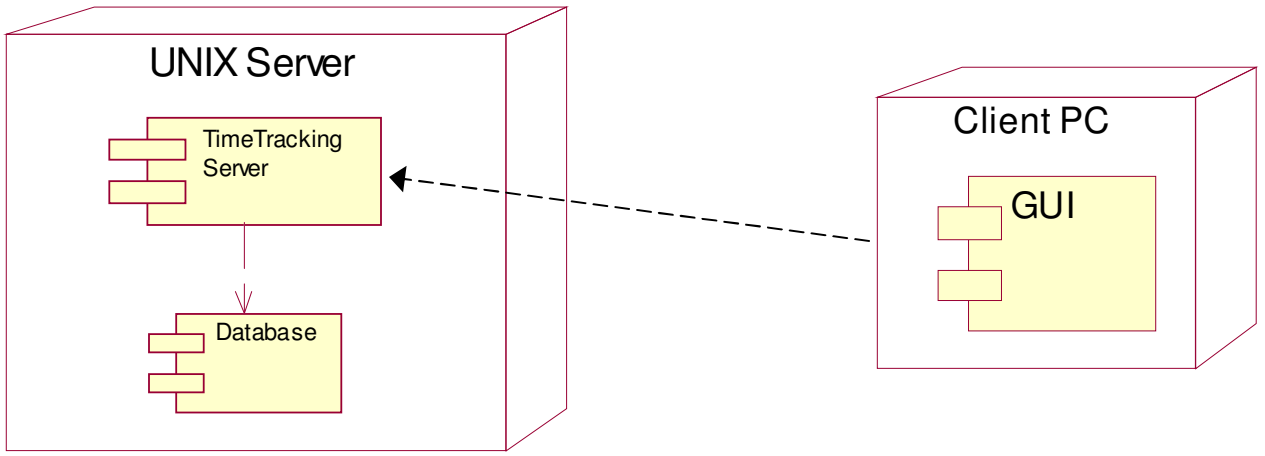
- Vertical line is lifeline of the object
- Objects can be created
- Objects can invoke operations on themselves
- Conditions may be added
  - Ex. [all Time Entries entered]
- Iterations can be indicated with *
- Return arrows are implicit or explicit

# Sequence Diagrams

- Objects can be deleted with an X
- Asynchronous messages can be created for use with multi-threading/processing.
  - Half arrows indicate the method is invoked and control is returned to the caller (no blocking)

# Deployment View

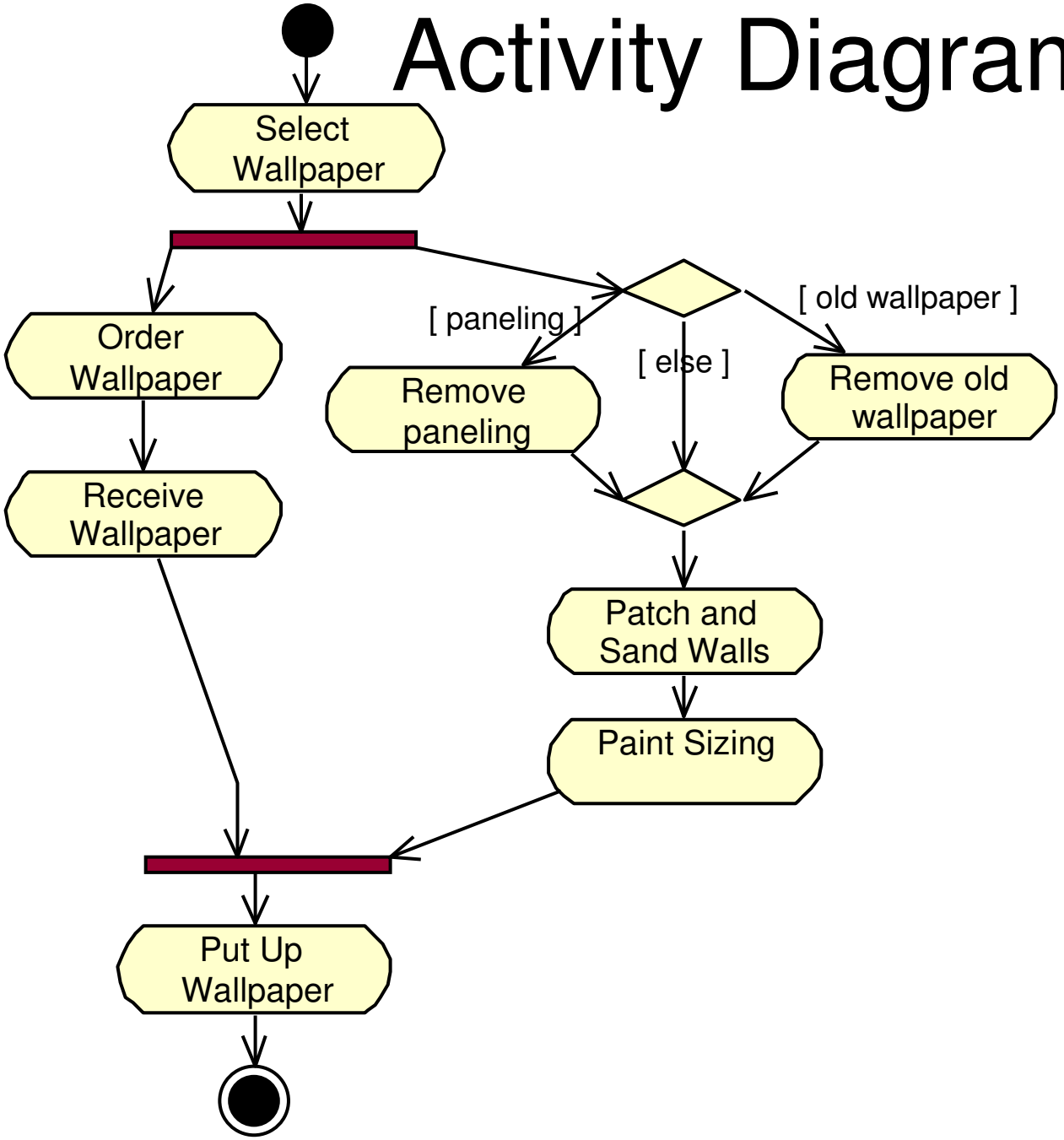
- Describes physical network configurations
- Concerns the performance, throughput, fault-tolerance, availability, installation, and maintenance
- Uses Deployment Diagrams



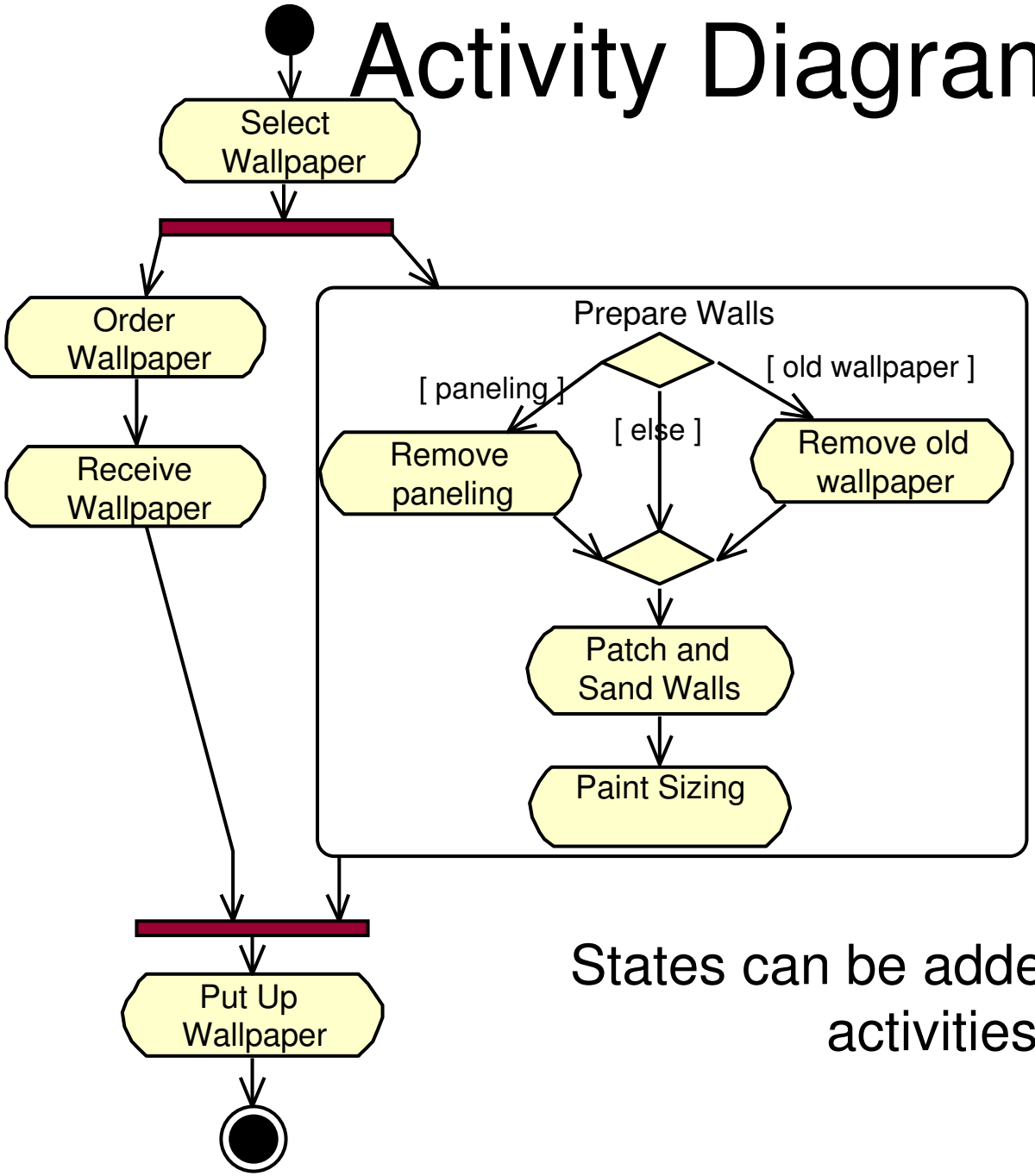
# Activity Diagrams

- Displays sequential behavior of a system
- supports conditional behavior
  - branch and merge
- supports parallel behavior
  - fork and join
- similar to state diagrams where states are activities

# Activity Diagrams



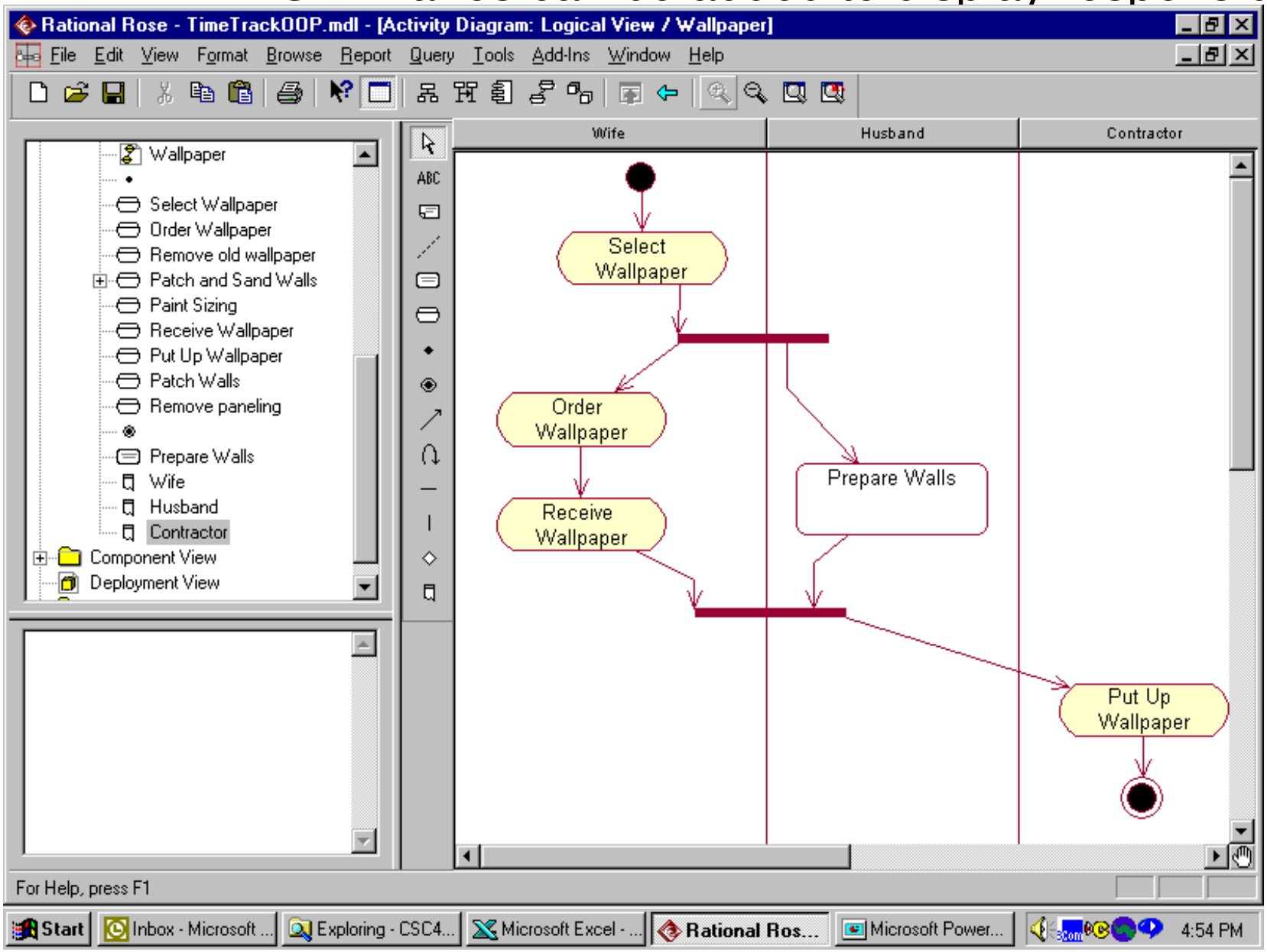
# Activity Diagrams




States can be added to group activities

# Activity Diagrams

Swimlanes can be added to display responsibilities



# Activity Diagrams

- Useful when
  - analyzing a use case
  - understanding workflow
  - flowcharting a complicated algorithm
  - describing tasks in a multithreaded application