Modeling Embedded Systems using UML

Abhik Roychoudhury
CS 4272 – Hardware Software Codesign

Ack: lecture notes of P.S. Thiagarajan

Modeling Languages

- UML – Unified Modeling Language
- Why model ?
- Simpler to build a model and test than build the real thing.
- Many details can be omitted.
  - Test Structure and Behavior
  - Don't think about implementation details.

Why Model?

- Modeling helps plan :
  - The structural pieces
  - How the pieces should behave
    - individually
    - Collectively
- Allows for growth and change.

System Modeling

- Use object-based methods.
  - Emphasizes the notion and reuse of components.
  - Hides implementation details.
  - Can lead to C++, JAVA code.
  - Widely accepted

Objects

- Object
  - Encapsulates data
    - Attributes
    - Fields
  - Supports operations (methods) on these data.
- Stack
  - Data : Function names
  - Operations : push( ) , pop( ), Is-Empty ( )

UML

- History
  - Booch
  - Rumbaugh
  - Jacobsen
- Rational Corporation
- Plus many others …
- OMG (Object management Group) recommended UML as a standard.

What kind of systems

- Highly reactive
  - Not necessarily data-intensive
    - ❑As in media-processing e.g. MPEG encoder/decoder
  - Can be control intensive, time-critical
    - ❑Automotive, avionics …
- Object-oriented
  - Line up with conventional SE.

What kind of objects?

- Not necessarily passive
  - State changed only by method invocation by other objects.
  - Can be active with control flow of its own.
- Concurrency
  - Can be multiple-threads also.
  - Even for single thread, various models for communication.

What kind of communication?

- Inter-object communication
  - Via events
    - ❑Objects in a class described via a State Diagram.
    - ❑Transitions in the State Diagram can be triggered by events.
  - Via Method Calls
    - ❑Can appear as annotations in the transitions of the State Diagram.
    - ❑Sequential flow of control.

Design Methodology ?

- UML is *not* a design methodology.
- It is just a *language*.
- Design Methodology :
  - A Language **plus**
  - A process (recipe) for using this language to produce a design.
- UML is meant for :
  - Iterative, rapid-prototyping styles.

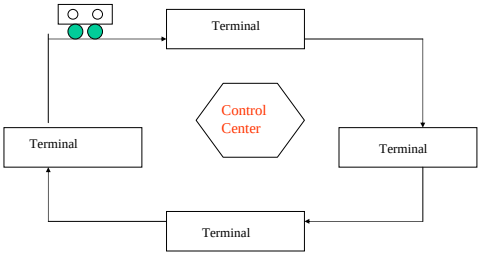
General Features

- Semi-Formal
- Discrete
- Universal
- Graphical notations
- **Weak** semantics
- **Weak** integration of the different views.

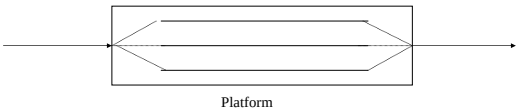
The Models of UML

- Requirement Models.
  - Meeting point between the client and the designer.
- Structural Models.
  - (Static) Relationships between Object classes. Actors.
- Behavioral Models.
  - Behaviors of the object classes (and interactions).
  - Close to implementation

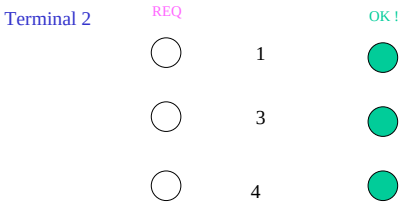
Rail Car System



Rail Car System



Rail Car System



Car

- Cruise Control
  - Off, engaged, dis-engaged
- Car should never come closer than 80 yards of any other car. (safety property)
- A stopped car can start only if the nearest car is at least 100 yards away.
- A car eventually gets permission to enter a platform (liveness property)

Passenger

- Passenger at a platform requests by pressing the chosen REQ button.
- When the car arrives at the platform that :
  - is going to the desired destination
  - is not full
- The appropriate “OK!” light flashes.

Requirement Models.

- What are requirements ?
  - IEEE Standard :
  - The requirements should include all the details the software developer needs to create a design.
    - ❑ functional
    - ❑ Performance
    - ❑ Technology Constraints
    - ❑ External Interfaces

Requirement Models.

- What are requirements ?
  - A Management Consulting Firm :
  - Requirements identify those facets of the business process which will be enabled by the new application……A solution that is feasible, cost-justified and has a high-degree of fit within the current culture and organization

Requirements Models of UML

- ***Use cases***.
  - Requirements will consist of a few –a dozen- use cases.
  - A use case captures a chunk of functionality which is externally visible.
  - System-level behavior rather than individual objects' behaviors and their implementations.

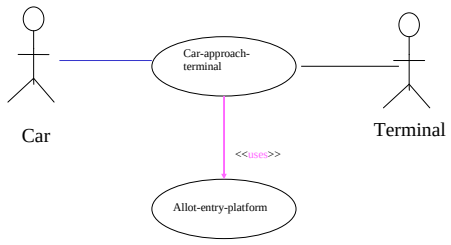
Use cases

- Use Cases
  - Each use case will have a set of *actors*
  - Actor is an object *external* to the system.
    - Human users, sensors, actuators
    - passengers
- Use cases can be related to each other.
  - Use other case as a sub-routine.
  - Be a specialized version of a general case.

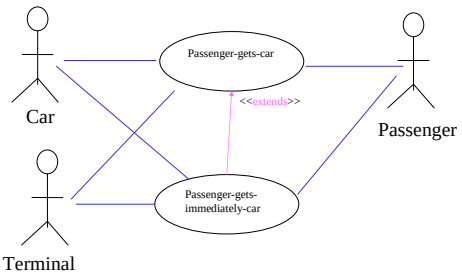
Use Cases

- Use case diagrams are static.
- Examples :
  - Car approaching terminal
  - Car departing terminal
  - Car passing through terminal

Use Case Diagrams



Use Case Diagrams

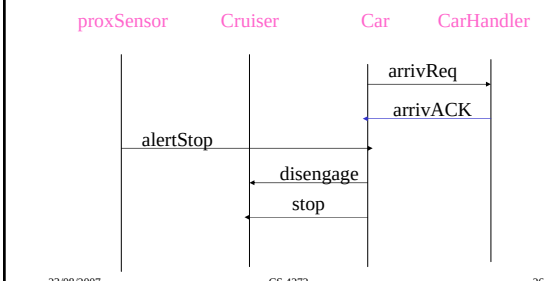


Use Cases

- Use Cases are captured via :
  - Use case diagrams (static)
  - Sequence diagrams (Message Sequence Charts, Scenarios, sequence charts) (dynamic)

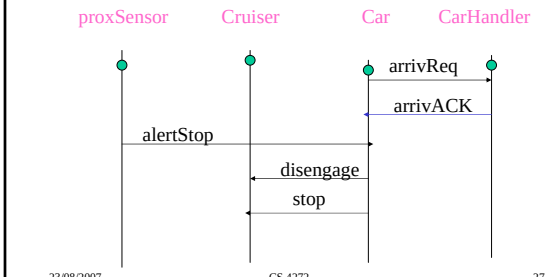
Sequence Diagrams

Car Approaching Platform



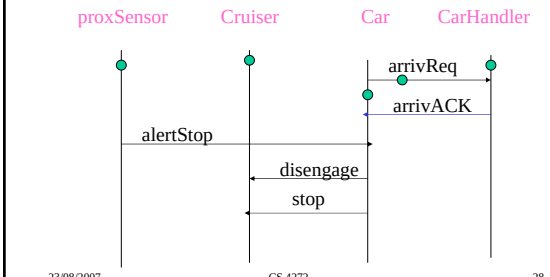
Sequence Diagrams

Car Approaching Platform



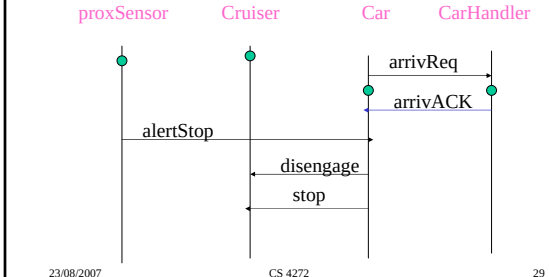
Sequence Diagrams

Car Approaching Platform



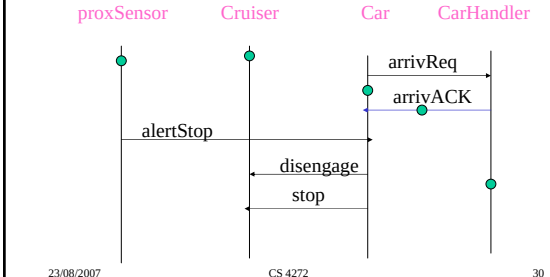
Sequence Diagrams

Car Approaching Platform



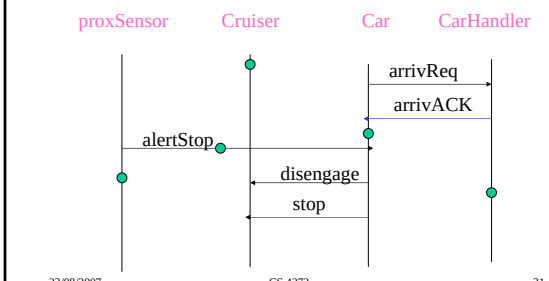
Sequence Diagrams

Car Approaching Platform



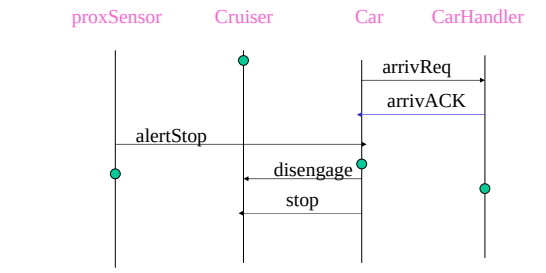
Sequence Diagrams

Car Approaching Platform



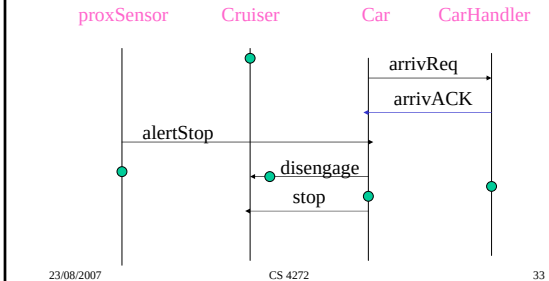
Sequence Diagrams

Car Approaching Platform



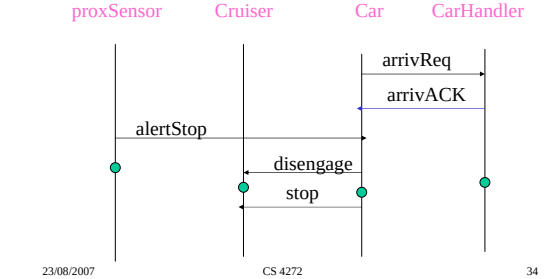
Sequence Diagrams

Car Approaching Platform



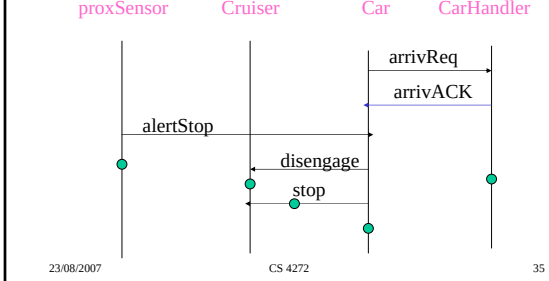
Sequence Diagrams

Car Approaching Platform



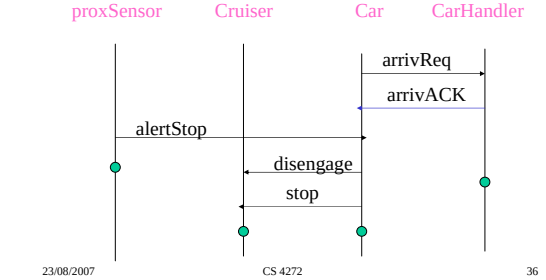
Sequence Diagrams

Car Approaching Platform

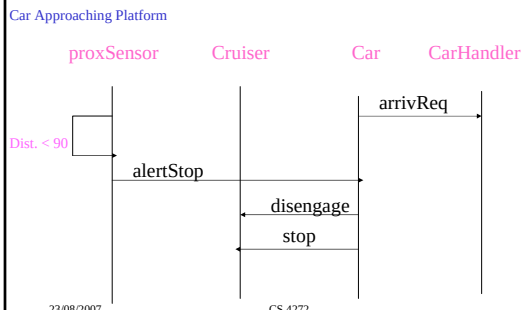


Sequence Diagrams

Car Approaching Platform



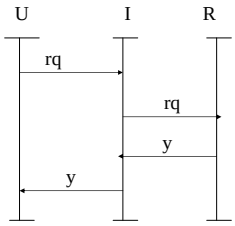
Sequence Diagrams

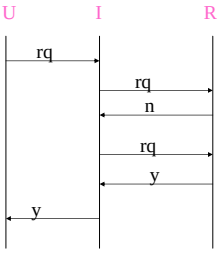


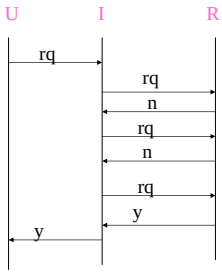
Sequence Diagrams

- Each use case will give rise to many sequence diagrams.
  - Some times infinite!
- Must find succinct ways of describing such collections of sequence diagrams.
- Augment sequence diagram syntax with conditions, branching, looping etc.
- Use HMSCs (MSGs)

Message Sequence Charts







Online Exercise

- How to capture all such interactions between User (U), Interface (I) and Resource (R) as a graph of MSCs?
  - What does an edge in such a graph mean?

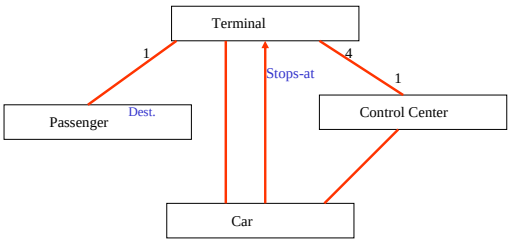
Structural Diagrams

- ***Class diagrams***
  - Key feature of object based methods
  - Show relationships between object classes
    - ❑ Associations
    - ❑ Sub typing
- Each Object class also has :
  - Attributes
  - Operations

Perspective

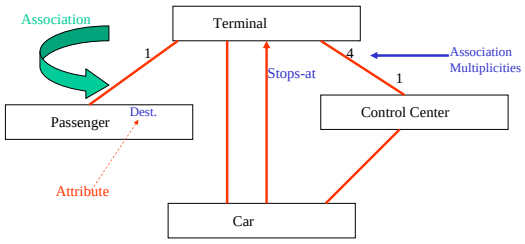
- ***Conceptual***
  - The Classes may be there to just help identify the key concepts.
  - May not be implemented directly
- ***Specification***
  - The classes specify the desired attributes and operations i.e. *interfaces* of the classes to be implemented.

Class Diagrams

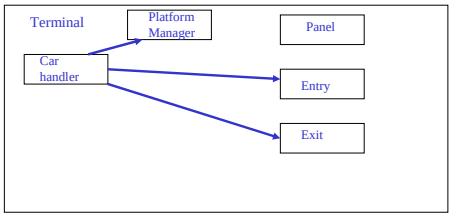


Can also specify object multiplicities in each class, special case is *

Class Diagrams



Object Classes



Aggregation shown here.

Class Diagrams

- There are many other features
  - Generalization
  - Constraint rules
  - Stereotypes
    - ❑ High level classification of objects such as “controllers”, “coordinators”
  - Dynamic classification
    - ❑ object – type relationships

Class Diagrams & Behavior

- Refer to only system structure?
  - Elaborated by State Diagrams for each class.
  - Use cases can be elaborated by Sequence Diagrams.
- Can also refer to initialization of behavior
  - How many objects in each class?
  - How many tuples in each association?

Initialization of Behavior

- Unambiguous
  - From class multiplicities and association multiplicities.
- Ambiguous, but bounded
  - Get the bound from object multiplicities.
- Otherwise
  - How to initialize?

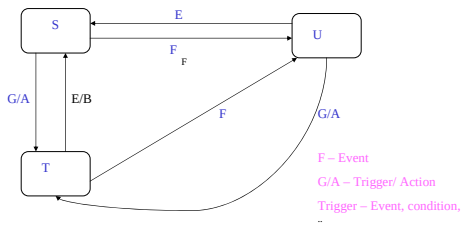
Behavioral Models

- Behavioral Models
  - State charts
  - Activity diagrams
  - Sequence diagrams
  - Collaboration diagrams
- Used to capture the behavior of object classes.
- Closer to implementations.

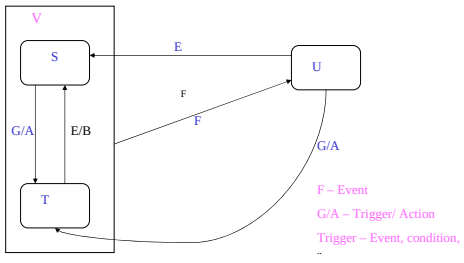
State Charts

- Due to David Harel (1984 … 1987 !)
- A significant extension of FSMs (Finite State Machines)
  - OR states
    - ❑ Composite states, sub-machines, hierarchy
  - AND states
    - ❑Concurrency, orthoganality
  - Many Other features !

Finite State Machines



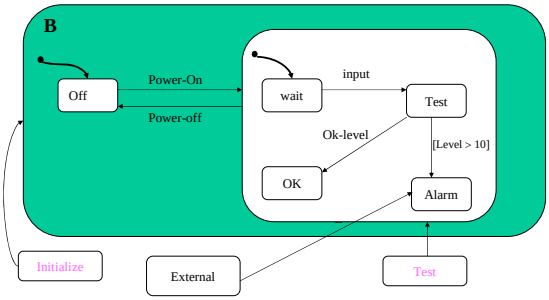
OR States



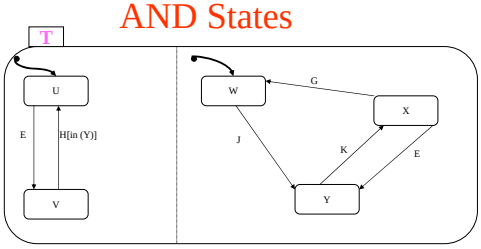
OR States

- V is an OR state
  - S or T
  - F can occur at S OR T. In both cases the machine goes to U.
  - From U
    - ❑ if E occurs then it goes to S
    - ❑ If G occurs, then A is executed and it goes to T.

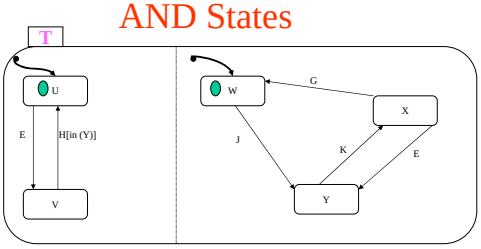
Multi Level OR States



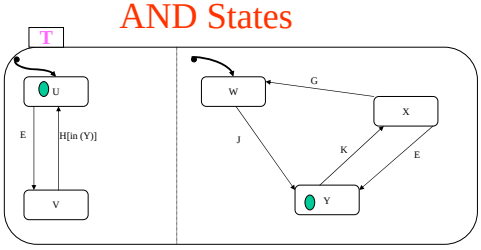
AND States



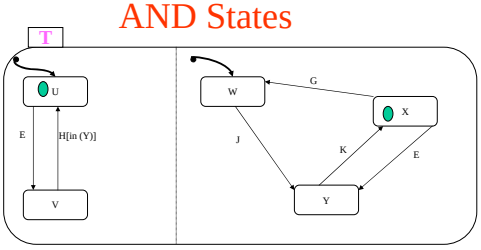
AND States



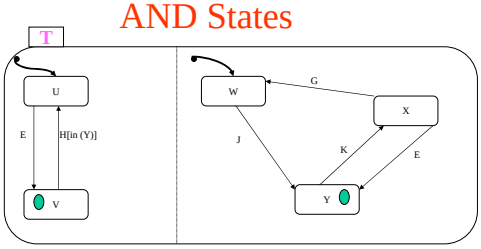
AND States



AND States



AND States



AND States

- If an AND state A has two orthogonal components and each component has 100 OR states then the total number of states in the "flat" automaton representing A could have 10000 states !
- State explosion.

State Charts

- If G/E occurs in a component then E is available during the next "tick" to all transitions whose trigger part contains E.
- Communication is, in this sense, broadcast.
- Connectors can also be quite complicated.
  - Fork
  - Join
  - …

Recap.

- Requirement models
  - Use cases : use case diagrams, sequence diagrams, collaboration diagrams.
- Structural models
  - Class diagrams : attributes, operations, associations and many other features.
  - Static
- Behavioral models
  - State Charts : OR states, AND states, Complex connections, support hierarchy and concurrency.

Status

- Very popular for documentation
- Tools : Rational Rose, **Rhapsody**, ..
- Semantics ?
- Consistency ?
- Code generation ?
- Verification ?

References

- Martin Fowler and Kendall Scott : ***UML Distilled : Applying the Standard Object Modeling Language.*** Addison-Wesley, (1997)
- David Harel and Michal Politi : ***Modeling Reactive Systems with State Charts: The STATEMATE Approach.*** McGraw-Hill, 1998
- See also references in the above references!

Question 1

- In the first lecture, we discussed the Y-chart approach towards embedded system design. In this approach, we evaluate an architecture instance and an "application". The key here is how the application is represented. In class, we saw various stages of developing an application --- from UML models to code. Suppose we have two Y-charts --- one operating at the UML model level and another at the code level. How will these two Y-charts be linked for the purpose of system design?

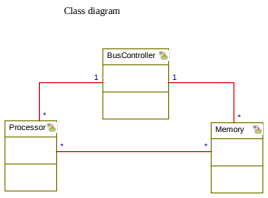
Answer to Q1

- Ans: The Y-chart at the UML model level will be used to do a high-level performance analysis and ruling out parts of the design space. For the remaining design points, we can do a more detailed analysis by following a Y-chart approach at the code level. This will be the overall strategy for design space exploration.

Question 2

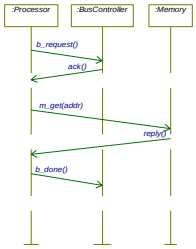
- Consider a multi-processor System-on-Chip (SoC) with multiple processors requesting for bus access to read/write to several memory/peripheral modules. We want to model the communication between the different components in this system using the Unified Modeling Language (UML). What will be the classes in the system and what will be the associations? Also, list some use cases from the point of view of a processor which is trying to access the bus. You may list the use-cases in English or elaborate them using Sequence Diagrams.

Answer to Q2.

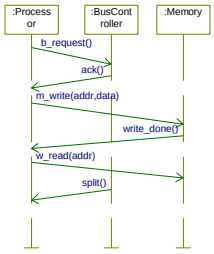


Answer to Q2

Use cases



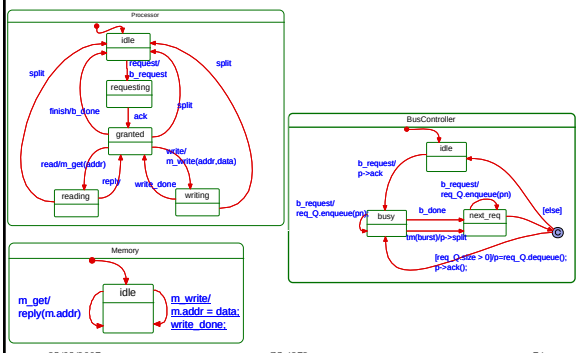
Answer to Q2



Question 3

- Elaborate the design by filling in the State Diagrams of each class you identified. Your design must satisfy the following criteria --- (a) at most one processor must access the bus at any time, (b) if there are one or more processors requesting the bus, the bus should not be idle, (c) any processor requesting the bus should eventually get access to the bus. Clearly state what parts of your Statechart design are ensuring each of these three properties. If you make any assumptions for ensuring these three properties, you should clearly state all your assumptions.

Answer to Q3



Explanations for Q3

Only one processor (denoted as *p*) can access the bus at any time. All other processors sending requests to the bus controller will be added to a FIFO queue (*req_Q*).

If the queue is not empty, i.e. one or more processors are requesting the bus, bus controller will immediately acknowledge the next requesting processor after current processor's communications are done or killed, which ensures the bus not idle.

A processor will inform the bus controller when its communications are done (modeled as an external event *finish*). Or it can occupy the bus for at most some amount of time (the *burst*). Thus, any processor requesting the bus will eventually get access to the bus, since the bus controller maintains a queue of waiting processors (who are waiting to access the bus). We of course rely on the assumption that any processor which is granted bus access does not occupy the bus for an indefinite amount of time.