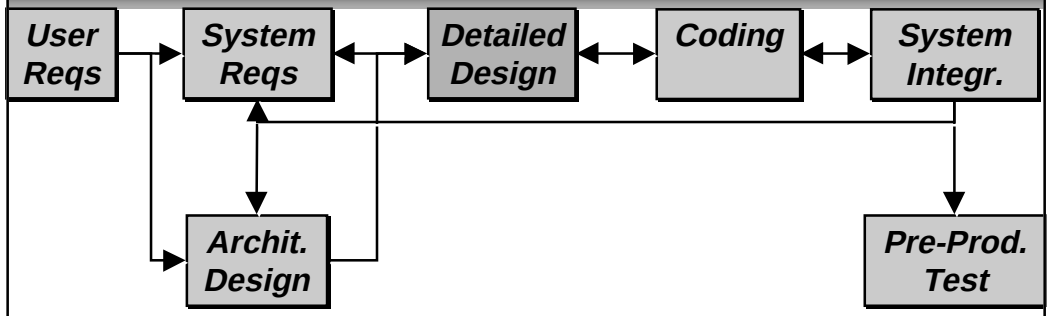
Communications Software Engineering Design Model

Wolfgang Emmerich

Lecture Overview

- *Relationship between analysis and design*
- *Stages of design*
- *Impact of implementation environment*
- *Definition of sequence diagrams*
- *Sequence diagrams for use cases*
- *Creation of class interfaces*
- *State diagrams*

Stages of Development Process



Project and Risk Management

System Management

Version and Configuration Management

Quality Management

Producing a Design Model

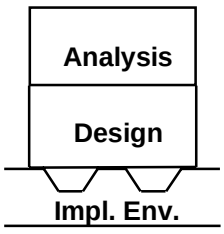
- *Inputs*
- *Actions*
  - *16 Identify implementation environment*
  - *17 Model initial design class diagram*
  - *18 Design control flow*
  - *19 Define class interfaces*
  - *20 Model classes state diagram*
  - *21 Finalise design class diagram*
- *Outputs*
- *Notations*

Design Model Contents

- *Inputs:*
  - *Requirements specifications relating to implementation environment*
  - *Analysis model class diagram*
  - *Use case descriptions*
- *Notations introduced:*
  - *sequence diagram*
  - *state diagram*
- *Outputs:*
  - *sequence diagrams [diagram x use case]*
  - *state charts [diagram x class]*
  - *complete design model class diagram*

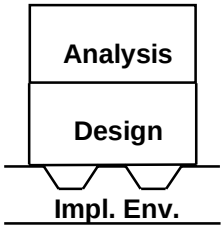
Implementation Environment Factors

- *Target operating system*
- *Programming language*
- *Deployed UIMS*
- *Available system integration mechanisms*
- *Underlying DBMS*
- *Available reuse libraries*
- *Non-functional requirements*
- *Development process*

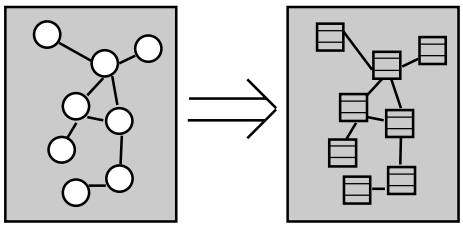


Changes for Environment

- *Add, delete, or change classes (OOSE blocks)*
- *Change associations , e.g.*
  - *extensions to stimuli*
  - *inheritance to delegation*



Analysis & Design Models in UML



Analysis Model:

logical, conceptual, frozen

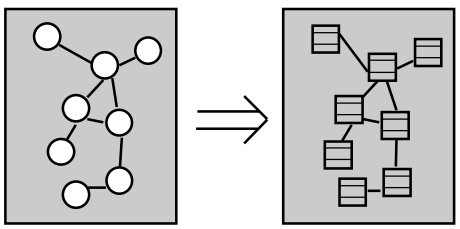
Design Model:

a practical abstraction

Objectives of Class Diagram Design

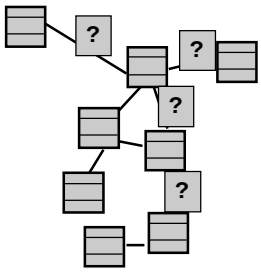
- ***Encapsulation***
  - ***at architectural level***
  - ***to provide cohesive packages***
- ***Normalisation***
  - ***of class structure***
  - ***to provide implementable interfaces with low coupling***

Initial Stages of Design



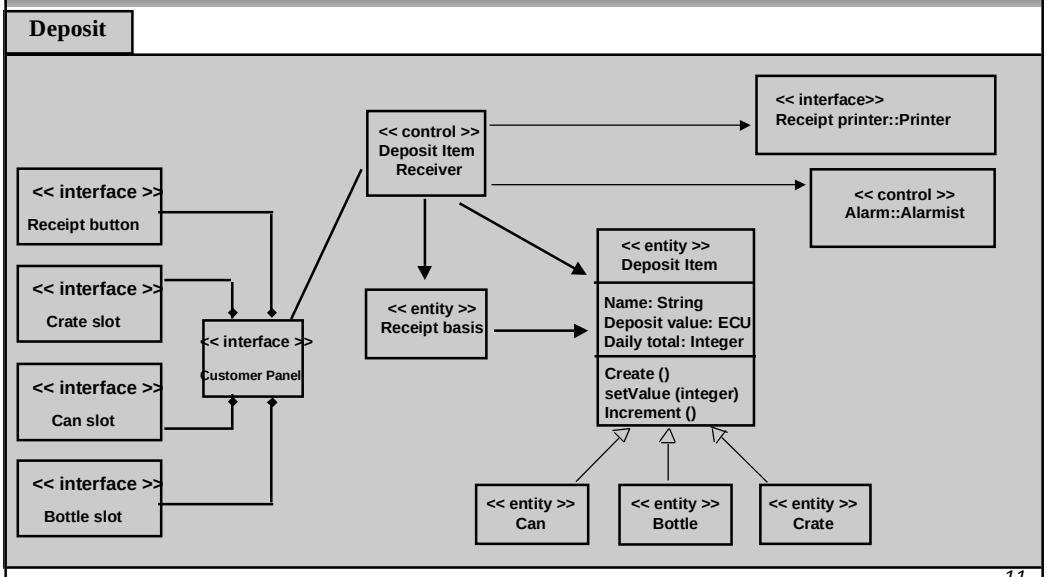
Translate from analysis model

Adapt to environment and revise



Design of control flow

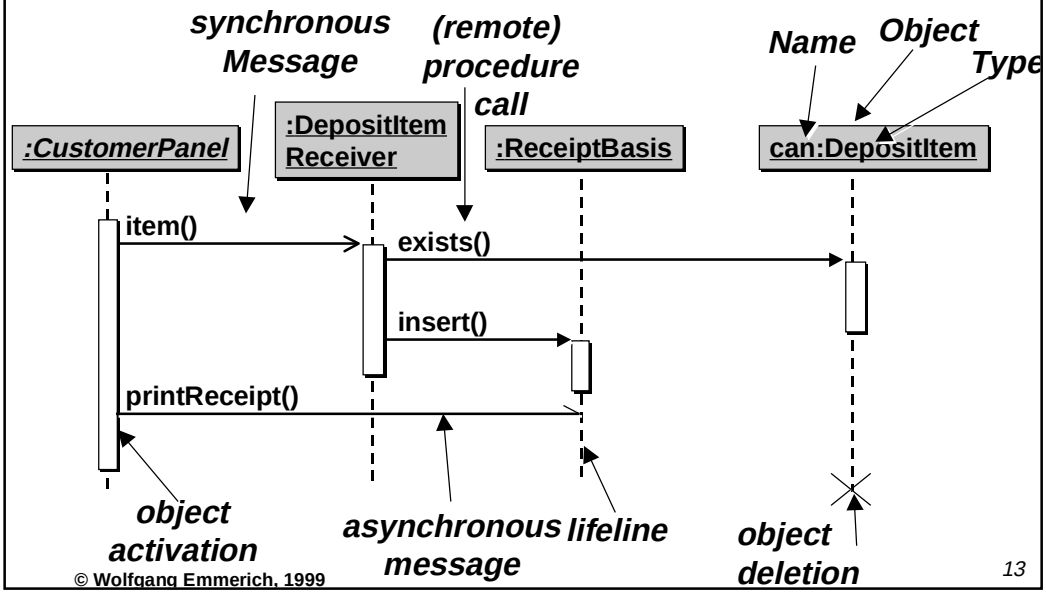
‘Deposit’ Package in Design Model



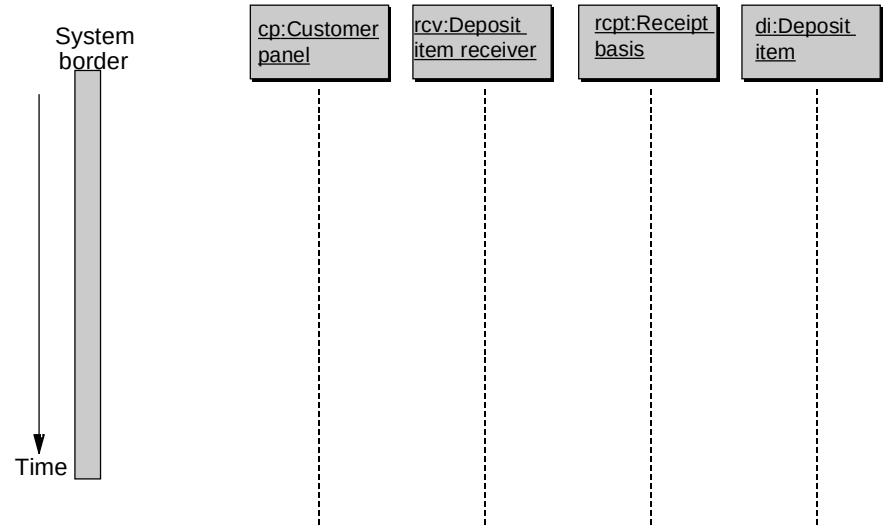
Sequence Diagrams in UML

- *Shows interactions among a set of objects in temporal order*
- *Objects appear as vertical lines*
- *Events marked by labelled horizontal (or slopped) lines*

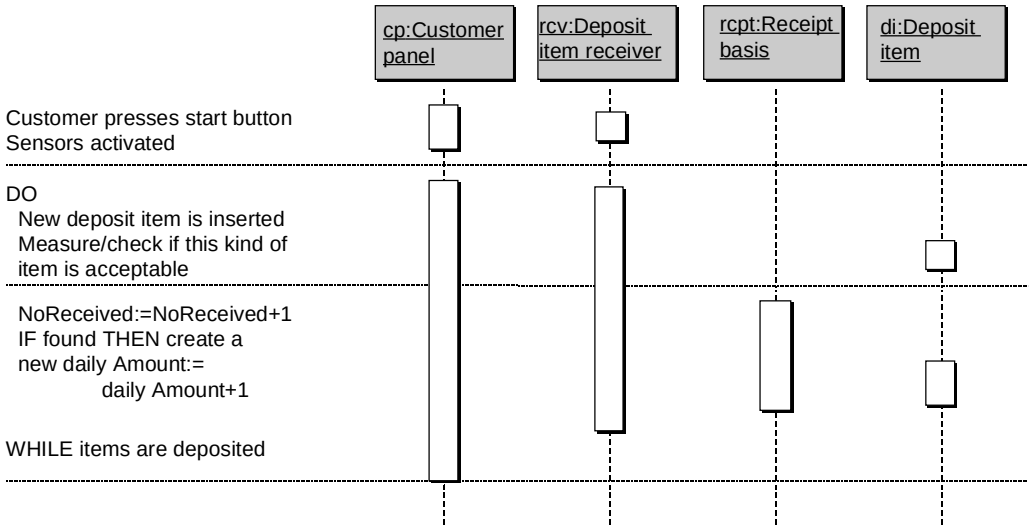
Sequence Diagram Notation



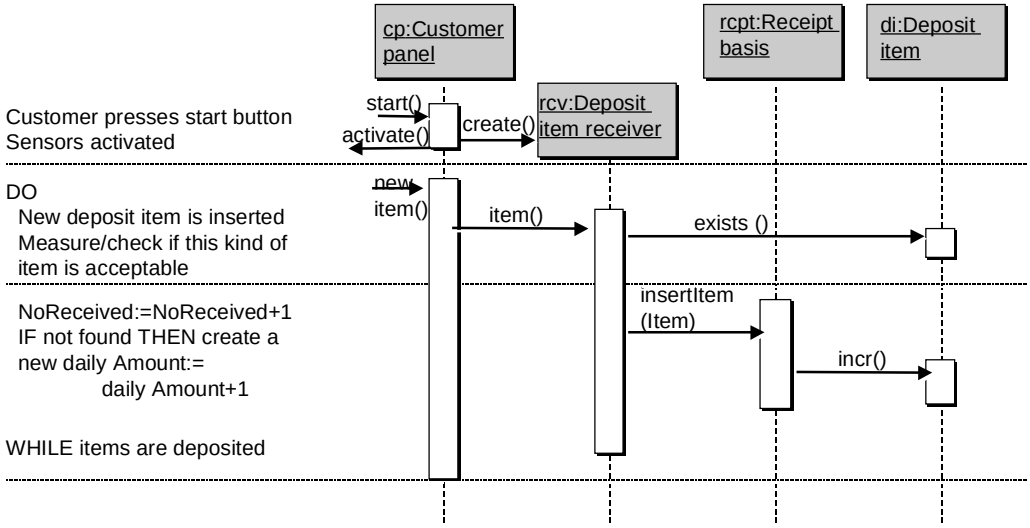
Returning Item (Skeleton)



Returning Item (Skeleton + Operations)



Returning Item (+ Stimuli)

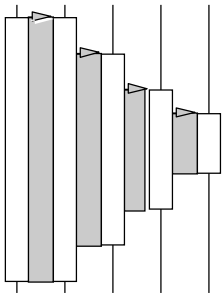


Defining Stimuli

- ***Issues to consider :***
  - ***name plus minimum number of parameters***
  - ***same name for similar behaviour***
  - ***creation also by stimuli***
  - ***basic case designed first***
  - ***two types :***
    - ***messages inside one process***
    - ***signals between processes***

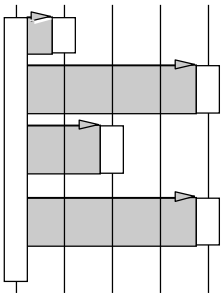
Sequence Diagram Structures

"Stair" decentralised



for well-structured sequence of operations

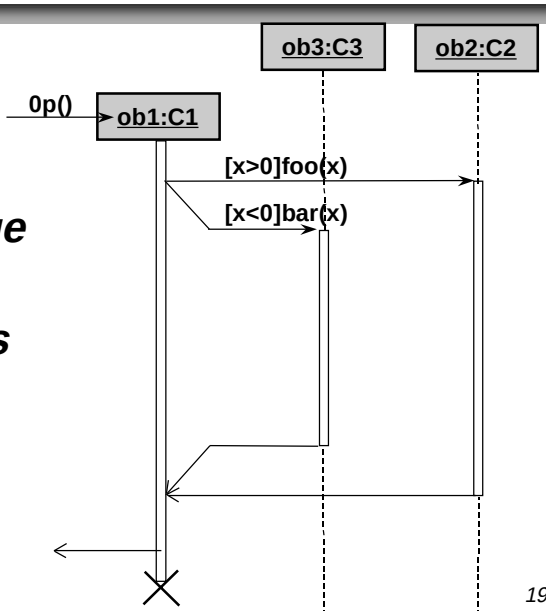
"Fork " centralised



for variable sequence of operations

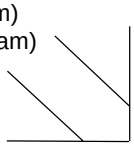
Sequence Diagrams & Conditions

- *Shows general interaction pattern*
- *Conditional shown by splitting message arrow (and return)*
- *Pre-existing objects as broken lines*



Detailing the Design

Receipt Basis
insertItem(item)
printOn(oStream)
delete



ReceiptBasis
itemList: list (ReturnedItem) sum: ECU

create;
insertItem(DepositItem);
printOn(oStream);
delete;

Design elements already considered

- **Design outputs so far :**
  - set of sequence diagrams [diagram x use case]
  - elaborated class diagram
- **Next steps change perspective**
  - from class outside to inside
  - from instance to type-level of abstraction
- **Need for further understanding of each class**
- **Development of state diagrams to define class behaviour**
- **But this can be quite complex**

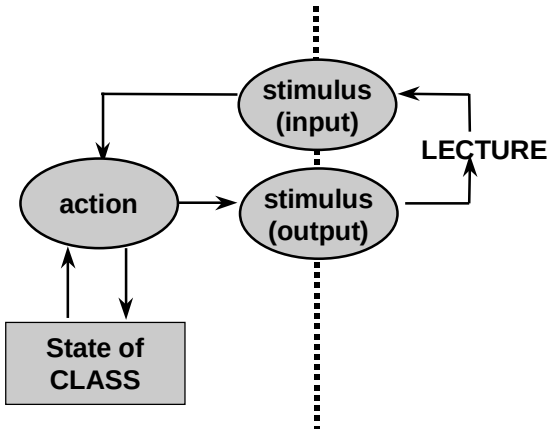
Tackling Complexity - Example

- **A system containing four buttons (B1 - B4) and two lights (L1 - L2)**
  - Since the last powering on, if B2 has been pushed more often than B3, then L1 shall be lit.
  - Since the last powering on, if B2 has not been pushed more often than B3, then L2 shall be lit.
  - At no time shall more than one light be lit.
  - If either light bulb burns out, the other bulb shall flash on and off in 2-second increments regardless of the number of B2 and B3 presses. This flashing shall cease when B4 is pressed and restart when B1 is pressed. When the malfunctioning bulb is replaced, the bulb shall cease to flash, and the system shall return to its normal operation.
- **What is normal operation, if we don't know whether the system records B2 and B3 presses while a bulb is broken ?**

Specifying Complex Behaviours

- *Need to formally specify behaviour of:*
  - *system in use*
  - *objects in system*
- *OOSE offers techniques*
  - *use case diagrams, interaction diagram,*
  - *state transition diagram*
- *UML provides notations*
  - *use case model, sequence diagram,*
  - *collaboration diagram, state diagram*
- *State diagrams provide essential means of showing how a class of objects evolves in response to external stimuli*

Modelling of States and Transitions



Finite State Machines

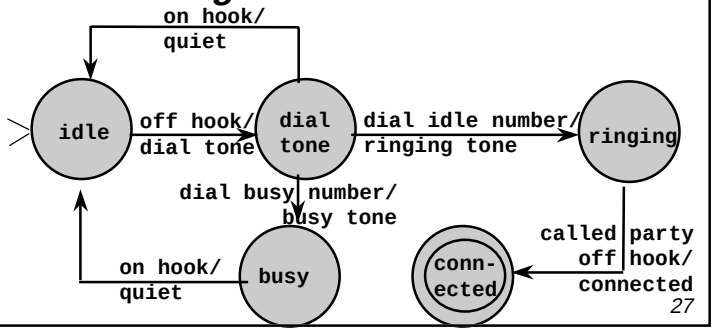
- *A formal model for states and transitions*
- *A finite state machine FSM is a five-tuple FSM=(S,A,σ, s, F) where*
  - *i) S, is a finite set of states*
  - *ii) A is a finite alphabet of events*
  - *iii) σ: S´A -> S, a partial function of transitions*
  - *iv) $s \in S$, a start state*
  - *v) $F \subseteq S$, a set of ending states*

FSM Execution Semantics

- *An FSM configuration is an element of SxA**
- *If FSM=(S,A,σ,s,F) is finite state machine then*
  - *i) A binary relation Γ is defined on configurations by $(q,w)\ \Gamma\ (q',w') \Leftrightarrow \exists \alpha \in A : (w=aw') \wedge (s(q,a)=q')$*
  - *ii) Γ^* defines the transitive closure of Γ.*
  - *iii) A sequence of events is acceptable by the finite state machine if there is an ending state $f \in F$ such that $(s,w)\ \Gamma^*\ (f,e)$.*

State Transition Diagrams

- **Graphical Notation for FSMs**
  - circle = state
  - double circle = finishing state
  - directed arc = transition two connected states
  - label = input/output events
  - hook = starting state
- **Example: STD for Making a Phone Call**



Motivation of State Diagrams

- **Real STD system models get very complex.**
- **Reasons of complexity:**
  - STD system model by can only be in one state
  - State is influenced by many factors
  - All factors need to be considered leading to exponential proliferation of states and transitions
- **State Diagrams manage complexity by**
  - Composition of states
  - Concurrent substates
  - Conditional transitions
  - History states

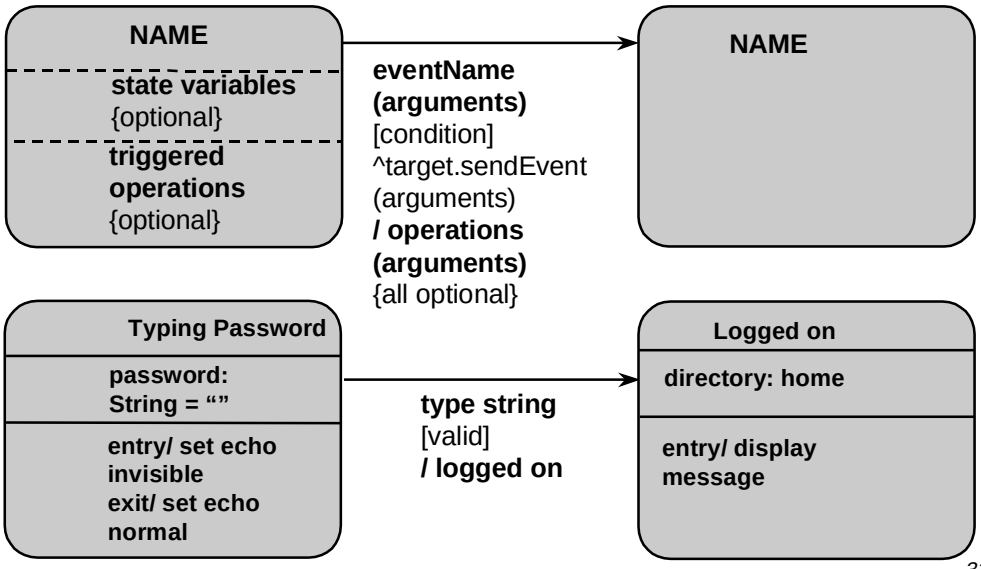
State Diagrams

- *A state diagram is*
  - *a directed graph of states connected by transitions*
  - *a formal specification of the behaviour of a class*
- *UML incorporates extensions to basic STDs made by Harel in his State Charts:*
  - *decomposition of states*
  - *default entry states*
  - *concurrent states*
  - *conditions on transitions*

STATE DIAGRAM CONCEPTS

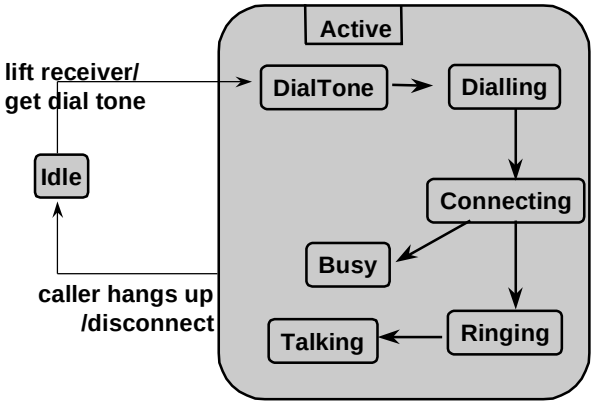
- *Three fundamental ideas :*
  - *event: an atomic occurrence at a point in time*
  - *state: a period in time during which an object is waiting for an event to occur*
  - *transition: a response to an external event received by an object in a certain state*

Basic UML State Diagram Notations

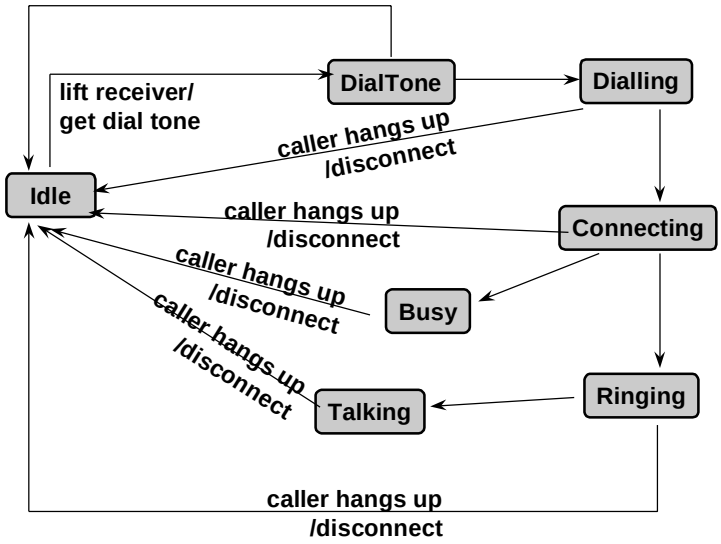


Composite States

- *A composite state is composed of substates.*



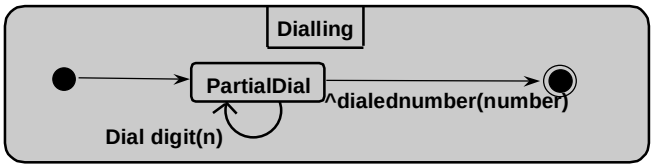
Equivalent STD



Composite States

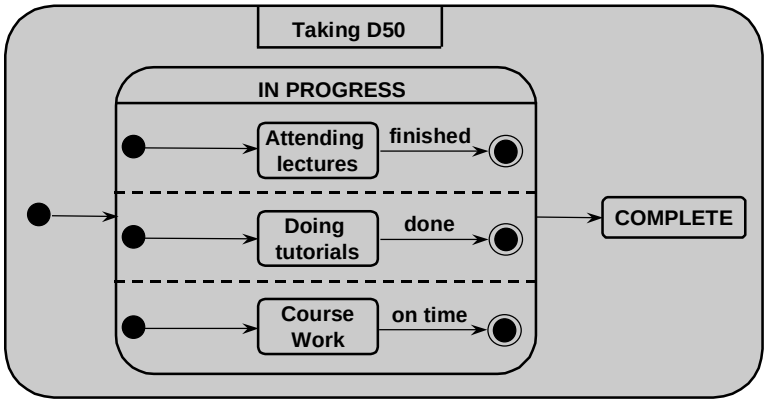
- *Depiction of substates can be omitted*

- *Default starting state begins at a circle*
- *Termination appears as a bullseye*
- *An event can be generated in another class using send event notation ^target.sendEvent*



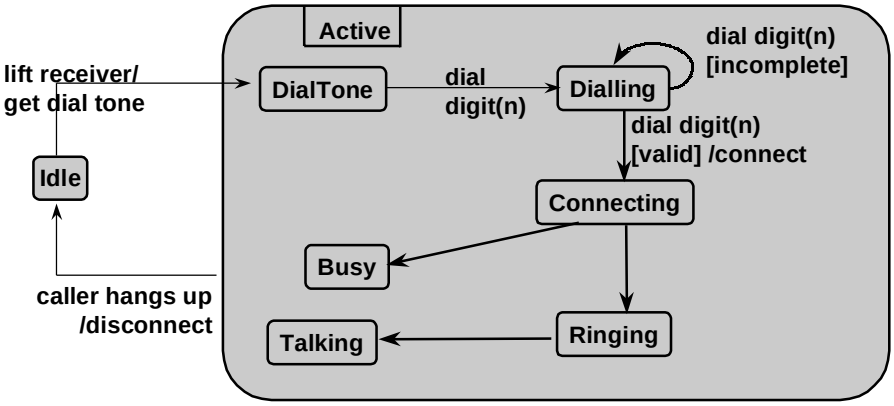
Concurrent Substates

- *When a state has multiple threads of control, each concurrent substate appears as a separate region separated by swim lanes*

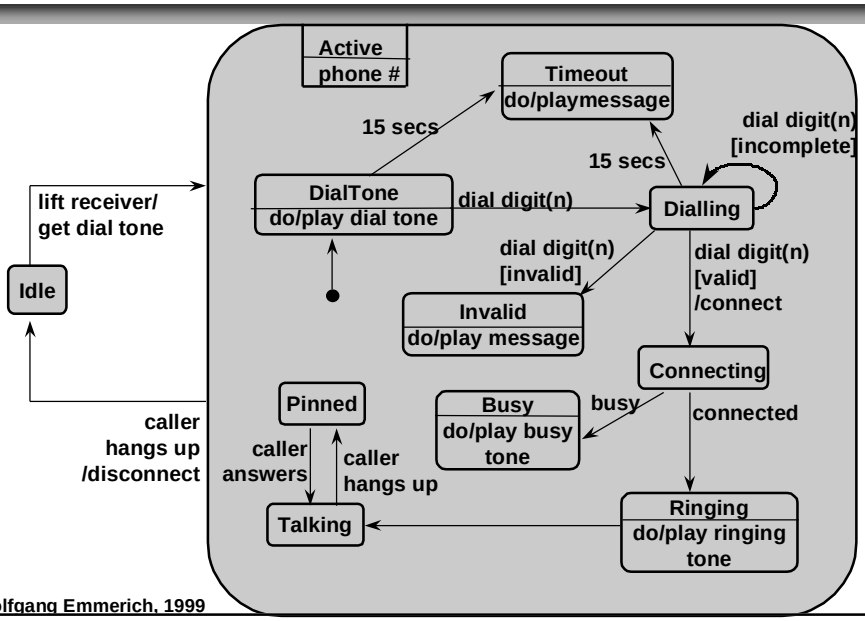


Conditions on Transitions

- *An optional guard [condition] may be attached to transitions after the event name*

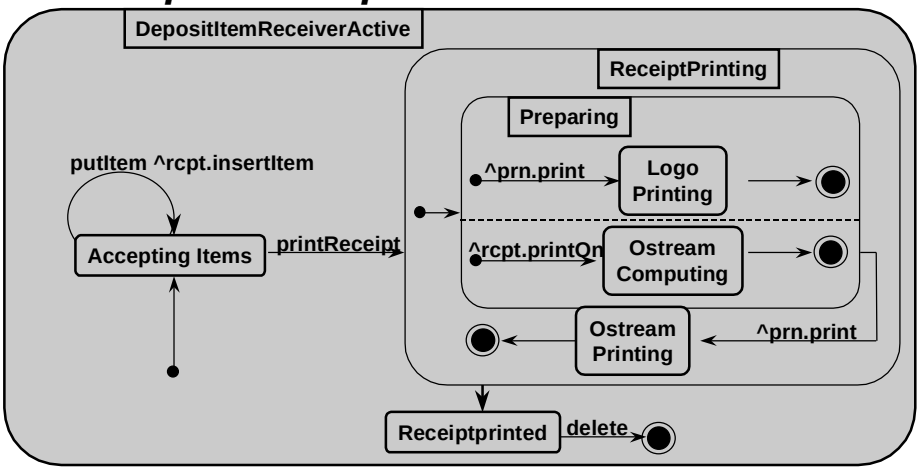


State Diagram for Telephone



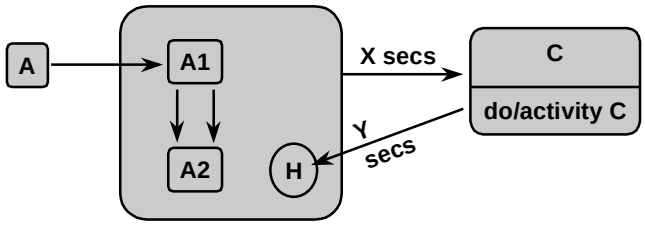
Recycling Machine State Diagram

- *One state diagram for each class*
- *Example for Deposit Item Receiver:*



Advanced Concepts

- *Activity: an ongoing operation within a state*
- *Elapsed time event: an event occuring*
- *a given time after entry into state*
- *History state: a state resumed upon reentry*



Role of state transitions in OOSE

- *To increase understanding of design blocks (classes)*
- *To model the 'state-controlled' objects, rather than the 'stimulus-controlled'*
- *To help in the abstraction of the actual code*
- *To describe stimuli received and what happens consequently*

Summary

- ***Design outputs:***
  - ***set of sequence diagrams [diagram x use case]***
  - ***elaborated class diagram***
  - ***state diagrams for classes that maintain internal states***
- ***Next lecture: Study how analysis and design can be supported by tools - Computer Aided Software Engineering***

Design Model Stages