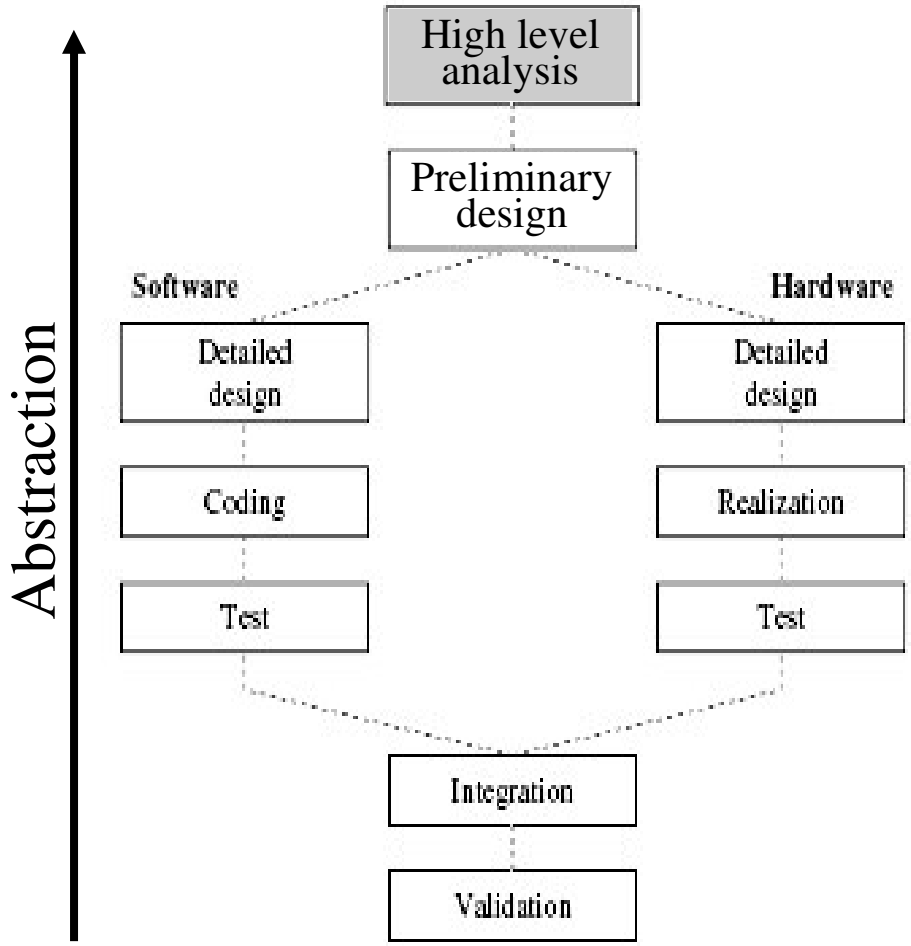
Real-time Systems – Design and analysis

Chapter 2: Analysis of Real-Time Systems

Outline

- General presentation for analysis stage
- Requirements Analysis
- Structural Object Analysis
- Behavioral Object Analysis

Analysis in the Methodology for Real-Time Systems Design



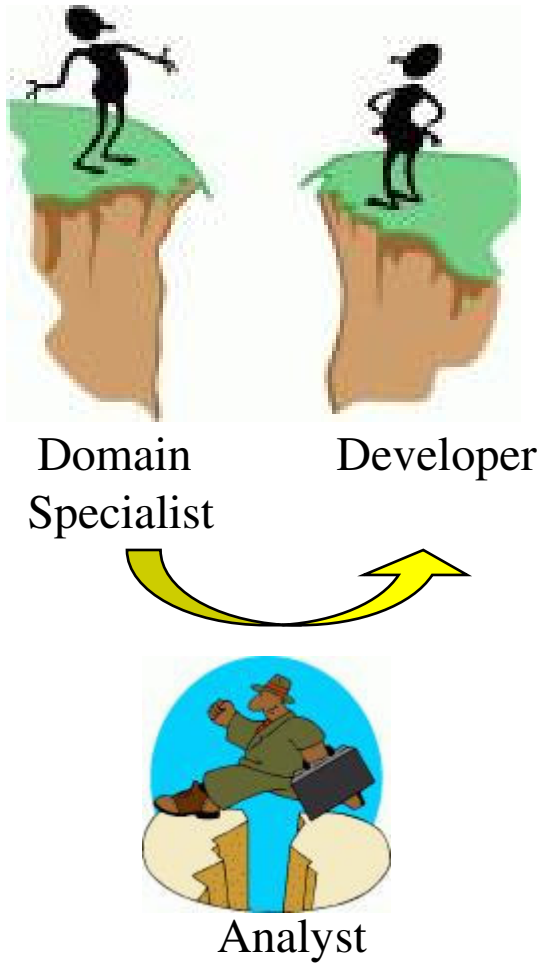
- Analysis consists of identification of all characteristics of the system that are essential to correctness
- It should be devoid of design characteristics that are free to vary
- Creates a single consistent abstract model, which is easy to maintain and closely follows the "real world"

Analysis Stage – General Presentation

- The main phases of the analysis stage are
  - Requirements analysis
  - Object analysis
    - Structural analysis
    - Behavioral analysis

Requirements Analysis of Real-Time Systems

- Requirements Analysis
  - Most of the design specialists are not used to rigorous thinking along the same lines as system developers
    - This disparity remains one of the key gap that must be closed by the annalist



Analysis Stage – General Presentation

- Requirements analysis
  - Discovers, in detail, the requirements of the final product
  - Extracts requirements from the customer (person in charge to define what the system do)
  - Challenges
    - Most customer understand the field use of the system but they don't think about it in a systematic manner
    - Customer may specify requirements that are mutually exclusive, impossible or impossible to implement
    - Customer may forget to specify "obvious" requirements and may specify implementations
  - It is almost exclusively a functional view. No classes and objects are identified.

Analysis Stage – General Presentation

- Structural and behavioral analysis
  - The essential objects and classes are identified and their important properties captured
  - These objects and classes must respect the requirements defined during the requirements analysis phase
  - This is the point at witch objects and classes appear
  - Only the objects and classes that are essential to all possibly correct solutions are captured
  - The structural and behavioral analysis sub-phases are most often done concurrently

Outline

- General presentation for analysis stage
- Requirements Analysis
- Structural Object Analysis
- Behavioral Object Analysis

Requirements Analysis of Real-Time Systems

- Primary concerns
  - Determine the main “clumps” of functionality that appear to be relatively independent and refine their behavior in an understandable and unambiguous way
  - Identify actors in the external environment
  - Identify the semantics and characteristics of the individual messages that pass between the system and its set of actors
  - Refine the protocols using the individual messages

Requirements Analysis of Real-Time Systems

- System requirements
  - Functional
    - Expectations of system behavior as seen from outside the system
    - Devoid of performance, reliability or safety considerations
  - Quality of Services (QoS)
    - Specify the necessary performance, reliability and safety of the functional requirements (e.g. image decoding)
    - Never stand alone – they elaborate one or more functional requirements (e.g. decoding resolution is 40 frames/second)

Requirements Analysis of Real-Time Systems

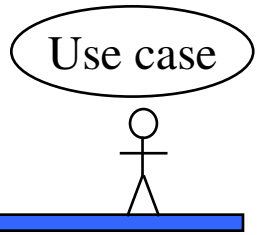
- Terms and concepts
  - Messages and events
  - Scenarios, protocols and state machines
  - Primary tools
    - Use cases
    - State charts
    - Sequence diagrams

Unified Modeling Language (UML)

- UML is a language for expressing the constructs and relationship of complex systems
- It was jointly developed by some of the major companies (e.g. Texas Instruments, Microsoft, I-Logics, Oracle,…)
- It was accepted as a standard by OMG (Object Management Group's)

Use Cases in UML

- A *use case* is a cohesive piece of functionality of the system that is visible from exterior. It is a function that returns an observable value to an actor
- *Actors* are objects that exists outside the scope of your system
- A use case is not a class, it must be ultimately realized by collaborations of objects working together to achieve the use case functionality
- Use cases can receive messages from actors and send messages to them
- A use case may have relations with other use cases
- Use cases are shown as ovals with solid borders
- The icon of an actor is a figure

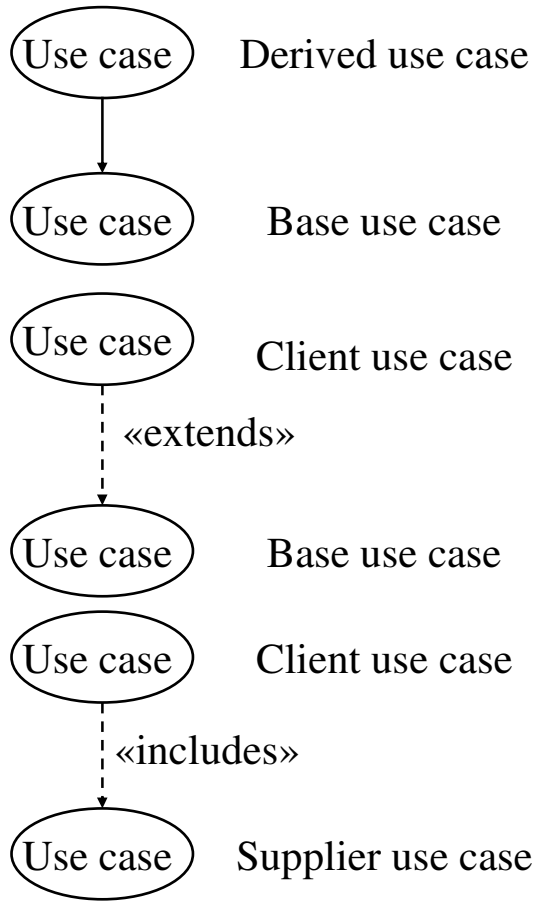


Sequence Diagrams in UML

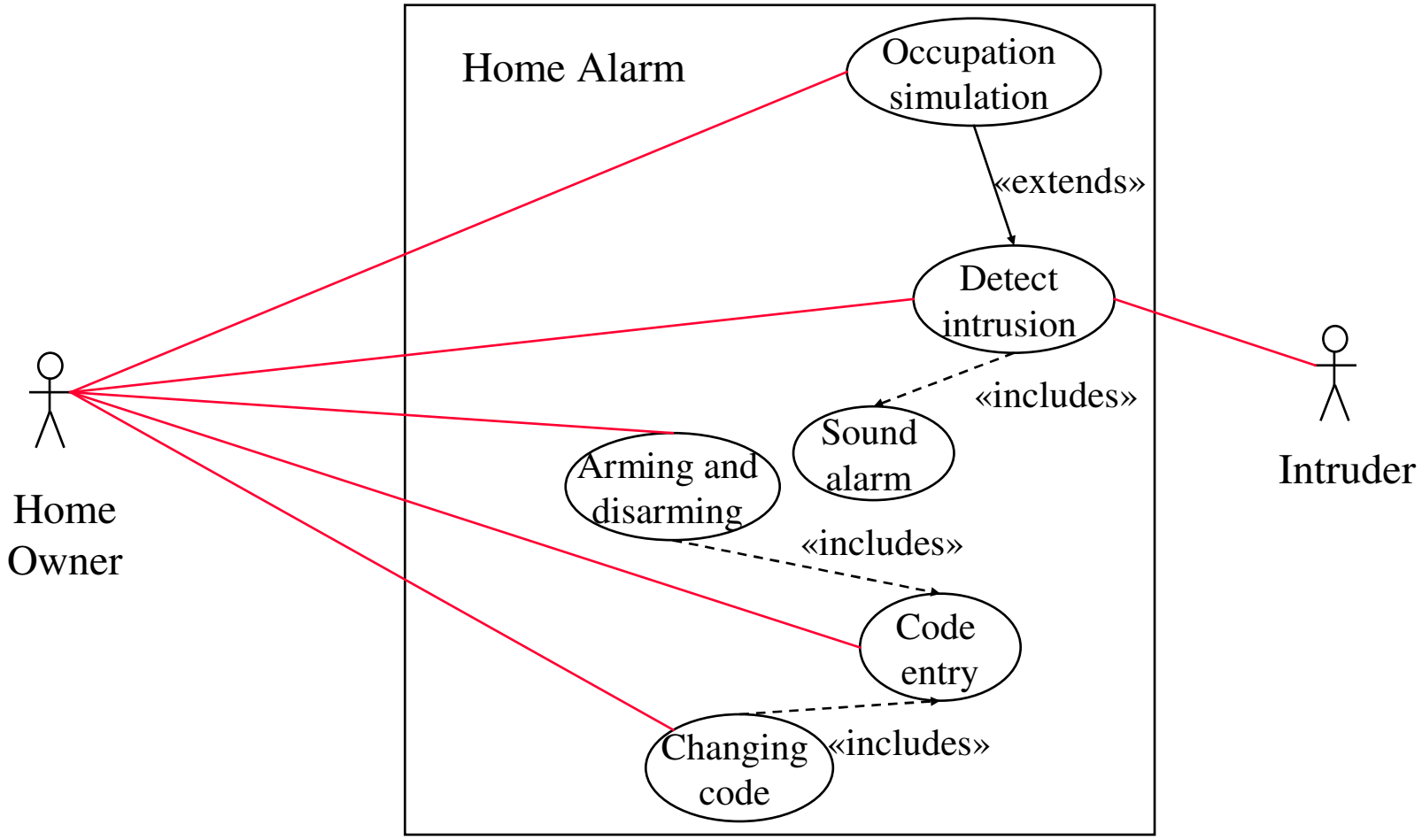
- A particular path through a use case is a *scenario*
- *Sequence diagrams* are used to capture the set of interesting scenarios
- Sequence diagrams may capture also the sequence of messages between objects

Requirements Analysis of Real-Time Systems - Use cases relations -

- Generalization relation
  - One use case is more general then an other (just as with class generalization)
- Extension Relation
- Includes Relation
  - It defines a behavioral segment that will be included in the base use case.

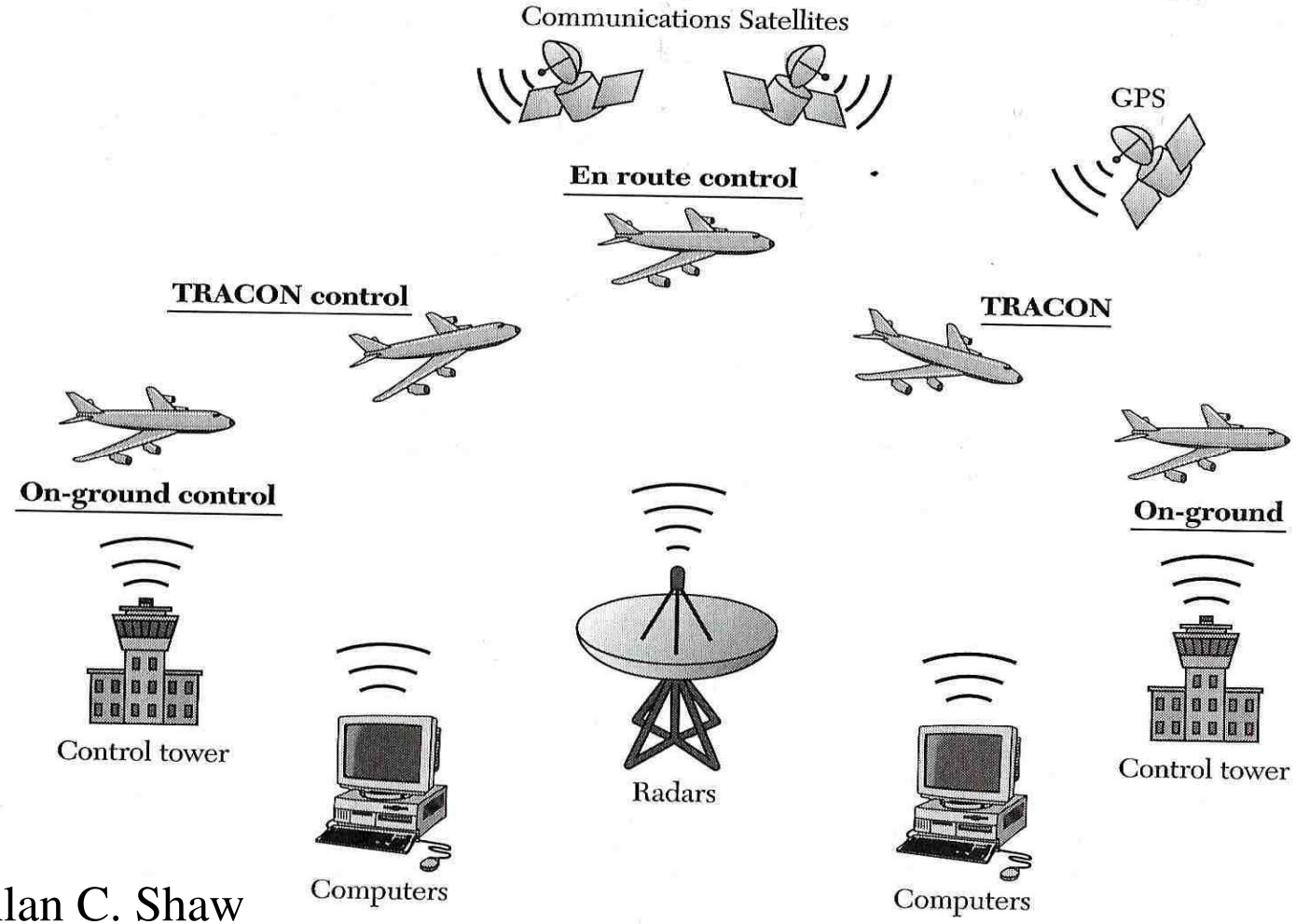


Requirements Analysis of Real-Time Systems - Example of Use Cases Diagram in UML -



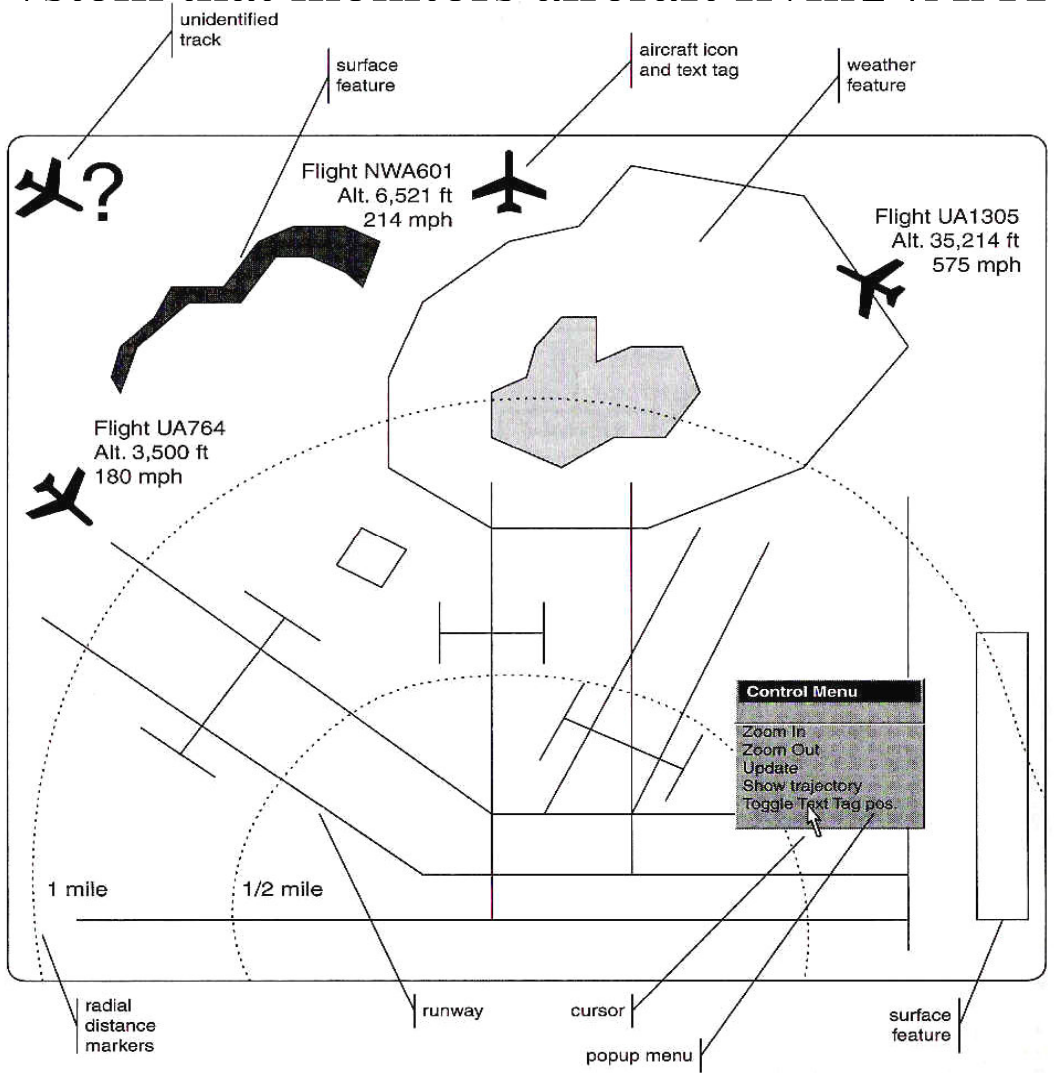
Requirements Analysis of Real-Time Systems - Example of Use Cases Diagram in UML -

- Real-Time System that monitors aircraft flying (AATC)

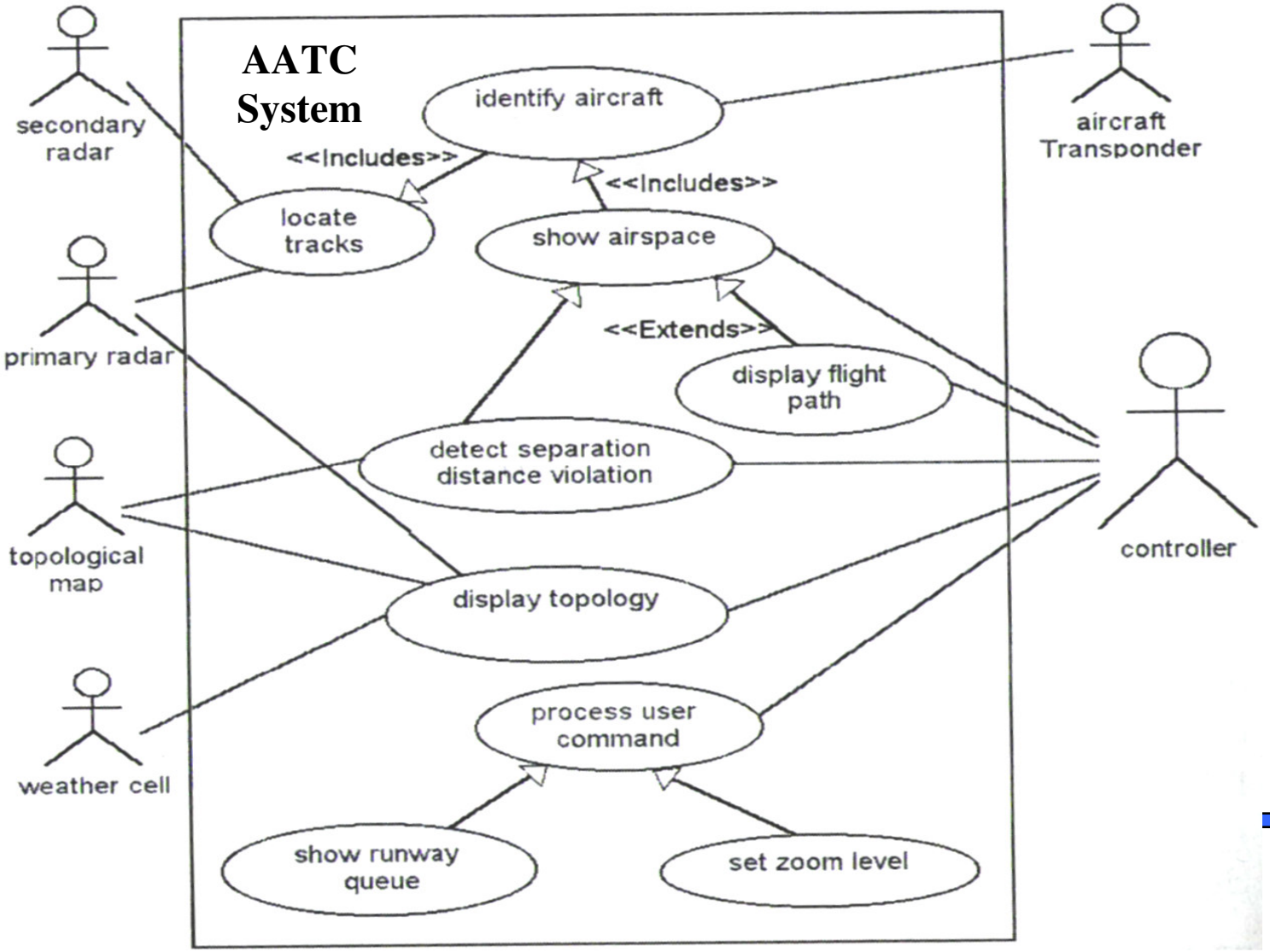


Requirements Analysis of Real-Time Systems
- Example of Use Cases Diagram in UML -

- Real-Time System that monitors aircraft flying (AATC)



Requirements Analysis of Real-Time Systems - Example of Use Cases Diagram in UML -



Requirements Analysis of Real-Time Systems - Messages/Events -

- A message is an abstraction of the exchange of information between a sender and a receiver
  - Logically messages consist of several aspects
    - Semantic content
      - Meaning of the message (e.g. an intrusion is detected)
    - Signature
      - List of parameters which aid the understanding of the black-box level of system abstraction
    - Quality of services of the message
      - Message arrival pattern
      - Message synchronization pattern
- During the analysis phase, only the essential properties of the messages are captured. Subsequent design will define the implementation of each message

Requirements Analysis of Real-Time Systems - Messages/Events properties -

- Arrival pattern
  - Describes the timing behaviour of the messages
    - Periodic
      - Period
      - Jitter
        - The variation with which messages arrive
    - Aperiodic (Episodic)
      - Minimum (maximum) inter-arrival time
        - Minimum (maximum) time that must occure between message arrivals
        - Average arrival rate
        - Autocorrelative dependencies (e.g. bursty messages)

Requirements Analysis of Real-Time Systems - External Events List -

- Detailed list of the environmental events/messages that are of interest for the system
  - Events/messages definition
    - Name
    - Description
    - Direction
  - Arrival patterns
  - Response performance

Requirements Analysis of Real-Time Systems - External Events List Example-

- AATC External Event List

	Event	Description	Direction	Arrival Pattern	Response Performance
1	Primary radar ping	Periodic command to transmit a first radar pulse	To 1st radar	Periodic	20 ms period
2	Primary radar return	Surface reflection return from an aircraft or environmental feature	To system	Episodic	<10ms response
3	Secondary radar ping	Periodic command to transmit a 2nd radar pulse	To 2nd radar	Periodic	20 ms period
4	Secondary radar return	Transponder responds to a request	To system	Episodic	<10ms response

Requirements Analysis of Real-Time Systems - External Events List Example-

- AATC External Event List (continued)

	Event	Description	Direction	Arrival Pattern	Response Performance
5	User control movement	User moves cursor	To system	Episodic	<5ms response
6	Whether cell identification	External weather system updates weather cell map	To display	Periodic	<50ms response
7	Separation distance violation detected	Violation must be visually tagged on the screen	To display	Episodic	<50ms response
…	…	…	…	…	…

Requirements Analysis of Real-Time Systems - Detailing Use Case Behvior -

- Detailed specification of behavior implied by a use case has to be provided
- The most common ways are
  - Informal textual description
  - Formal textual description (e.g. expressed in programming language)
  - Scenarios
  - Statecharts

Requirements Analysis of Real-Time Systems - Informal Textual Description -

- Description realized in a natural language (e.g. English)
- Example

5.6.1 Informal Textual Description

Consider the use case "Detect airspace separation distance violation." A number of algorithms are possible. Let us suppose that the domain expert has given us one in the form of a textual description:

The system shall use an alpha-beta tracker:

$$\text{New position} = \text{alpha} \times \text{predicted position} + \text{beta} \times \text{measured position}$$

Measured position comes from slant range (use the speed of light and two times the distance because we are reflecting, and use the time at which the signal came in—that is, there is a counter from the time the transmitted signal went out) and azimuth angle at which the target replied. We need to average the azimuths from the multiple replies to get the azimuth of the "return." Now we have slant range and az angle. To get to a range parallel to the flat earth, we need to know altitude. From the secondary radar, you know the altitude if you can correlate the track from the primary (which has no ID info) and the secondary. Generally, this correlation can be done because the aircraft tracks from the different sources are very close. So now you have range, az, and alt. The primary and secondary radars may be co-located, but this is not the general case. When they are not co-located, a geometric translation must be performed so that they reference objects using the same spatial origin.

Now you have measured position. So all you need is old, predicted position, which is obtained by extrapolation. In the beginning of the track, there is no old, predicted position to work with. So at first, position is measured position. Then on the second scan, you can measure velocity by subtraction (these are not Doppler radars). At first, you can't extrapolate with acceleration, but after a few measurements, you can use a finite difference as an estimate of the first derivative. That gives us position, velocity, and acceleration.

Separation is difference in position, and predicted separation is difference in predicted position.

Requirements Analysis of Real-Time Systems - Formal Textual Representation -

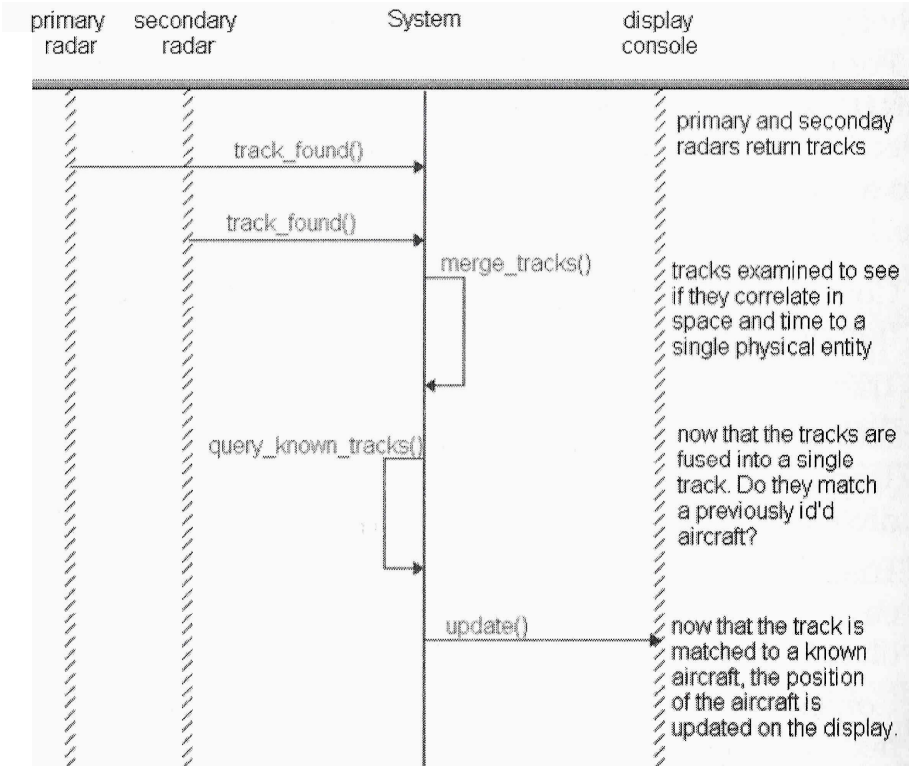
- Many programming languages have been especially designed or modified to handle the unique requirements of real-time problems
- Representative languages
  - Ada
    - International standard
    - It is very influential
    - It has a large and reach set of facilities
  - Real-time Java
    - Extension of Java which is very popular general purpose and objec-oriented programming language
  - CSP and its OCCAM implementations
    - Used for both theoretical and practical works
    - Notable for its small and elegant features
  - Esterel
    - Event-oriented language for programming the reactive part of an application

Requirements Analysis of Real-Time Systems - Scenarios -

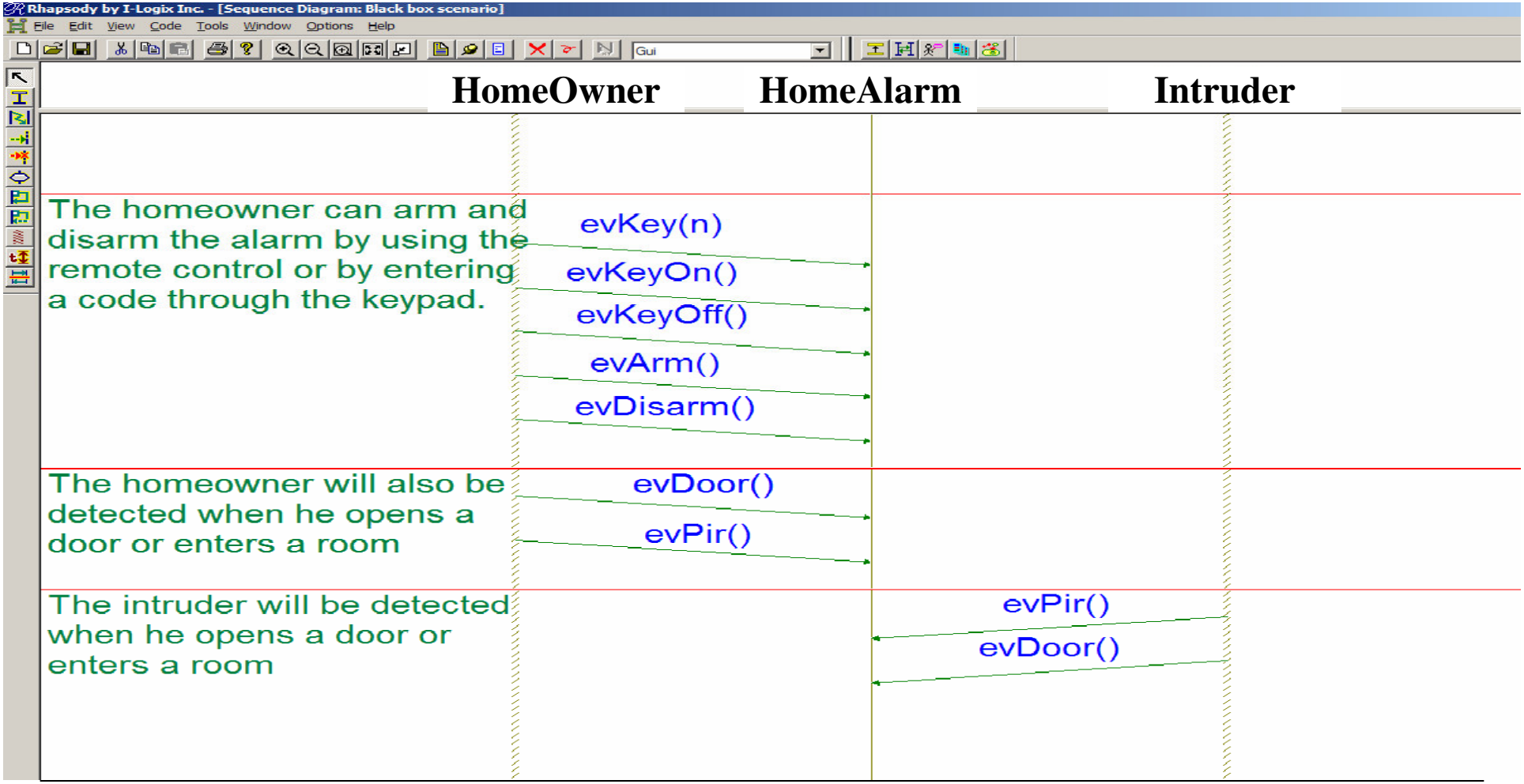
- Scenarios model order-dependent message sequences among objects collaborating to produce system behavior
- They provide an invaluable tool for validating the problem statement against the user's expectations
- Early in analysis, objects available for scenarios are the system and the external objects identified in the use case diagrams
- The most important scenario models existing in UML are sequence diagrams

Requirements Analysis of Real-Time Systems - Sequence Diagrams -

- Graphical syntax for sequence diagrams
  - The vertical lines represents objects; their name is written above/below the line
  - The horizontal directed lines are messages. The name of the message is written on the line. Some of the messages are internal to the objects
  - Time flows from the top to the page downward. The time axis shows only sequence; the scale is not linear
  - Textual annotations along the right side of the diagram identifies initial conditions, actions and activates not shown by the message sequence itself



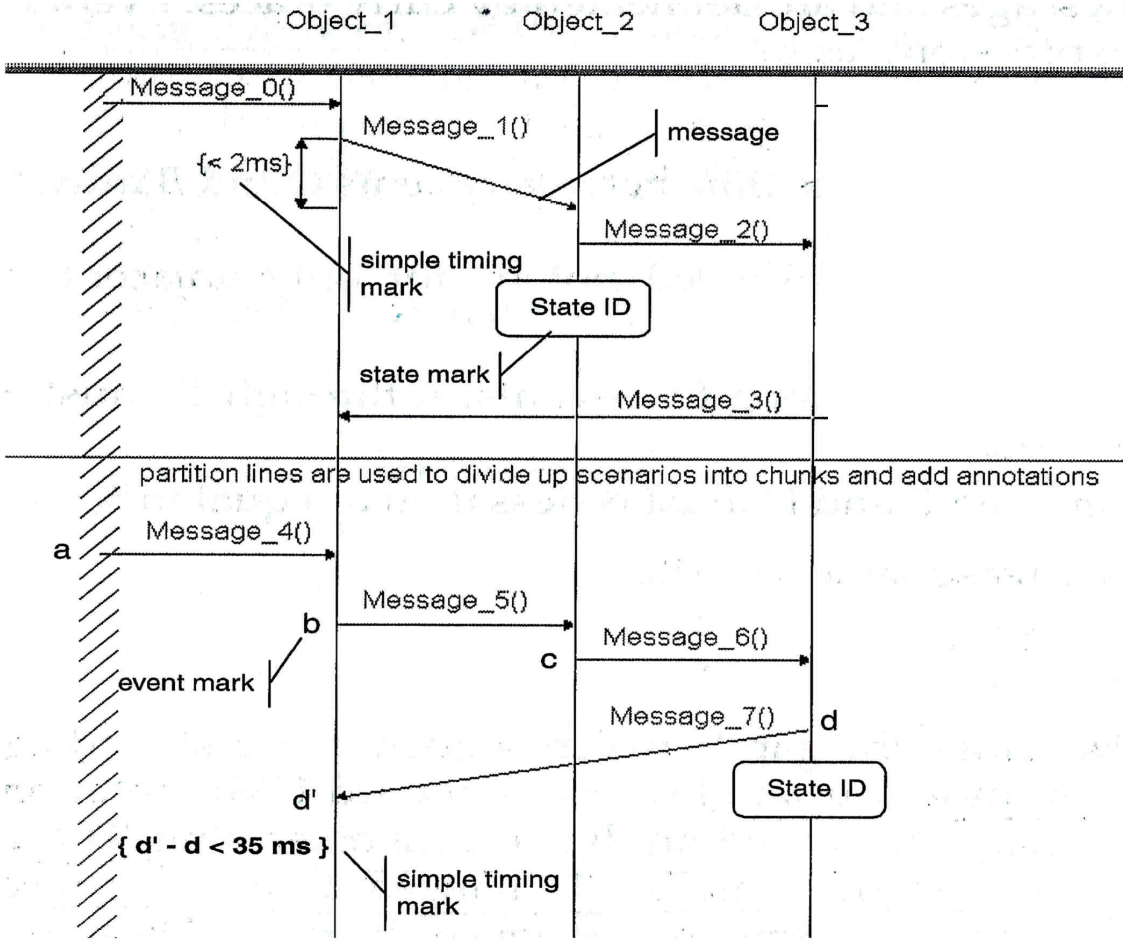
Requirements Analysis of Real-Time Systems
- Example of Sequence Diagram in UML -



Requirements Analysis of Real-Time Systems - Additional Features for Sequence Diagrams -

- Sequence diagrams can contain more information when necessary
  - Partition lines can be used to divide up scenarios into chunks and add annotations
  - Optional message identifier
  - Timing marks
    - Relational expressions indicating timing constraints; messages IDs are used
      - E.g. {c – b <= 5 ms }
    - Timing-mark expressions need not be for only adjacent events
      - Imagine 2 constraints
        - The sum of the time among 5 events, A through E, must be less then 10 s
        - The time between C and D must be less then or equal to 500 ms
      - The timing mark expression will be { e – a <10s
        d – c <= 500 ms}
  - State marks are rounded rectangles (standard UML notation) placed on the vertical object lines
    - This allows the scenario to follow the state as it changes because of incoming events

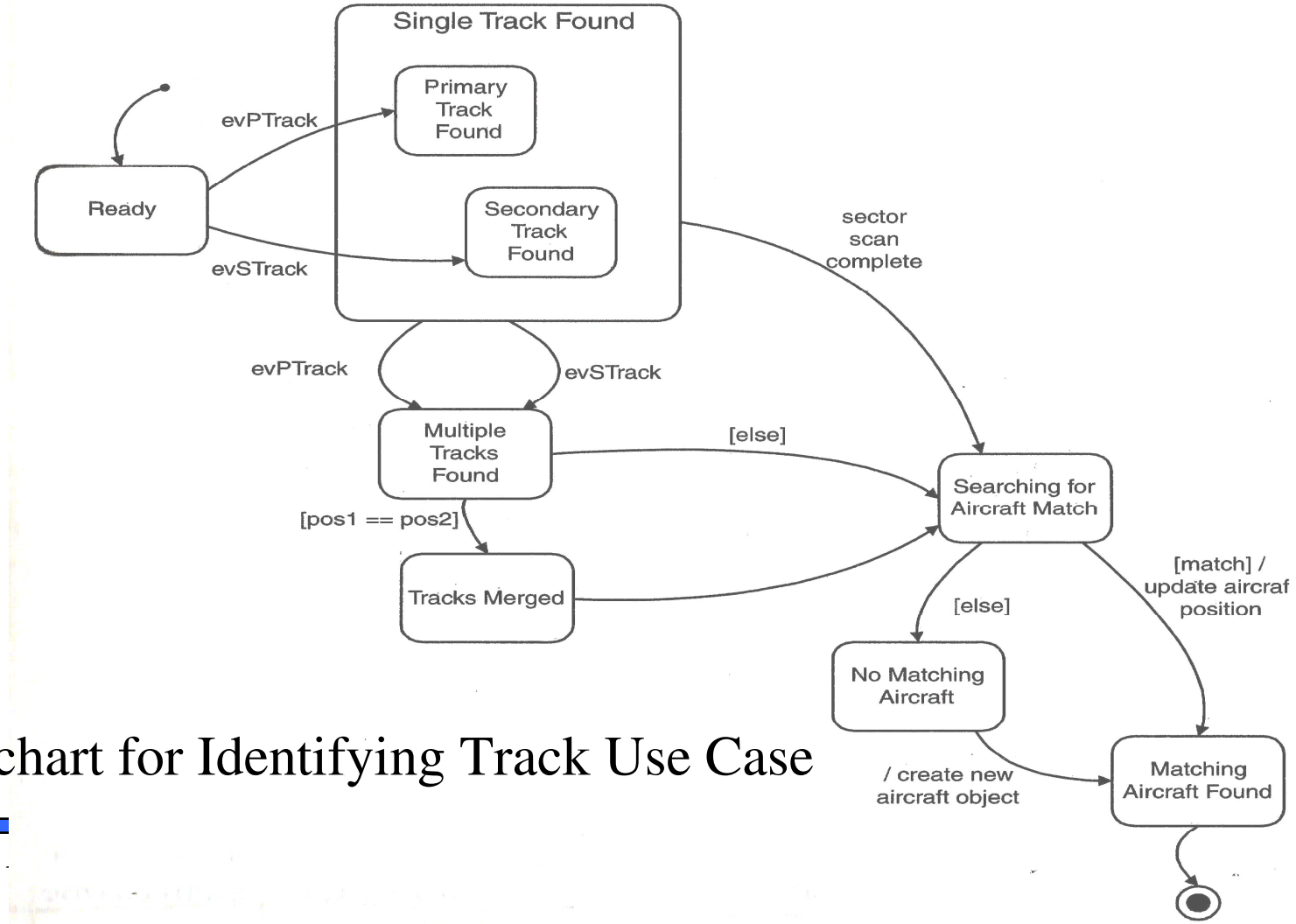
Requirements Analysis of Real-Time Systems - Additional Features for Sequence Diagrams -



Requirements Analysis of Real-Time Systems - Statecharts -

- Statechart – formal graphical language for specifying behavior
- One of the diagrams used to realize use cases
- Behavior is specified using state machines
  - States
    - distinguishable conditions of existence that persist for a significant period of time
  - Actions
    - a set of behaviors performed in each state
  - Transitions
    - means by which objects change states in respond to events
  - Transitions labels
    - Event_name ‘(’parameter_list’)’ ‘[’ guard‘]’ ‘/’ action

Requirements Analysis of Real-Time Systems
- Statecharts -



State chart for Identifying Track Use Case

Requirements Analysis of Real-Time Systems - Identifying Use Cases -

- Although some domain experts may think in abstract terms, the vast majority are more comfortable identifying specific scenarios rather than use cases
- The analyst may sit with the customer and ask
  - What are the primary functions of the system?
  - What are the secondary functions of the system?
  - With what physical devices/systems must the system interface?
  - What are the different roles played by the human users?
  - Why is the system being built? What it is replacing and why?
- The analyst must then identify
  - Roles the actor and system play in each scenario
  - Interactions necessary to complete the scenario
  - Sequence of events and data to realize the scenario
  - Variations of scenarios that are possible
- The number of scenarios can quickly grow very large
  - All scenarios that invoke identical system responses constitute a single scenario
    - Primary scenarios – elaborate the major ways the use case is realized in practice
    - Secondary scenarios – relatively minor variations of the primary scenarios
- A moderate project will have a dozen use cases

Heuristics for Good Requirements Analysis Diagrams

- Use Case Diagram (UCD) Heuristics
  - Only in simple systems (>15 use cases, < 12 actors) show all use case on a single diagram
  - Show one use case generalization hierarchy per UCD
  - Show use of a common set of collaborating actors per UCD
  - Label each UCD with a note (the system name, the author, the revision number)
  - Actors may be humans if the system scope includes user interface devices
  - Actors may be devices rather than humans if the user interfaces outside its scope

Heuristics for Good Requirements Analysis Diagrams

- Use Case Heuristics
  - For rigorous use case definition, construct state charts
  - Minimize associations-line crossing
  - Represent complex protocols of interactions via statecharts bound to the relevant use case
  - Don't show associations among actors, only among actors and use cases
  - Mark extension points in whatever medium captures the details of the use case
  - If you have several domain experts, tag each use case with the relevant domain expert or reference documentation

Heuristics for Good Requirements Analysis Diagrams

- Use Case Sequence Diagrams (SD) Heuristics
  - Show only actors and the system as instances when the system is the scope
  - Build one SD per interesting scenario
  - Identify the pre-conditions for each scenario, as well as its purpose, using a text note
  - Identify the system, the use case, author and initial date of creation, and revision number for each diagram
  - Show timing marks to capture timing constraints
  - Multiple overlapping timing constraints should be analysed for compatibility

Heuristics for Good Requirements Analysis Diagrams

- Use Case Sequence Diagrams (SD) Heuristics (cont'd)
  - Mark states of the use case on the sequence diagram where the use case is represented with statecharts
  - Label each message meaningfully
  - Use horizontal lines for normal messages
  - Use slanted lines for events
  - Show arrival and synchronization patterns for messages where dictated by the problem domain

Requirements Analysis Artifacts

Artifact	Representation	Basic	Description
Requirements document	Text	Yes	Textual description of the system contents, interface to the external actors, and its externally visible behavior including constraints and safety requirements
Use Cases	Use case diagram	Yes	Identification of the major functional areas of the system the interaction of the actors with the use cases
	Statecharts	No	Capture the full behavioral space of use cases
	External event list	No	Spreadsheet describing the properties of events received/issued by/from the system
	Context diagram	No	Contains the system and actors that interact with the system. Messages and events passed between actors and the system are identified

Requirements Analysis Artifacts (continued)

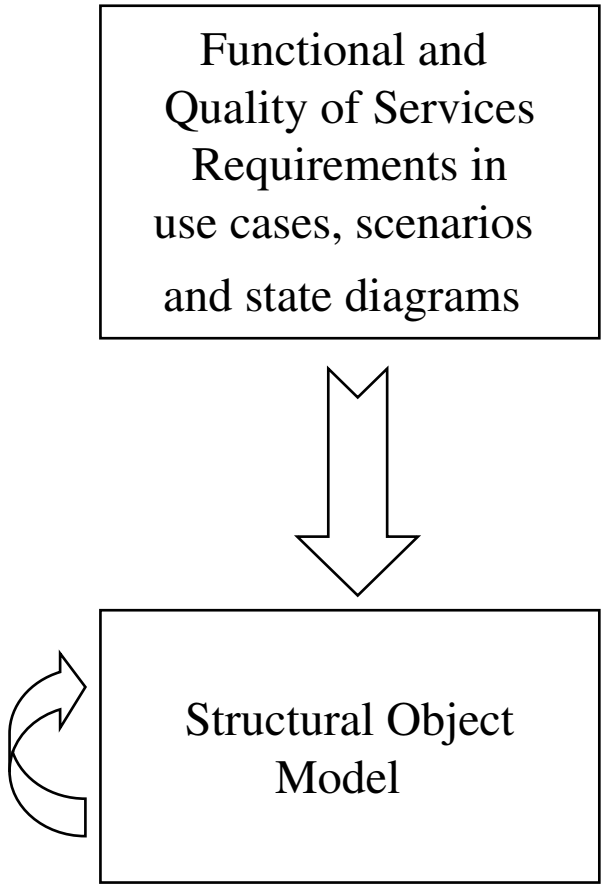
Artifact	Representation	Basic	Description
Use case scenarios	Sequence Diagram	Yes	Specific Path through a use case
	Timing Diagrams	No	Instances of use cases. Usually applied to reactive use cases, they show states along the vertical axes and linear time along the horizontal axis.
Hazard analysis		No	Document that identify the key hazards that the system must address and their properties such as severity and probability of the hazard, its computed risk
Test vectors	Textual	Yes	Specification of test to validate the system against requirements

Outline

- General presentation for analysis stage
- Requirements Analysis
- Structural Object Analysis
- Behavioral Object Analysis

Structural Object Analysis

- Identify essential objects that appear in the problem domain
- Identify essential attributes of the classes
- Identify essential relations among the classes and objects
- Generalization taxonomies
- Associations (including multiplicities)
- Capture the structural object model in the tool repository
- Create diagrammatic views and reports of the structural object model
- Not all classes/members are shown on a single diagram



Key Strategy Structural Object Analysis
- Identify only the essential objects and classes -

- The job of the analysis model is to identify the required aspects of the solution so that any delivered system lacking these aspects would be considered incorrect or incomplete
- The essential object model is the model that contains only those objects without which any possible delivered system would be incorrect
- The essential model separate lower level decisions from important decisions
  - Experience has shown that defects in analysis are orders of magnitude more expensive to correct than implementation defects, because the number of dependencies is higher on a model's fundamental constructs

Key Strategy Structural Object Analysis
- Underline the nouns -

- Used to cut a first-cut object list; the analyst underlines each noun and noun phrase in the problem statement and evaluates it as a potential object

When an aircraft is cleared for taxiing, it taxis from its gate to the designated runway, where it awaits clearance to take off. Once final clearance is given (subject to current runway use and other flights awaiting final clearance), the plane is allowed to take off using the runway designated by the tower controller. At any time after a plane has left the gate but before it is given take-off clearance, it may be routed to a holding area. This may be due to environmental or scheduling conditions.

When an aircraft approaches for landing, it does so along its flight plan. As the aircraft approaches the airport, it must request clearance to land. The controller may grant clearance and provide a runway ID and directions, or he may give an alternate flight path, such as circling the airport at a designated altitude, distance, and air speed.

The system maintains the original flight plan, along with an actual flight plan, which is updated during the course of the flight. All aircraft in the local airspace are monitored using a primary radar, which uses a rotating emitter/receiver and aircraft skin reflection to identify targets in the airspace. The primary radar masks out ground-based obstructions (such as mountains and buildings). Processing associated with the primary radar identifies tracks based on computed position (requiring direction and time-of-flight of signal) and the last known position of all aircraft in the airspace. In addition to the primary radar, a secondary radar also provides information on aircraft. This radar sends out a transponder code request. Upon receiving the transponder identification request, the aircraft responds automatically, within a narrow and fixed time window. The return signal contains the aircraft beacon code and the

Key Strategy Structural Object Analysis
- Underline the nouns (continued) -

- Objects identified in this way can be put into one of the following categories
  - *Objects of interest*
  - Uninteresting objects
  - Attributes of objects
  - …

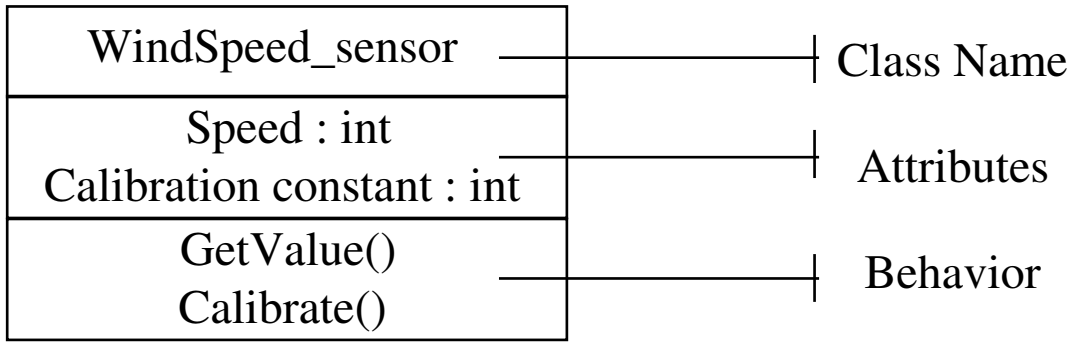
- Example of objects and attributes

Object	Attribute
Aircraft	Position
	Ground Speed
	Altitude
Flight plan	Point of departure
	Time of departure
	Point of arrival
	Time of arrival
	Average Ground speed
Transponder Code	Call sign
	Owning company

Class diagrams in UML

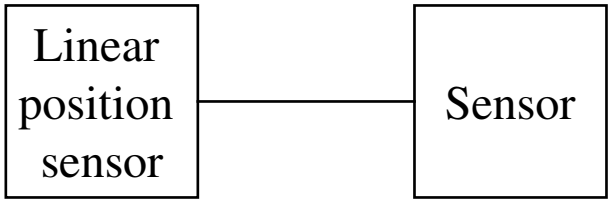
- UML classes are represented using rectangles with the name of the class inside
- A variation uses a three-segment box
  - The top segment holds the name of the class
  - The middle segment contains a list of attributes
  - The bottom segment contains a list of operations

  Not all the attributes and operations need to be listed



Class diagrams in UML
– relations between class/objects –

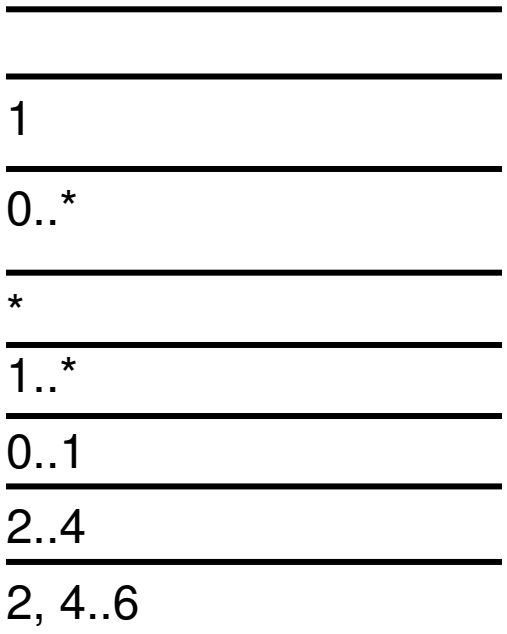
- *Associations* are relationships that typically manifest themselves at run-time to permit the exchange of messages between objects
  - It is shown using simple lines connecting two objects
  - Unless otherwise specified, associations are bidirectional and support messaging in either direction
- Examples



Class	Association	Class
Engine	has a	Piston
Flight computer	controls an	Engine
Linear position sensor	is a type of	Sensor

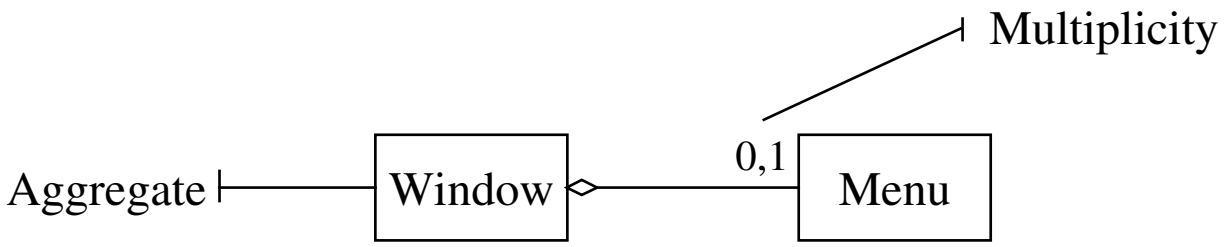
Class diagrams in UML
– Association: Multiplicity –

- Denotes the number of objects in the relationship
  - Unspecified
  - Exactly one
  - Zero or more (many, unlimited)
  - One or more
  - Zero or one
  - Specified range
  - Multiple, disjoint ranges



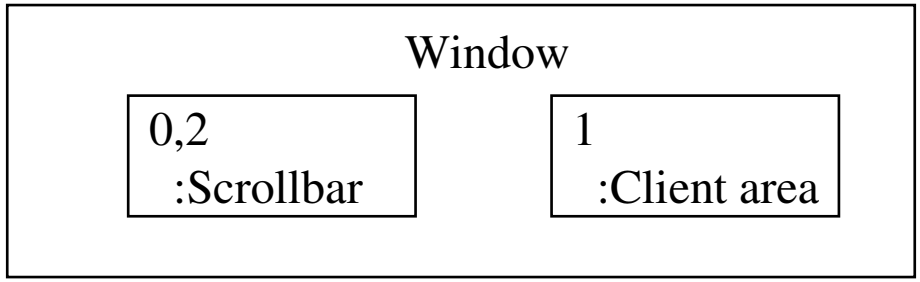
Class diagrams in UML
– relations between class and objects –

- *Aggregation* associations are used when one object logically or physically contains an other
  - It is shown with diamond at the owner (aggregate) end of the relationship



Class diagrams in UML
– relations between class and objects –

- *Composition* is a strong form of aggregation in which the owner is explicitly responsible for the creation and destruction of the part objects
  - It is shown either with filled-in aggregation diamonds or the physical containment of classes within the composite class
  - Note: components of composites can not be shared



Class diagrams in UML
– relations between class and objects –

- *Generalization* is the relationship used when one class is the specialization of another
  - It is an "is-a-kind-of" relationship among classes (e.g.: an infrared sensor is a kind of sensor)
  - It allows objects to be specified by difference rather than from scratch each time
  - Generalization is shown with directed lines with closed arrowheads

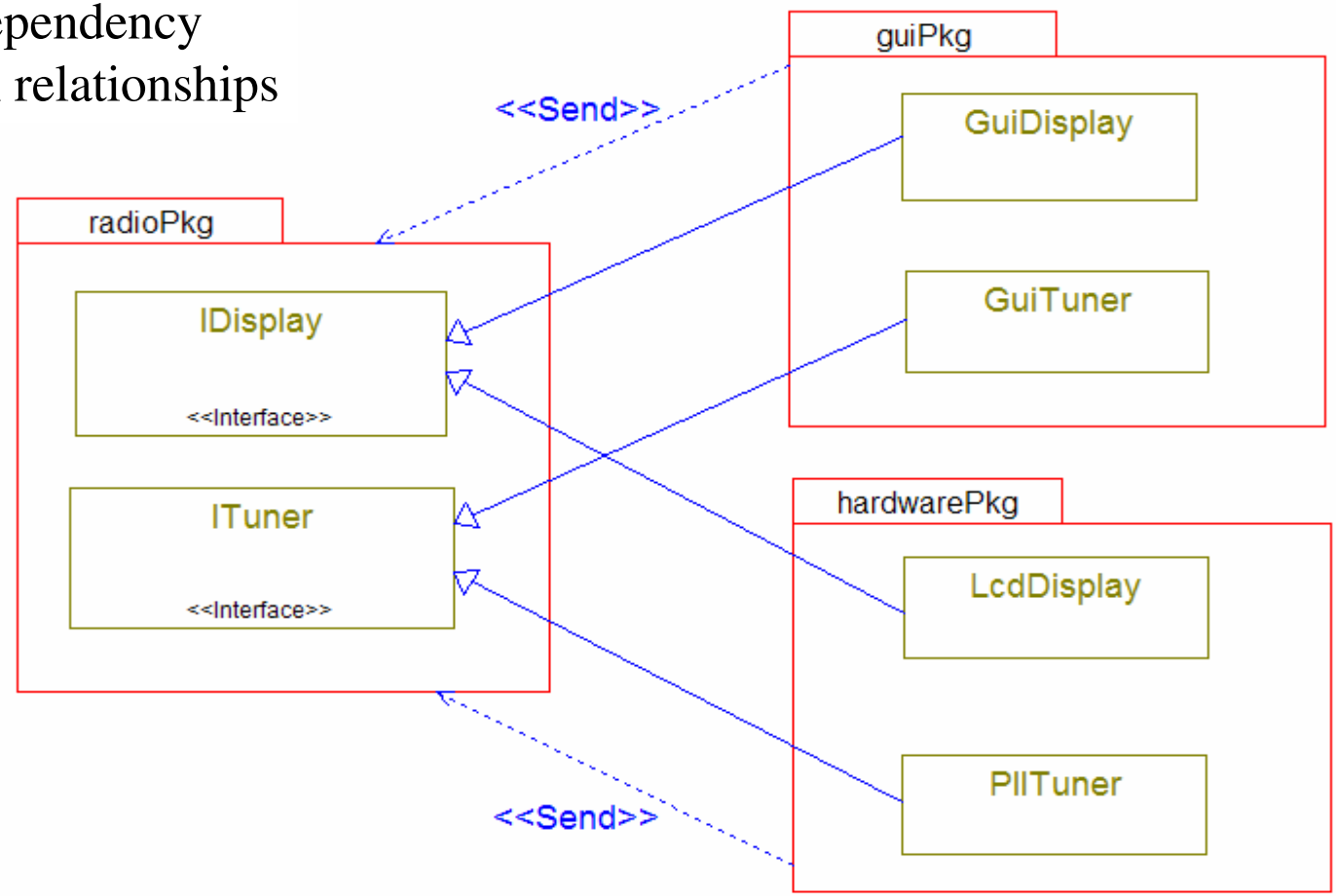
Class diagrams in UML
– relations between class and objects –

- *Dependency* is a relationship in which changes to one model element (the supplier) impact another model element (the client)
  - It is displayed in the diagram editor as a dashed line with an open arrow that points from the client model element to the supplier model element
  - Types of dependency

Type of dependency	Description
Abstraction	Relates two models that represent the same concept at different abstraction levels
Binding	Connects templates arguments to template parameters
Realization	Indicates that the client is an implementation of the supplier
Usage	Indicates that one model requires an other model for its full implantation

Class diagrams in UML
– relations between class and objects –

Example for dependency
and generalization relationships



Key Strategy Structural Object Analysis
- Identify causal agents and coherent services -

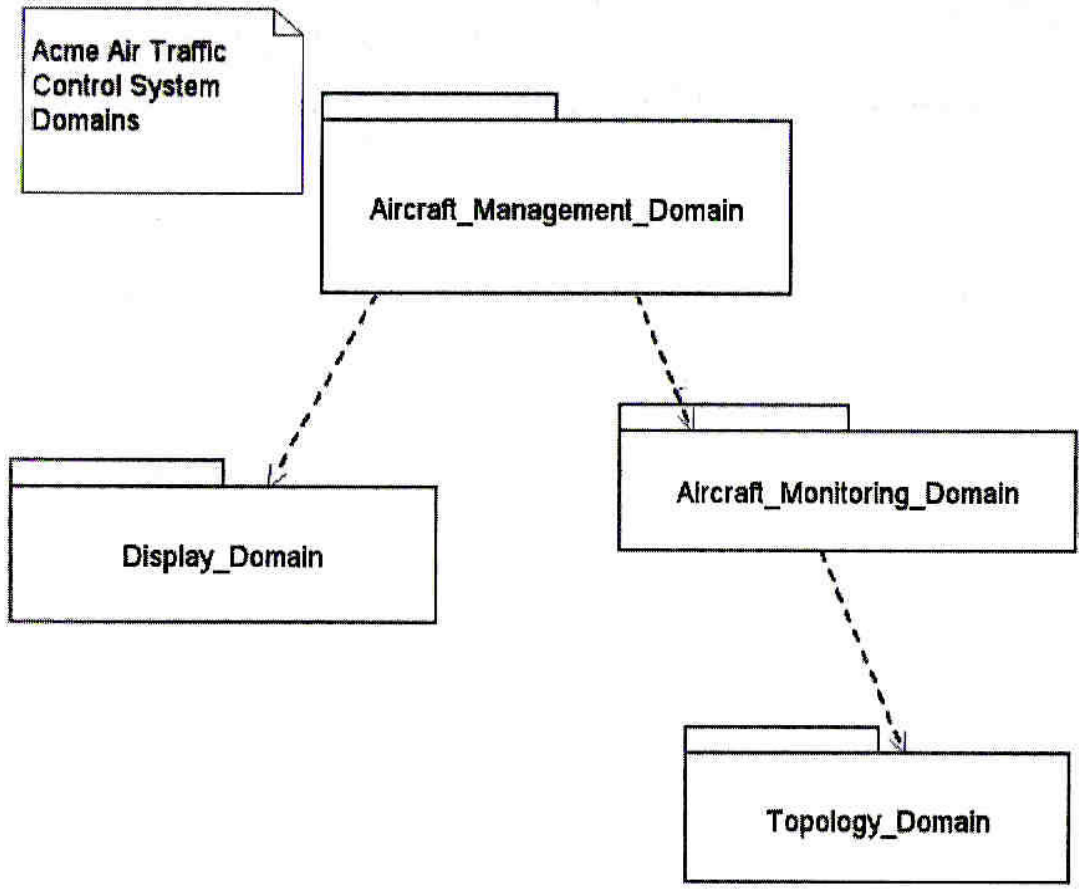
- Identify causal agents
  - Identify the sources of actions, events and messages
  - Includes the coordinators of actions
  - E.g. of AATC causal agents
    - Objects that reify the radars interfaces
    - Separation distance monitor object that scans identified aircraft for such violations, raising events when they are detected
- Identify coherent services
  - Coherent services are set of operations that seem to be intrinsically bound
    - E.g. operations
      - on complex number, services provided by the same object or that must execute in a specific order
  - Passive objects (server objects) provider objects are usually less obvious then causal agents
    - E.g. of AATC passive objects
      - Controller display, menu …

Key Strategy Structural Object Analysis
- Identify real-world items and physical devices -

- Objects representing real-world items
  - Represent the interface to that items and translate operations in the executing object system into real-word causes and effects (e.g. setting actuator positions, enabling sensors …)
  - Represent a real-word resource that must be managed (e.g. runway or flight path)
  - Represent constraint of the system (e.g. a mountain to avoid)
- Real-word items in AATC system
  - Radars
  - Display
  - Aircraft
  - Air space
  - Surface topology
- Generally real-time systems controls and monitors internal and/or external physical devices
- The devices may provide services to the system

Key Strategy Structural Object Analysis
- Identify essential abstractions of domains -

- A domain is a coherent subject matter that may be modeled as a set of objects with various relations among them
- Domains provide a means of decomposing a potentially unmanageable space into a smaller more manageable chunks
- Examples
  - User interface domain
    - Objects: widow, scroll bar, cursor, icon, …
  - Communication domain
    - Objects: packet, frame, data format, message quality of service



Key Strategy Structural Object Analysis
- Identify transactions -

- Transactions are objects that arise from interactions of other objects that have a temporarily persistent existence
- Examples of transaction objects

Object 1	Object 2	Association	Transaction object
Controller	Actuator	Controls	Control message over bus
Customer	Account	Owns	Deposit Withdrawal Close account Open account
Customer	Store	Buy things at	Order Return

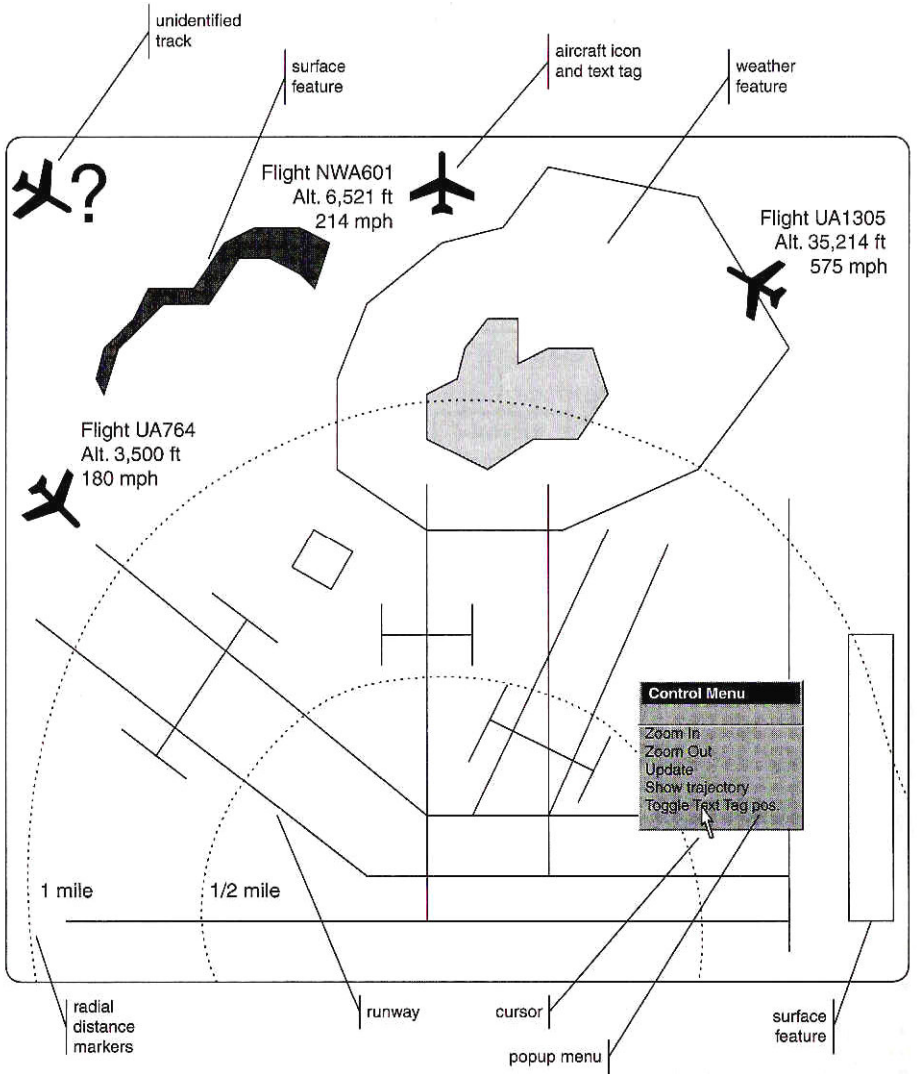
Key Strategy Structural Object Analysis
- Identify persistent information -

- Persistent information is typically held within passive objects (e.g. memories, stacks, queues, databases …)
- Example of persistent information

Information	Description
Task plans	Programs for a robotic system must be constructed, stored, recalled for editing, and recalled for execution
Security access	Stores valid users, their identifiers and passwords to permit different levels of access

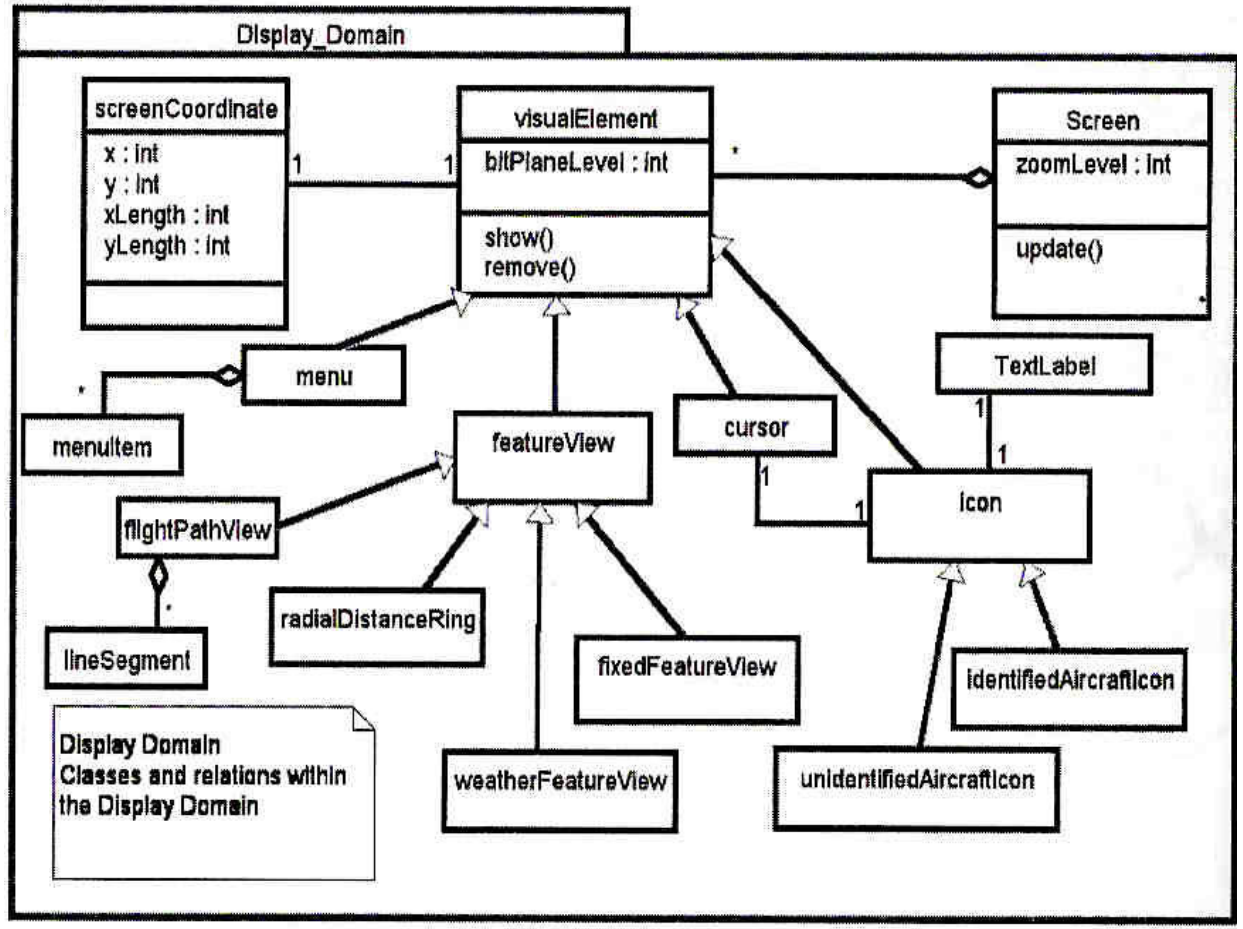
Key Strategy Structural Object Analysis
- Identify visual elements -

- Useful for real-time systems interacting directly or indirectly with human users
- Real-time displays may be
  - Blinking LED to indicate power status
  - GUI with windows, icons, …
  - AATC system display
    - Surface feature
    - Radial distance line
    - Weather feature
    - Cursor
    - Menu
    - Plane icon



Key Strategy Structural Object Analysis
- Identify visual elements (continued) -

- Class diagram for the ATTC display



Key Strategy Structural Object Analysis
- Identify control elements -

- Control elements are objects that control other objects
- They are specific types of causal agents
- Composites often orchestrate the behaviors of their tightly aggregated component objects
- ATTC system control elements
  - Button
  - Mouse
  - Separation distance manager
  - Runaway manager

Key Strategy Structural Object Analysis
- Identify Associations -

- The association relation is the glue that allows objects to send messages to one another and collaborate
- Object association strategies
  - Identify messages
    - Each message implies an association between the source and target objects
  - Identify message sources
    - The sensors that detect information/events and the creators of information/events are all message sources. They pass information on to other objects for handling and storage
  - Identify message storage depots
    - The depots have associations with message sources, as well as with the users of that information
  - Identify message handlers
    - They form connections to either or both message sources and storage depots
  - Execute scenarios on the object model
    - Walk through the scenarios, using the identified objects. When, in order to achieve the purpose of the collaboration, one message must send a message to an other → one message exist between them
- Note: Multiplicity, Aggregation and Composition have to be also considered

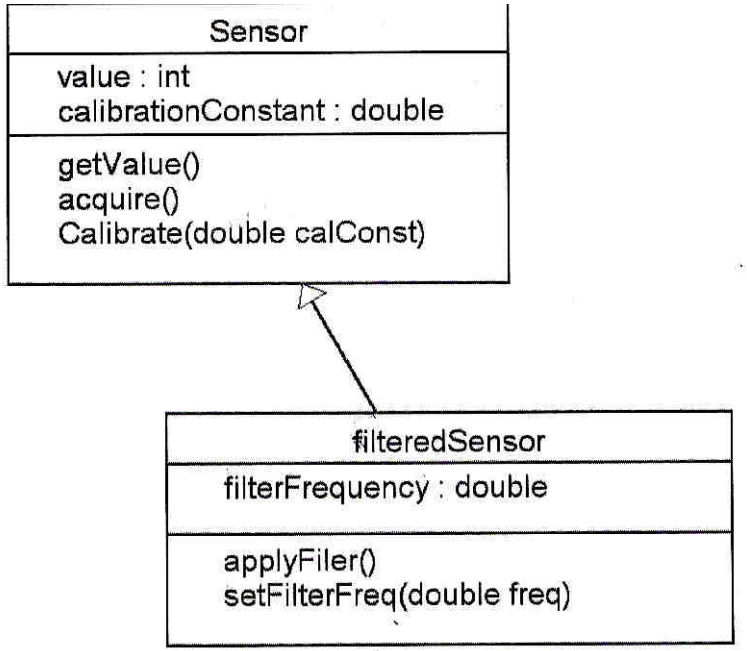
Key Strategy Structural Object Analysis
- Identify Associations (continued) -

- Low level implementations for associations
  - OS handles
  - OS messages posts
  - RPC
  - Simple pointers
    - Example

```
class dog
{
  Eyes * pE;   // low level implementation of association
public :      // these operations send a message to the Eyes object
  void BeHappy (void)          {pE-> Intensity(20)}
  void BeSad (void)              {pE-> Intensity(2)}
  void BeMelancholy (void) {pE-> Intensity(5)}
  …
}
```

Key Strategy Structural Object Analysis
- Object Attributes -

- Attributes
  - A named property of a type
  - Simple primitives – behavior no more interesting than get_ and set_ operations
  - Data portion of an object



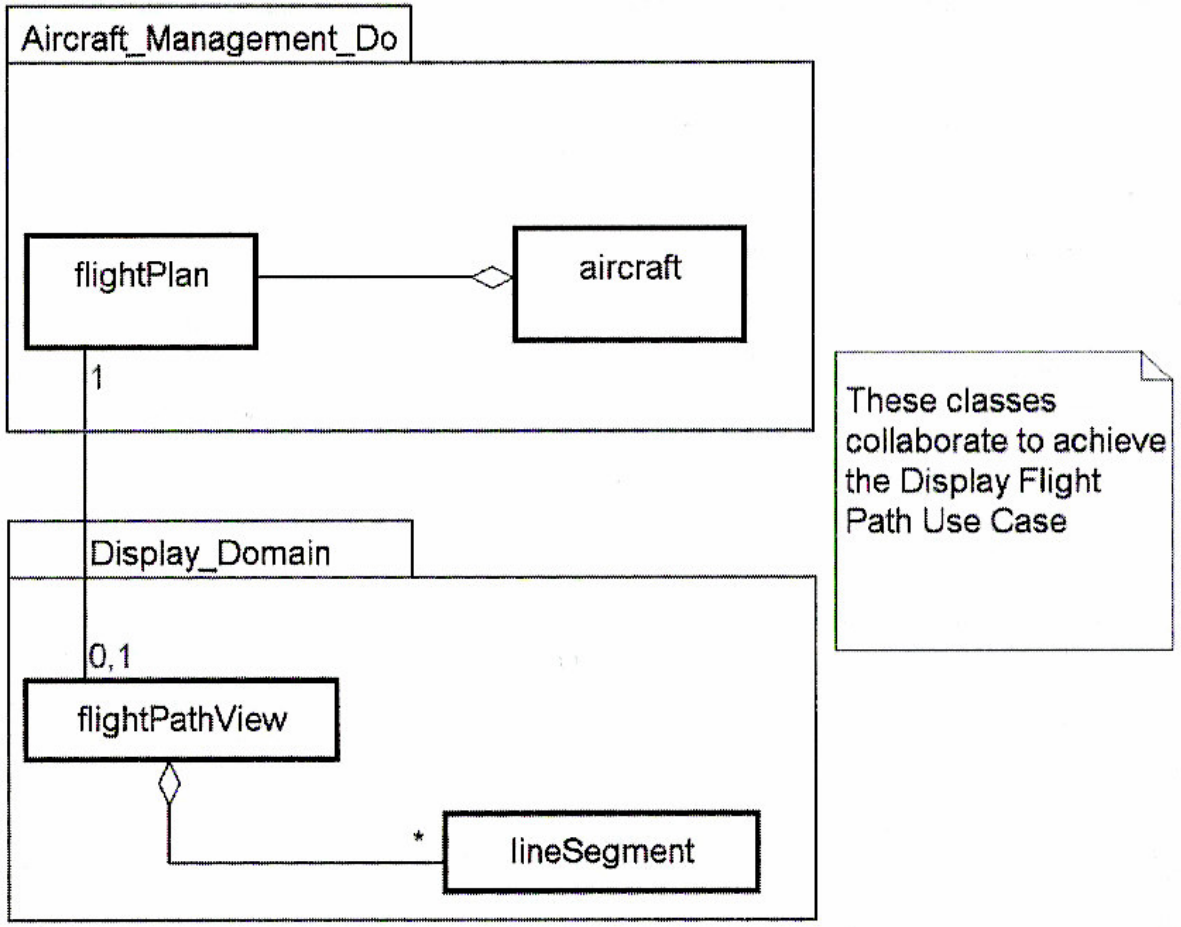
Key Strategy Structural Object Analysis
- Object Attributes (continued) -

- Key questions to identify the most important attributes of objects
  - What information defines the object ?
  - What information dot he object's operations act upon?
  - From the object view point, what do I know?
  - Are the identified attributes rich in either structure or behavior? If so, are they objects rather than attributes?
  - What are the responsibilities of the object? What information is necessary to fulfill these responsibilities?

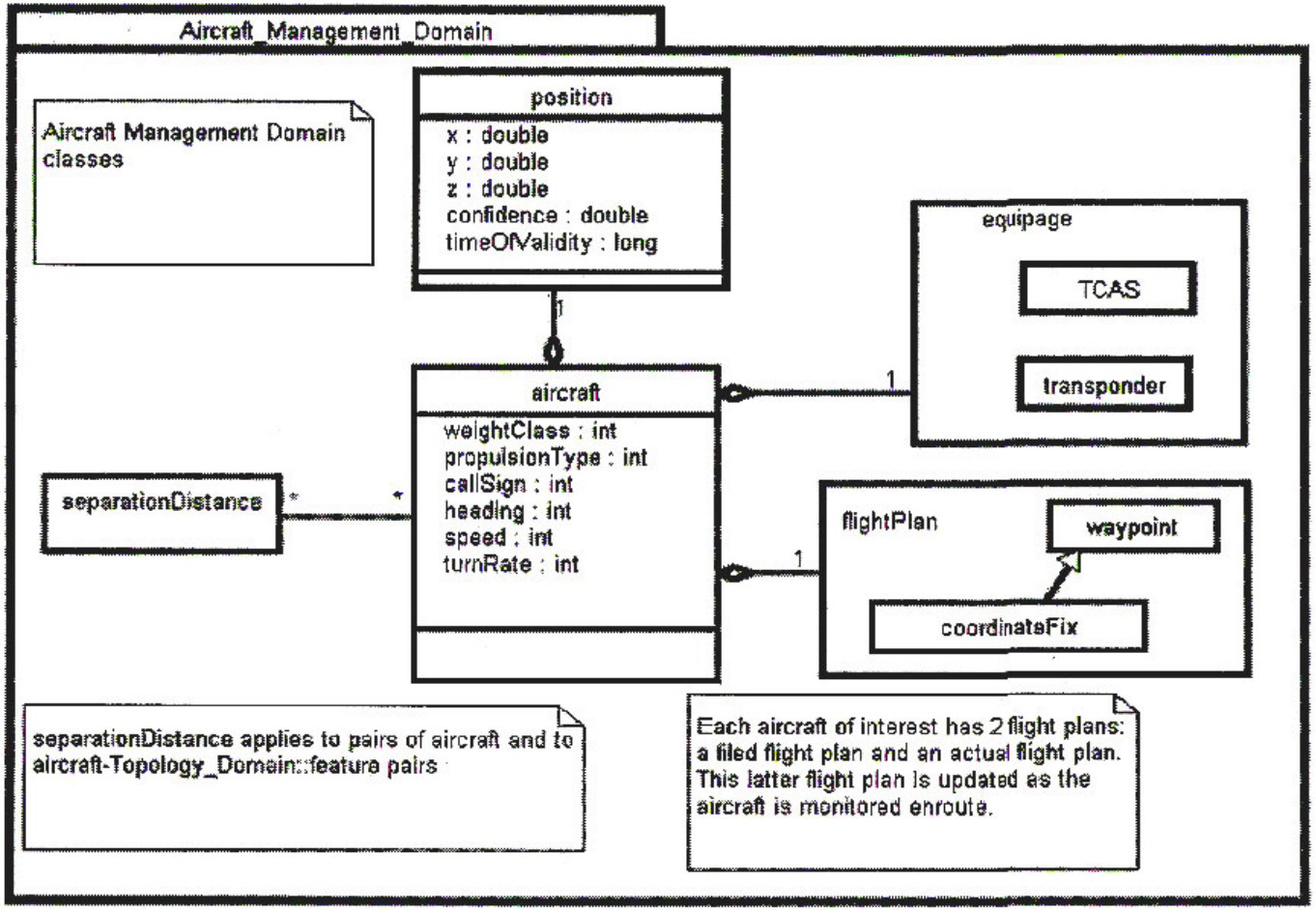
Key Strategy Structural Object Analysis
- Object Attributes (continued) -

- Answers for the *aircraft* object in AATC System
  - What information defines the object ?
    - Physical entity traversing the airspace. AATC models the aspects of the aircrafts that allow it to know where it is, where it is going (flight plan), its capabilities (equipage), its identity (call signal) and its current direction.
  - What information dot he object's operations act upon?
    - The aircraft must maintain its planned flight plan and create its actual flight plan during the course of a flight
  - From the object view point, what do I know?
    - Who I am, where I am, where I am going and what I am doing right now
  - Are the identified attributes rich in either structure or behavior? If so, are they objects rather than attributes?
    - Flight plan, position (x, y, z, t, confidence), equipage
  - What are the responsibilities of the object? What information is necessary to fulfill these responsibilities?
    - Primary responsibility – represent the location and direction of a physical aircraft to facilitate and support the controller's decisions about a flight path; runaway selection; take-off and landing clearances; to identify for the controller potential airspace and runway conflicts.

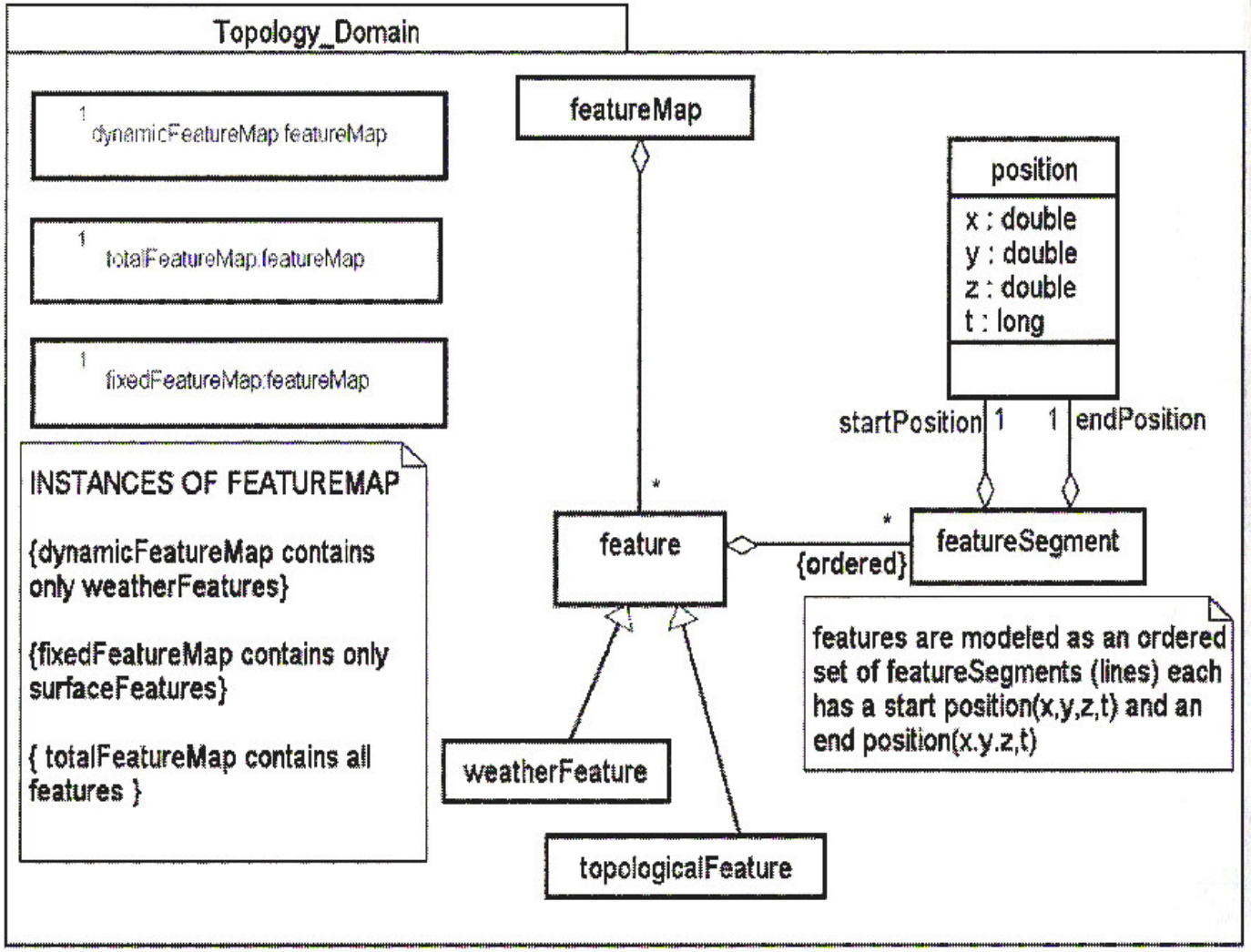
Key Strategy Structural Object Analysis
- Class Diagrams Examples (1/4) -



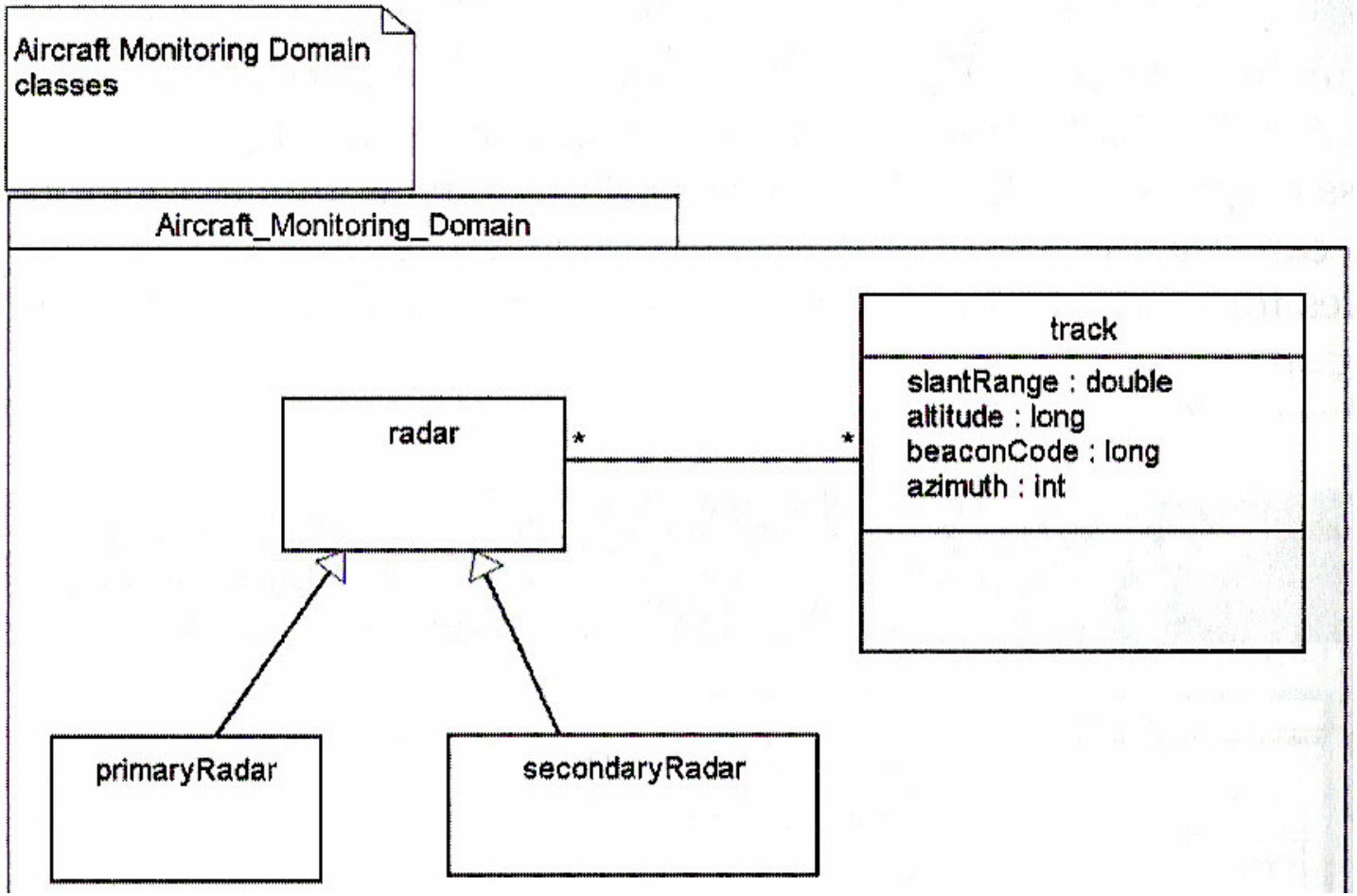
Key Strategy Structural Object Analysis
- Class Diagrams Examples (2/4) -



Key Strategy Structural Object Analysis
- Class Diagrams Examples (3/4) -



Key Strategy Structural Object Analysis
- Class Diagrams Examples (4/4) -



Key Strategy Structural Object Analysis - Heuristics for Good Class Diagrams -

- Each diagram should have a mission
- Build one diagram per use case
- Create one diagram per domain, including generalization hierarchies within that domain
- Create a single task diagram if there is a significant multi-tasking
- For rich hierarchies, create one diagram per generalization hierarchy
- Don’t mix generalization and collaboration unless it makes sense
- Don’t show attributes and operations on class diagrams

Key Strategy Structural Object Analysis
- Heuristics for Good Class Diagrams (continued) -

- Don’t cross lines
- Show multiplicity for all shown associations
- Show role names for associations
- Use containment composition notations
- Use composition to represent layers of abstraction within the class model
- Show constraints in note boxes where appropriate
- Show stereotypes where they add value
- Use line styles to convey information
- Use a note box to label each diagram (project, author, date and purpose)

Structural Objects Analysis
- Artifacts -

- Class and objects diagrams
  - Identify key abstractions and their logical organisation in packages and mechanisms
- Domain diagrams
  - An idiomatic usage of a class diagram, consisting primarily of packages organized around subject meters of interest and their dependency relations
- Collaboration diagrams
  - Same as sequence diagrams but visually organised similar to object classes

Outline

- General presentation for analysis stage
- Requirements Analysis
- Structural Object Analysis
- Behavioral Object Analysis

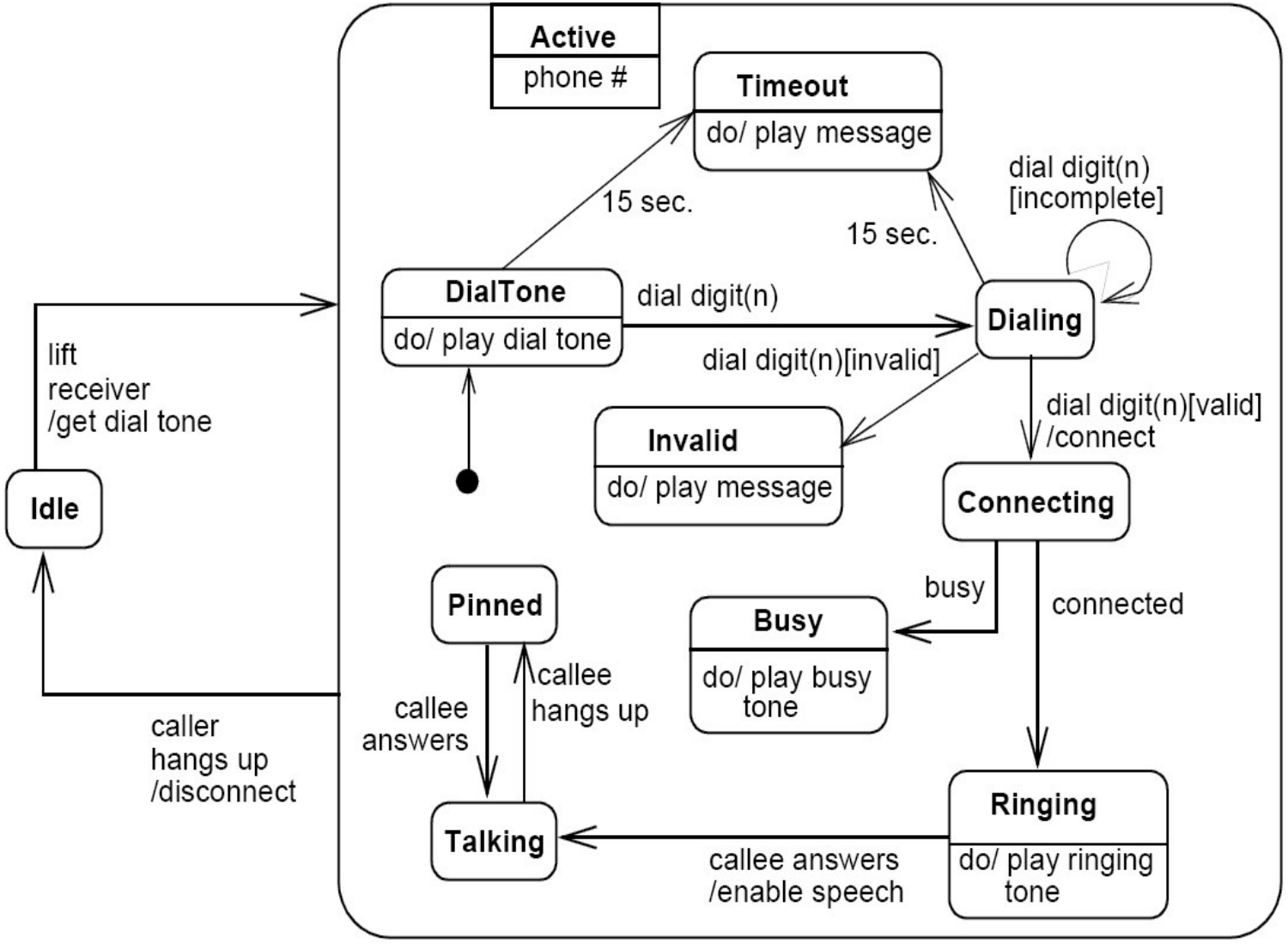
Behavioral Object Analysis

- The behavioral analysis cover the dynamic aspects of objects – the definition of behavior and the operations implementing this behavior
- Objects behavior
  - Simple behavior
  - State behavior
- Representations
    - State machines are used during this stage
      - State diagrams
      - Activity diagrams
    - Scenarios representations
      - Sequence diagrams
      - Timing diagrams

Behavioral Object Analysis
- State Diagrams -

- Shows the sequences of states that an object or an interaction goes through during its life in response to received stimuli, together with its responses and actions
- A state diagram is a graph of states and transitions. It is also a graph of states connected by physical containment and tiling
- The entire state diagram is attached to a class or a method (an operation implementation)

Behavioral Object Analysis
- State Diagrams example -



Behavioral Object Analysis
- State Diagrams -

- State
  - A condition during the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event. An object remains in a state for a finite (non-instantaneous) time.
- Sub-states
  - A state can be refined using *and*-relationships into concurrent substates or using *or*-relationships into mutually exclusive disjoint substates

Behavioral Object Analysis
- State Diagrams -

- Transition
  - Shows a path between states that indicates that a change of state is occurring.
  - The three parts of a transition (all are optional)
    - A trigger
      - An event that must occur for a transition to start.
    - A guard condition
      - Boolean condition that must be true for a transition to occur
    - An effect
      - An effect is an action or activity that the object performs when a guard condition is satisfied

Behavioral Object Analysis
- State Diagrams -

- Events
  - Transitions are initiated by events
  - UML defines four types of events

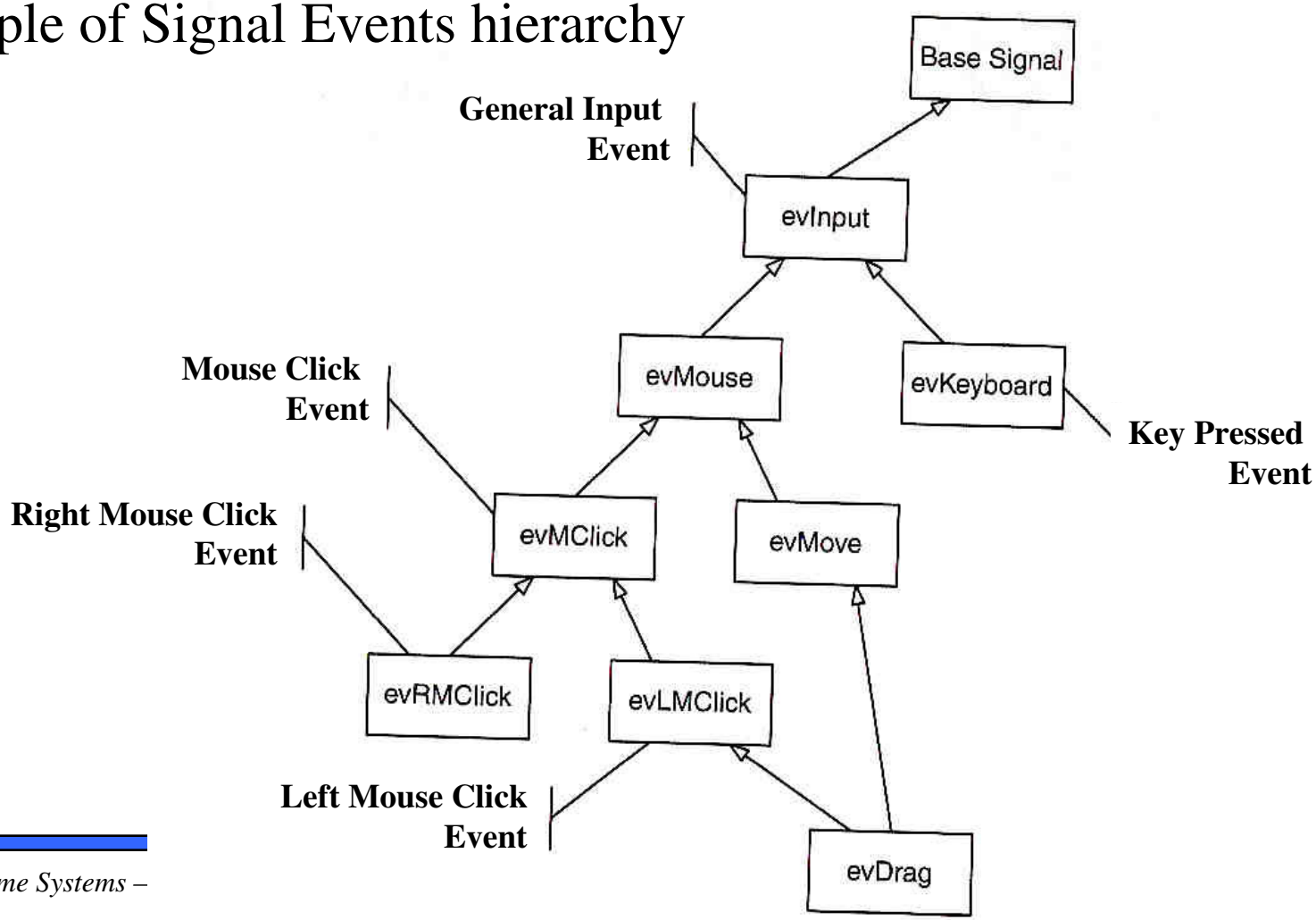
Type of event	Description
SignalEvent 	A specified message that, when received by an object, triggers a transition.
CallEvent 	An object receives a request to invoke an operation. The invocation of the operation triggers a transition.
ChangeEvent 	A Boolean condition is specified that triggers a transition when the condition is true.
TimeEvent 	A specified period of time that must pass or an absolute time that triggers a transition.

Behavioral Object Analysis
- State Diagrams -

- Signal Events hierarchy
  - Signals associated with SignalEvents are modeled with signal classes → signals hierarchies may be constructed
  - Useful for
    - Means to structure and classify the taxonomy of signals and their associated events
    - Polymorphic handling of events
      - Various objects can accept events at different levels
    - A class may perform different actions based on both general type of a system, as well as more specific type

Behavioral Object Analysis
- State Diagrams -

- Example of Signal Events hierarchy

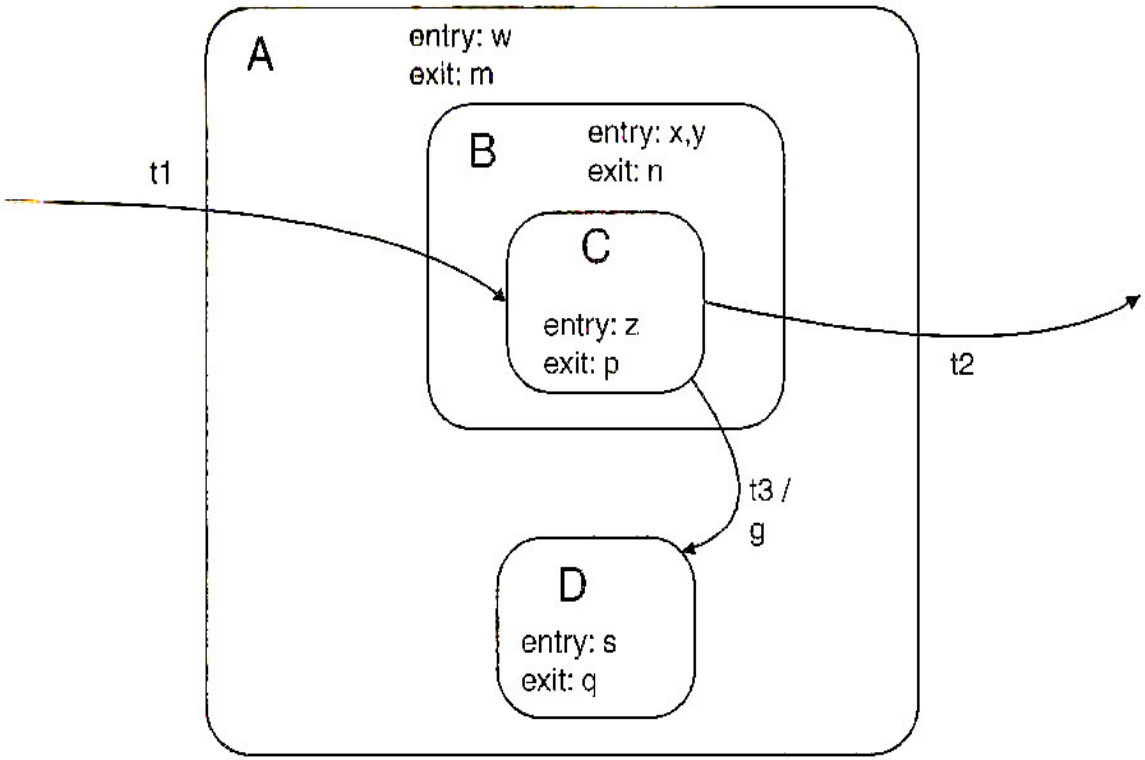


Behavioral Object Analysis
- State Diagrams -

- Actions and Activities
  - Actions
    - Noninterruptible behaviors executed at particular points in state machines or a sequence of actions that is noninterruptible
    - They are considered to take essentially zero time
    - Actions may be done on entry to a state, exit from a state, or when a transition is taken
    - Action Properties for states
      - Entry/[Action_list]
      - Exit/[Action_list]
      - Do/[Action_list]
      - …
  - Activities
    - Behaviors that can take significant time, so they must be generally interruptible
    - Activities may be done within states

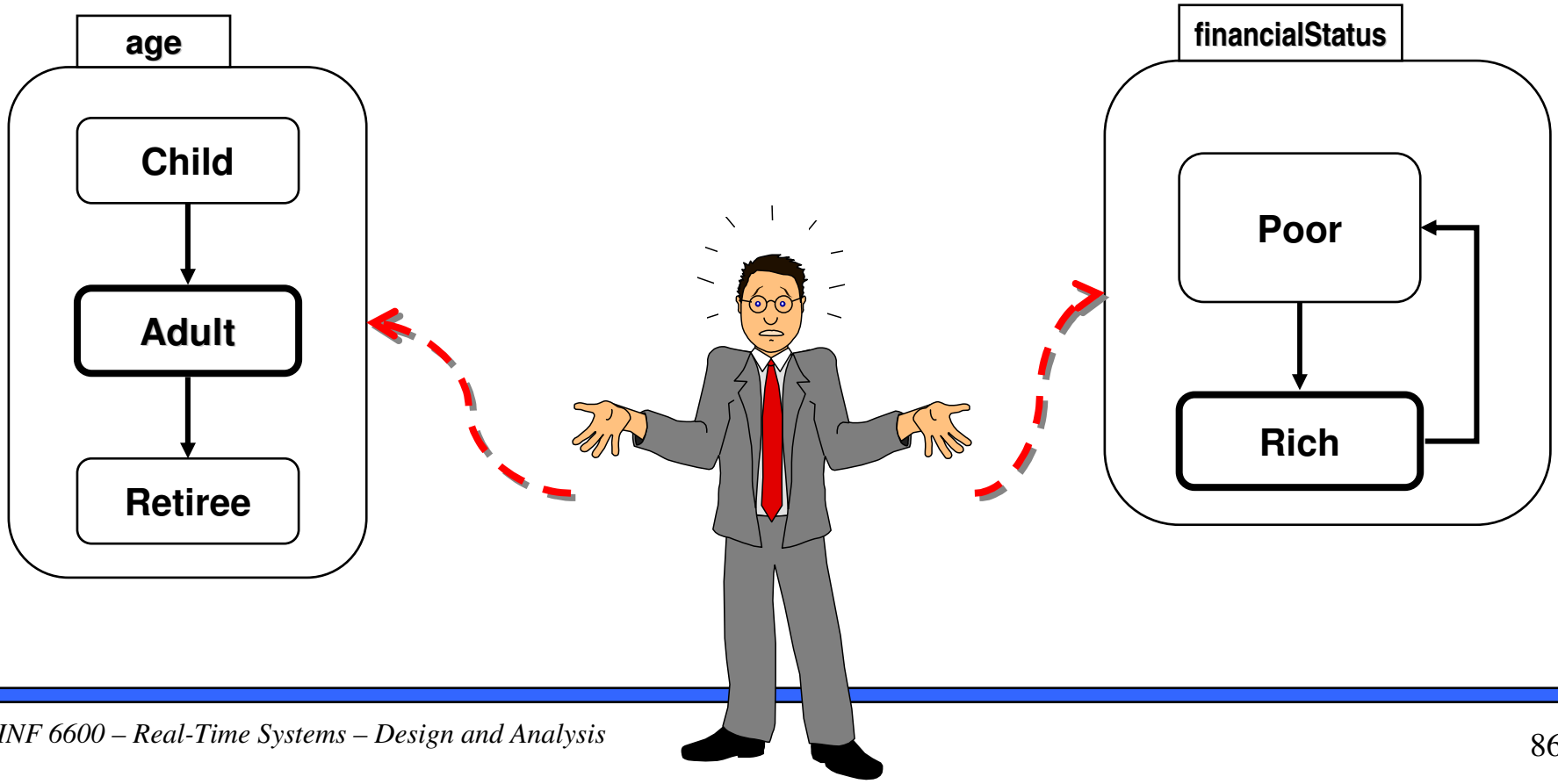
Behavioral Object Analysis
- State Diagrams -

- Actions and Nested States
  - T1 transition
    - w then x,y then z
  - T2 transition
    - p then n then m
  - T3 transition
    - p then g then s

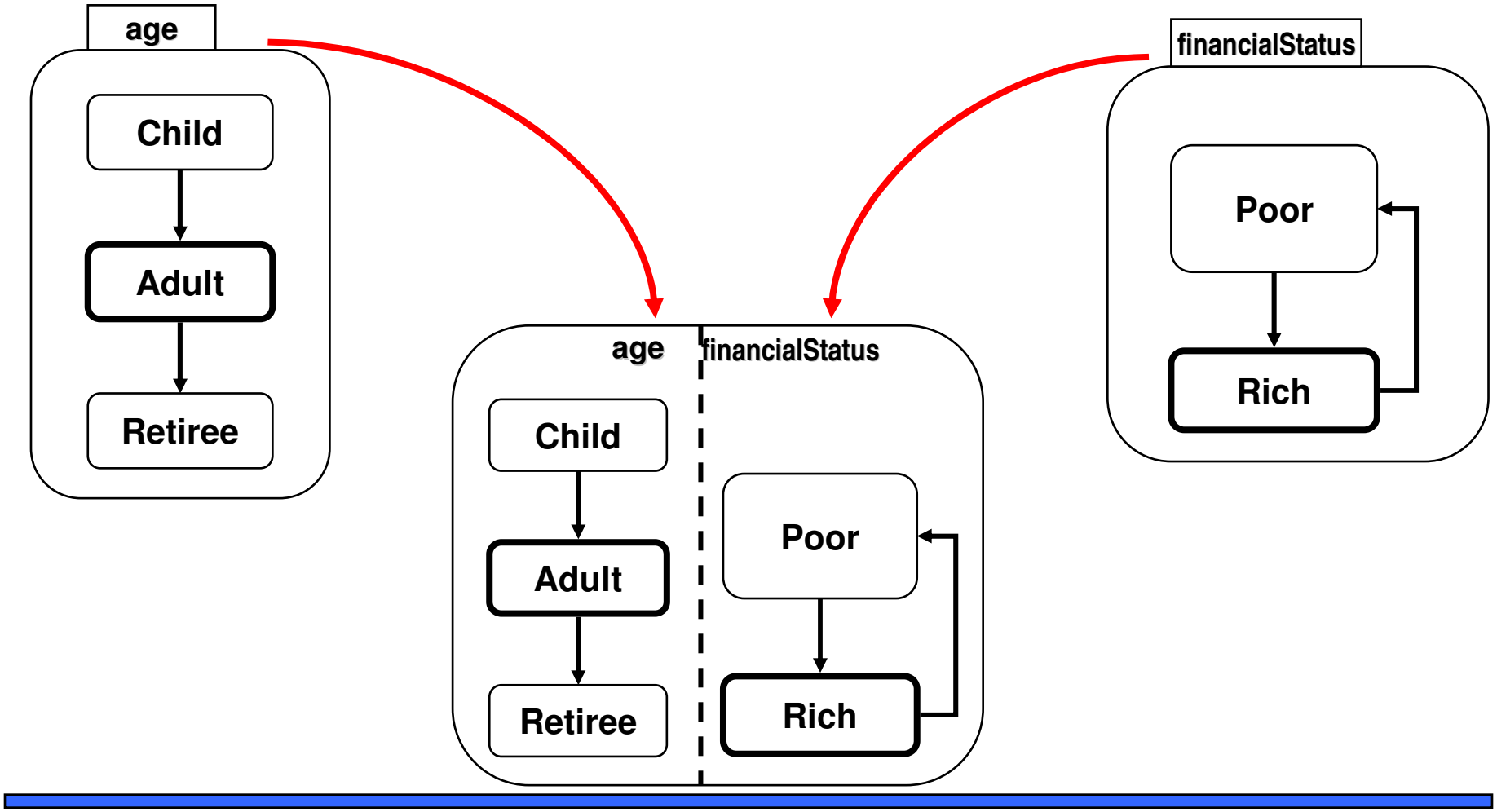


Behavioral Object Analysis
- State Diagrams -

- Orthogonal regions in State Diagrams
  - Multiple simultaneous perspectives on the same entity



Behavioral Object Analysis
- State Diagrams -



Behavioral Object Analysis
- State Diagrams -

- Pseudostates – UML entities that are almost, but quite states

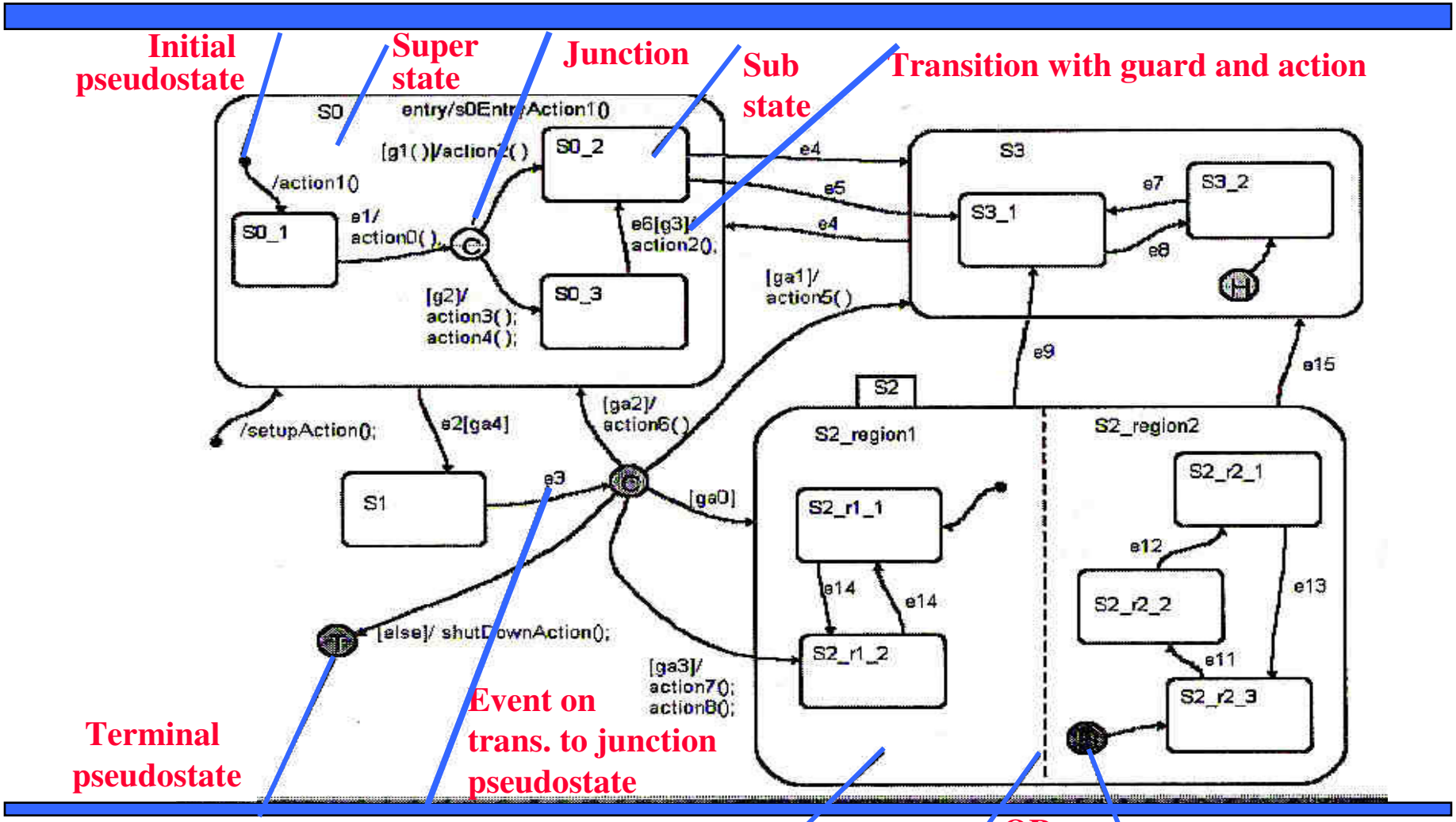
Pseudostate	**Description**
Initial	Placed within a region to identify the starting state
Choice	Receives a single incoming transition and outputs two transitions each with a guard condition, one of which is true.
Junction	Used between two or more fragments of a transition that each have a distinct entry or exit actions
Syncro	Used to synchronize two or more orthogonal regions
History	Used when the initial state within a state context is not fixed but depends instead upon the substate within that context that was occupied last. Two flavors : shallow and deep
Join	Receives two or more incoming transitions that meet to form one outgoing transition. Each incoming transition to the join pseudostate must originate from a different region of an orthogonal state

Behavioral Object Analysis
- State Diagrams -

- Pseudostates – UML entities that are almost, but quite states

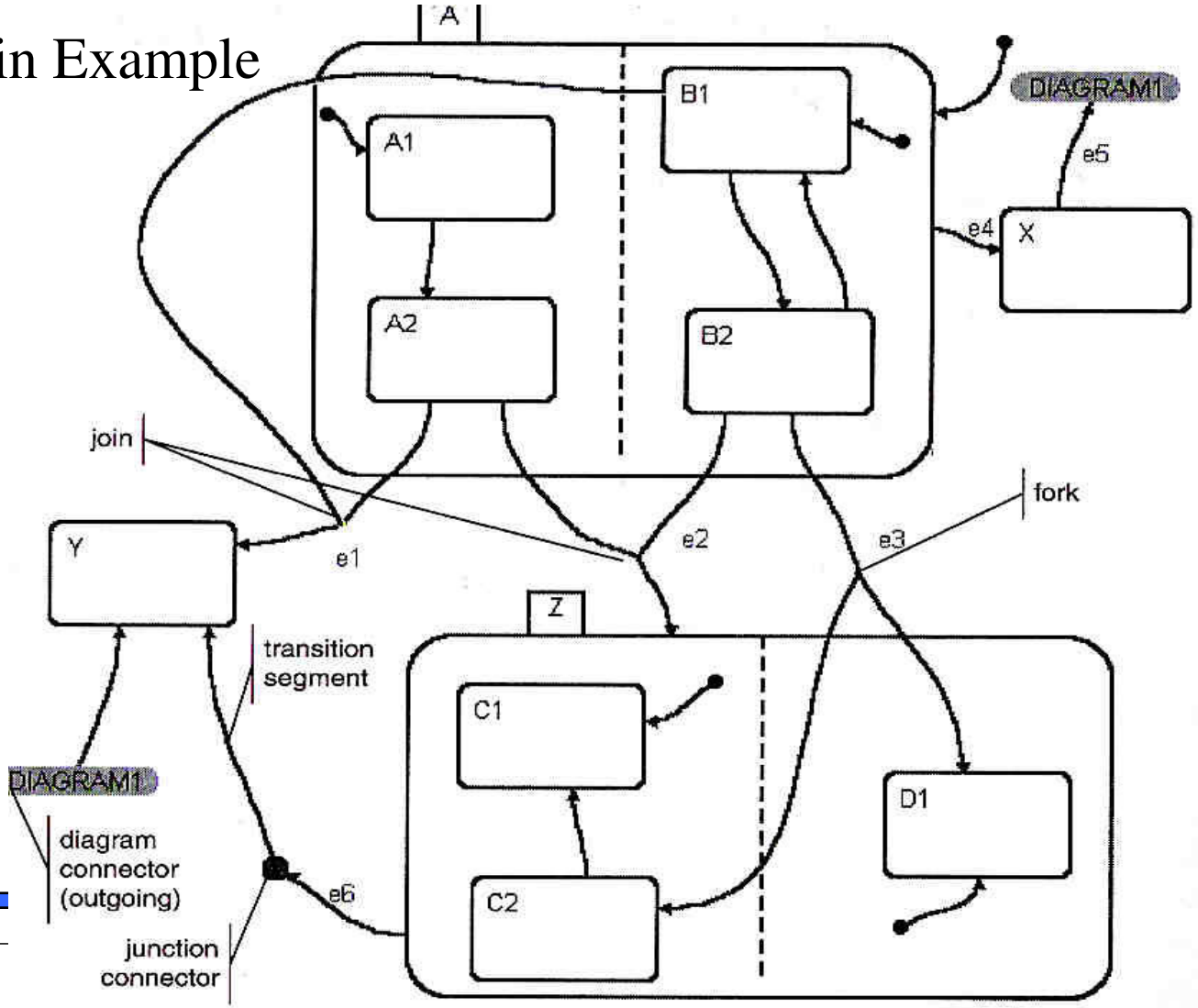
Pseudostate	**Description**
Fork	Receives one incoming transition that splits into two or more outgoing transitions. Each outgoing transition from the join pseudostate must target a state in a different region of an orthogonal state
Entry point	Placed on the boundary or in a region of a state machine or composite state with a single outgoing transition to a substate. Used when there are multiple ways to enter a state and there is no single default substate for the transition to target
Exit point	Placed on the boundary or in a region of a state machine or composite state with a single incoming transition from a substate. Used when there are multiple ways to exit a state. Each exit point can be the source of an external transition
Terminate	Identifies the end of the execution of a state machine

Behavioral Object Analysis
- State Diagrams -



Behavioral Object Analysis
- State Diagrams -

- Fork and Join Example

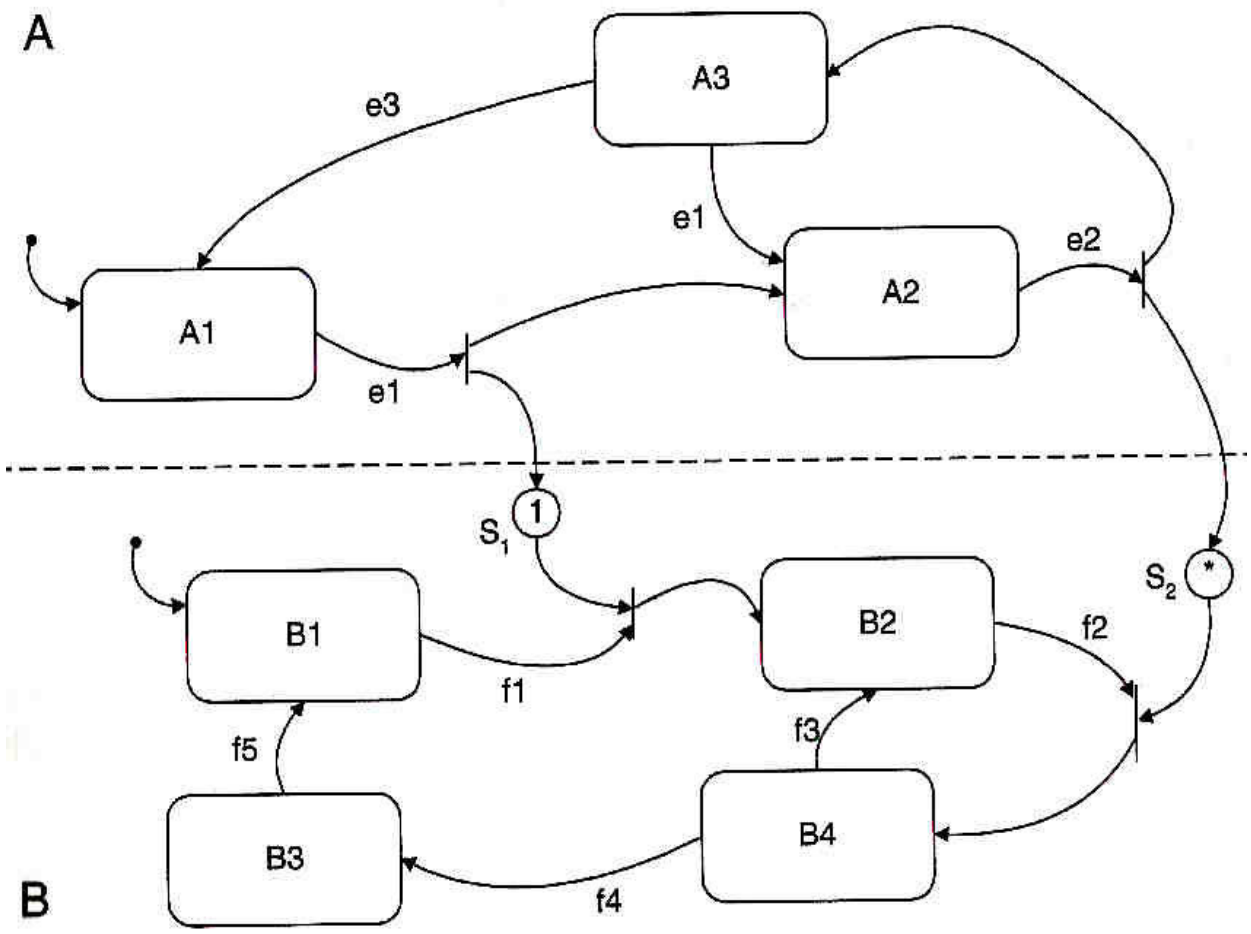


Behavioral Object Analysis
- State Diagrams -

- Synch States

  Examples

  – S1 allows the transition triggered by event f1to occur (in region B) if and only if the transition triggered by event e1has previously occurred
  – S1 – capacity 1
  – S2 – infinite capacity

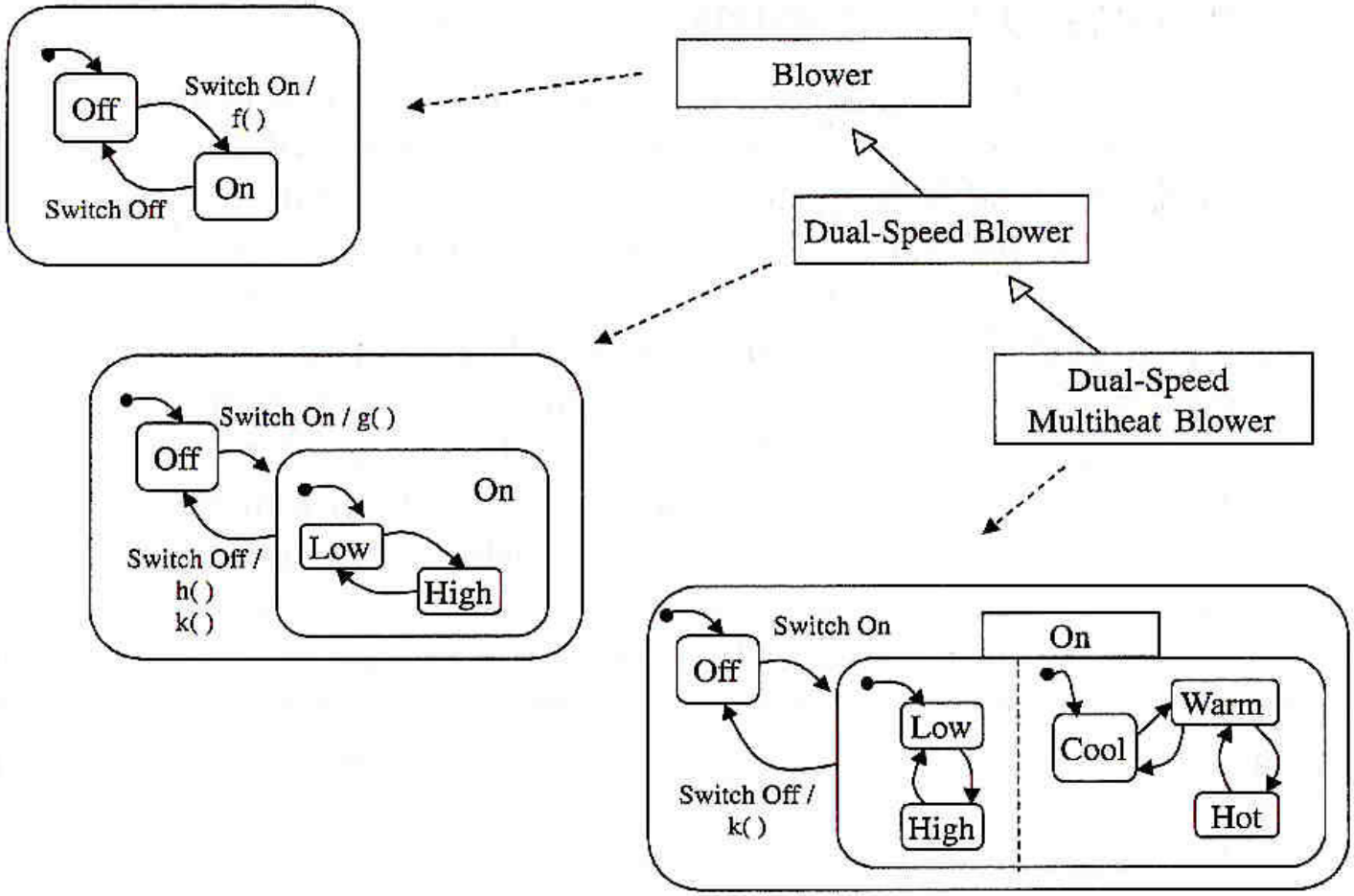


Behavioral Object Analysis
- State Diagrams -

- Inherited State Models
  - Approaches to support inheritance
    - Just ignore the parent class's parent and reconstruct the child's state model from scratch
      - The reuse advantage of object-oriented approach is not exploited
    - Inherit the parent's state model but specialize and extended it where necessary
      - New states and transitions may be added in the child class
      - States and transitions defined by the parent cannot be deleted
      - Actions and activity list may be changed
      - Actions and activities may be specialized in the subclasses
      - Substates may not alter their enclosing superstate, including adding a new superstate
      - Transition may be retagreted to different states
      - Orthogonal components may be added to inherit states

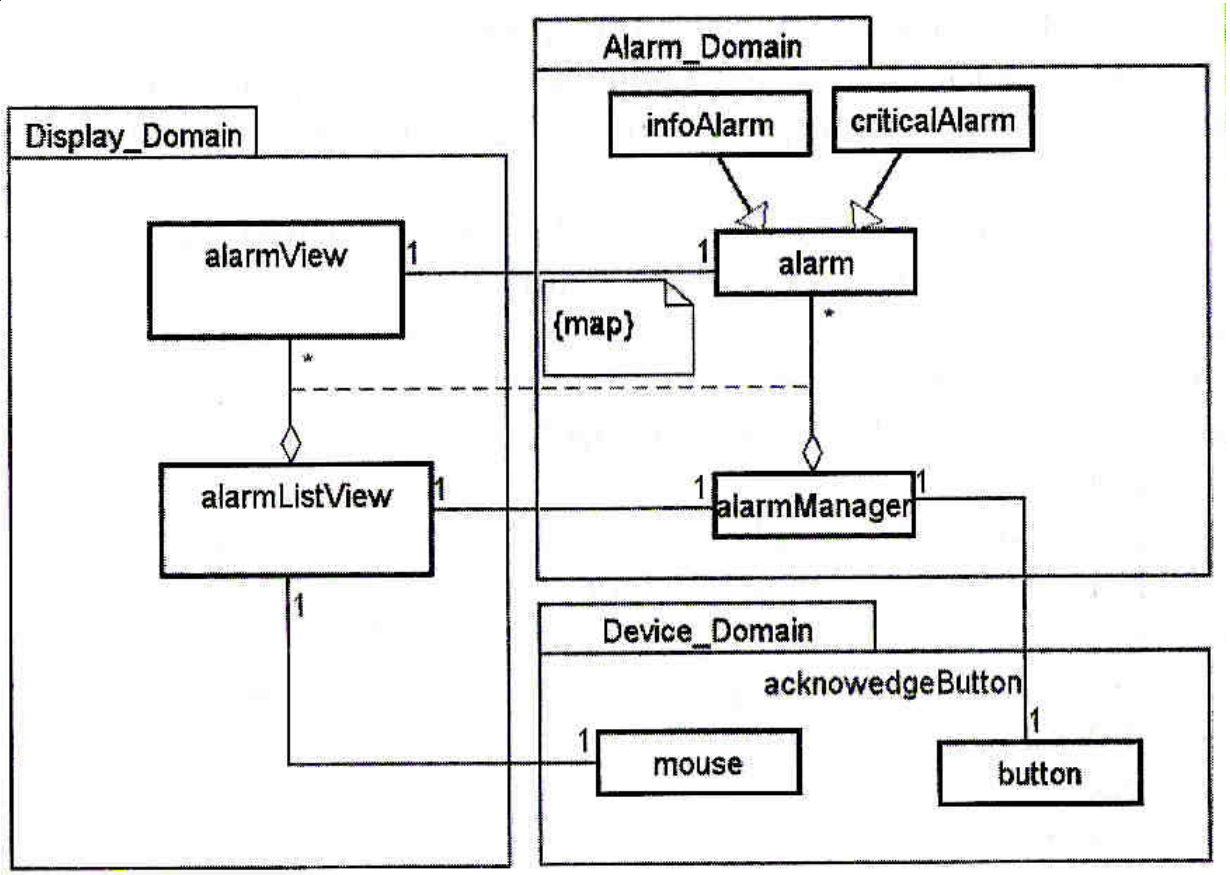
Behavioral Object Analysis
- State Diagrams -

- Inherited State Models Example



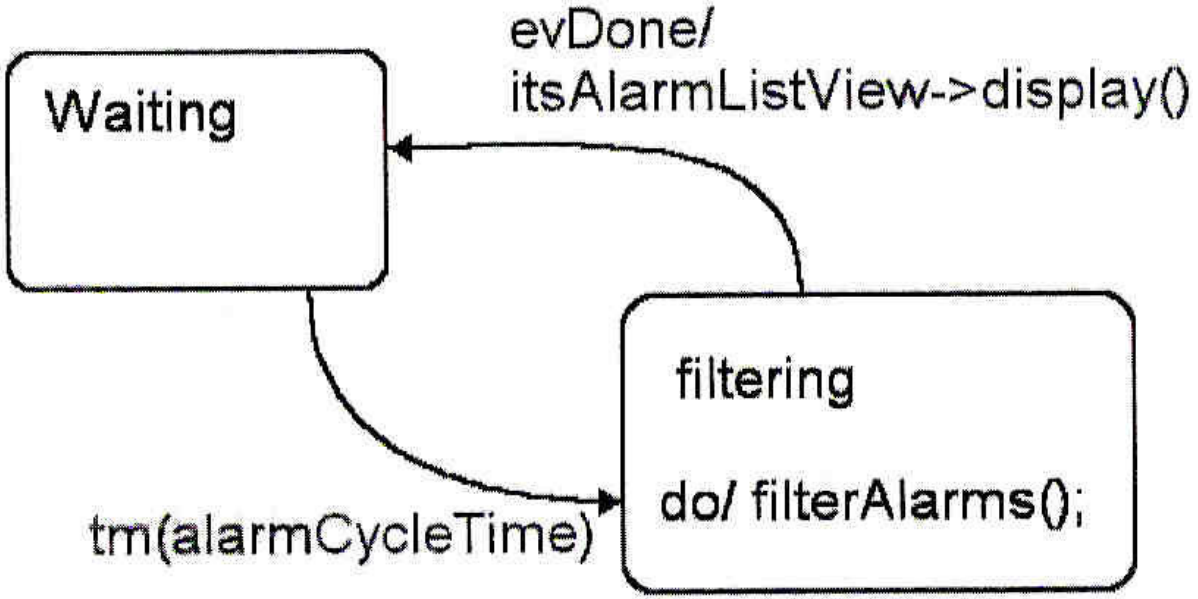
Behavioral Object Analysis
- Example: AATC Alarm System -

- Class Diagram



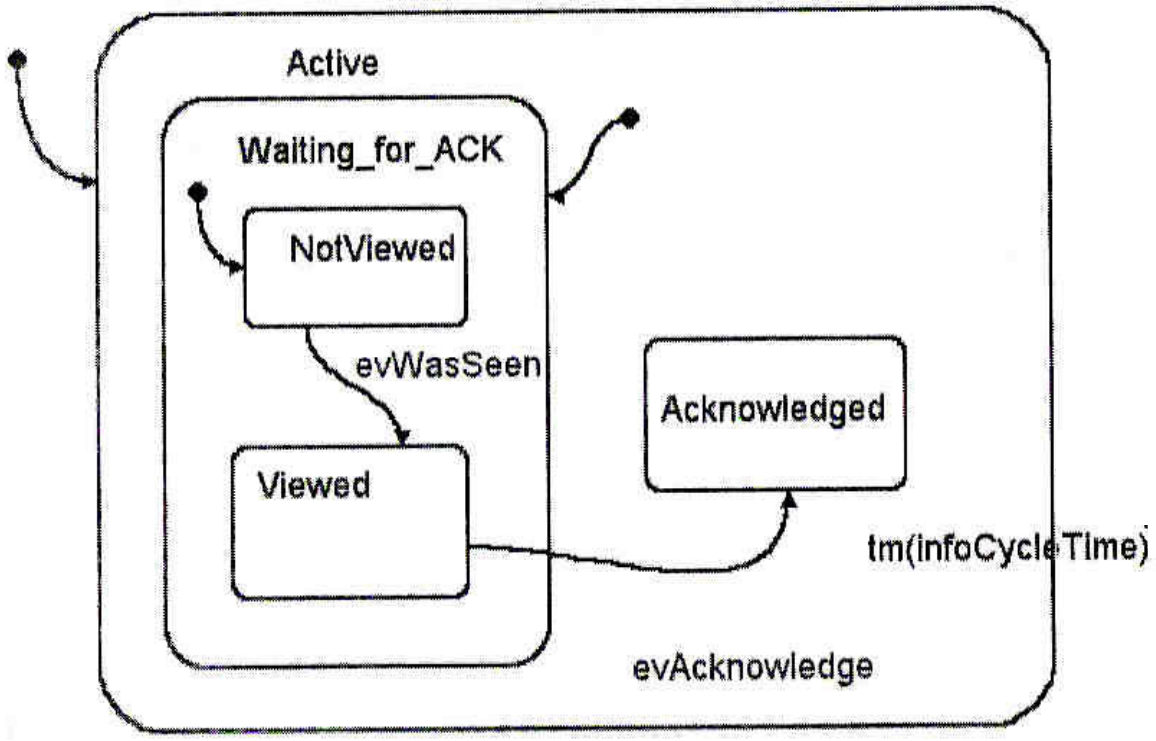
Behavioral Object Analysis
- Example: AATC Alarm System -

- AlarmManager State Diagram



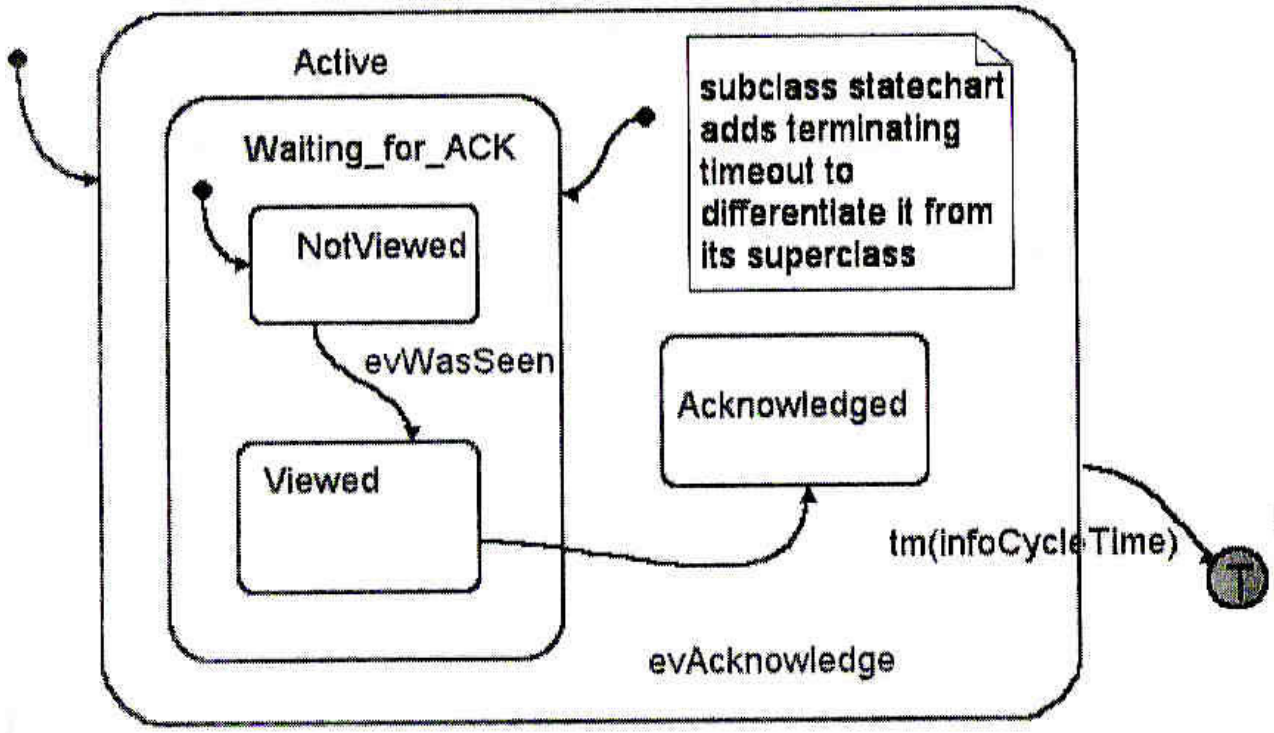
Behavioral Object Analysis
- Example: AATC Alarm System -

- Alarm State Diagram



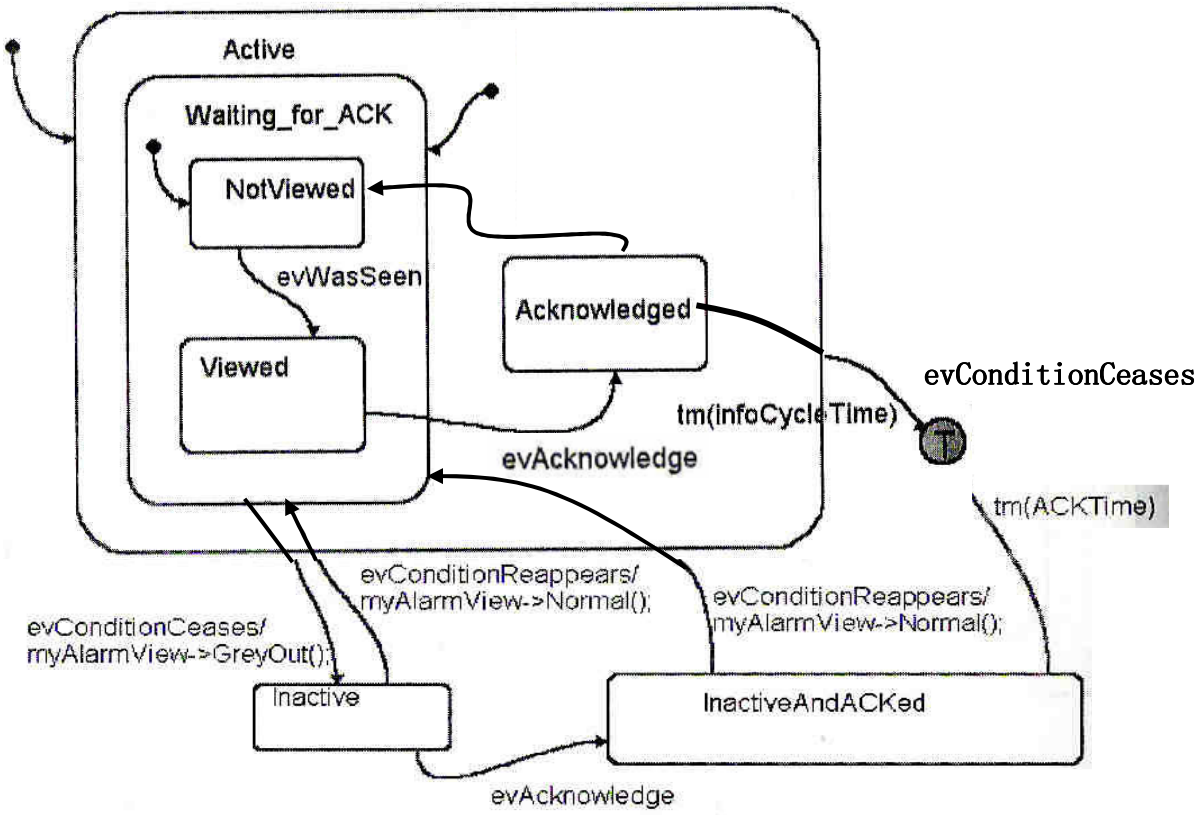
Behavioral Object Analysis
- Example: AATC Alarm System -

- infoAlarm State Diagram



Behavioral Object Analysis
- Example: AATC Alarm System -

- criticalAlarm State Diagram

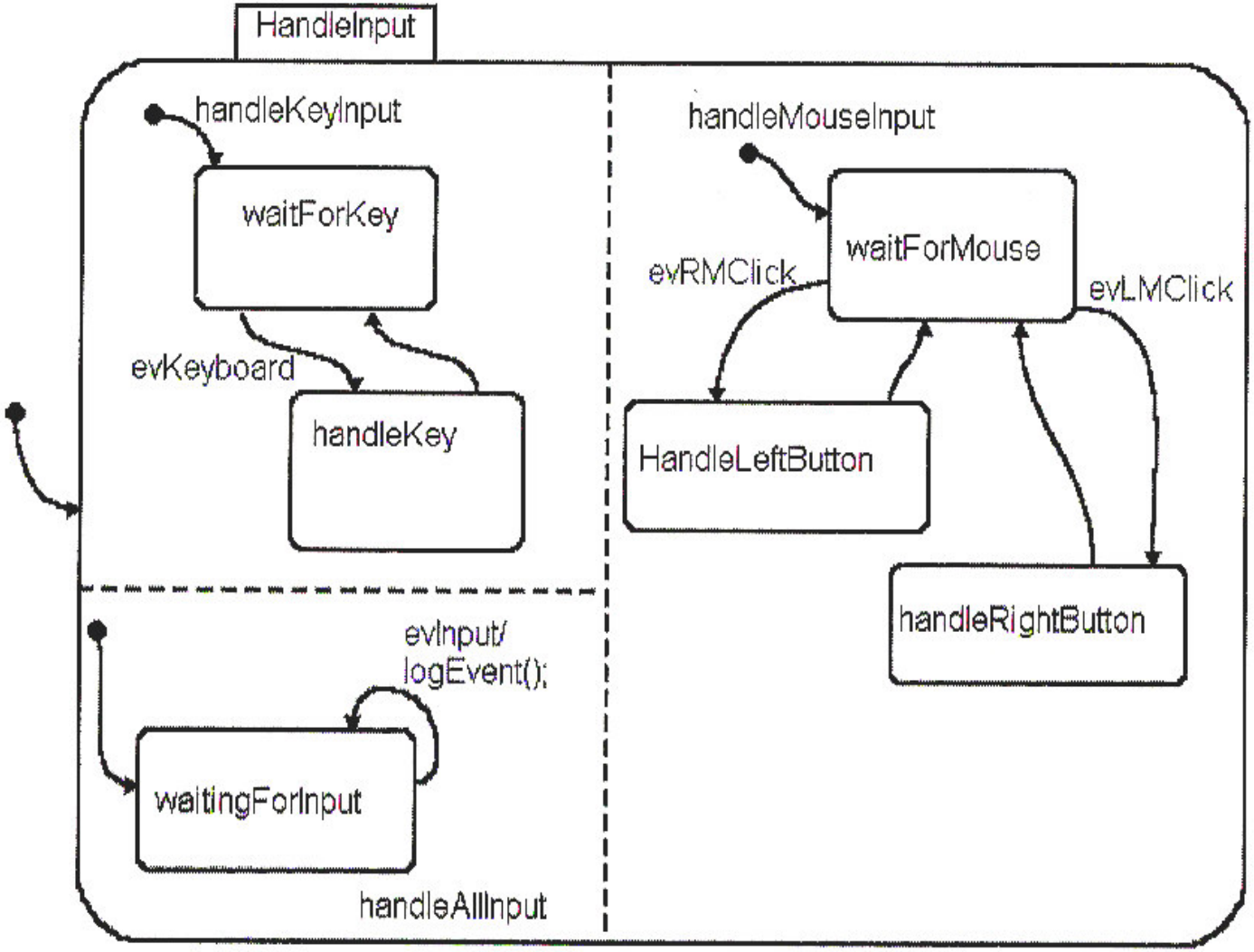


Behavioral Object Analysis
- Heuristics for State Diagrams -

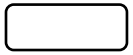
- Use states to hold behavior that take significant time (activities). Activities may take a long time to execute, so they are interruptible. They do not have to be attached to transitions, state entry or state exit
- Use actions on state entry or exist
- Exploit nested states and orthogonal regions
- When inherit state behavior do not
  - Remove transitions
  - Add a new superclass to on inhered one
  - Delete states
- When inherit state behavior you may
  - Add transitions
  - Change actions lists on transitions or states
  - Change activities within states
  - Use polymorphic operations

Behavioral Object Analysis
- Exercise -

- Explain the advantage of signals event hierarchy for the given state diagram



Behavioral Object Analysis
- Activity Diagrams -

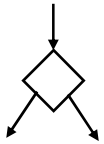
- An activity diagram is a special case of a state diagram in which all (or at least most) of the states are activities and in which all (or at least most) of the transitions are triggered by completion of the actions in the source states.
- The entire activity diagram is attached (through the model) to a class or to the implementation of an operation or a use case.
- The purpose of this diagram is to focus on flows driven by internal processing (as opposed to external events).
- Use activity diagrams in situations where all or most of the events represent the completion of internally-generated actions. Use ordinary state diagrams in situations where asynchronous events occur.
- Activities diagrams are composed of
  - Activities 
  - Transitions

Behavioral Object Analysis
- Activity Diagrams -

- Control Nodes
  - Initial Node, Fork and Join
  - Final flow

    - This node represents a point where one flow in an activity terminates, but does not affect the other flows in the activity
  - Final Activity

    - Represents a point where all flows in an activity terminate. An activity can have several activity final nodes. When the first activity final node is encountered, all flows stop and the activity terminates
  - Decisions

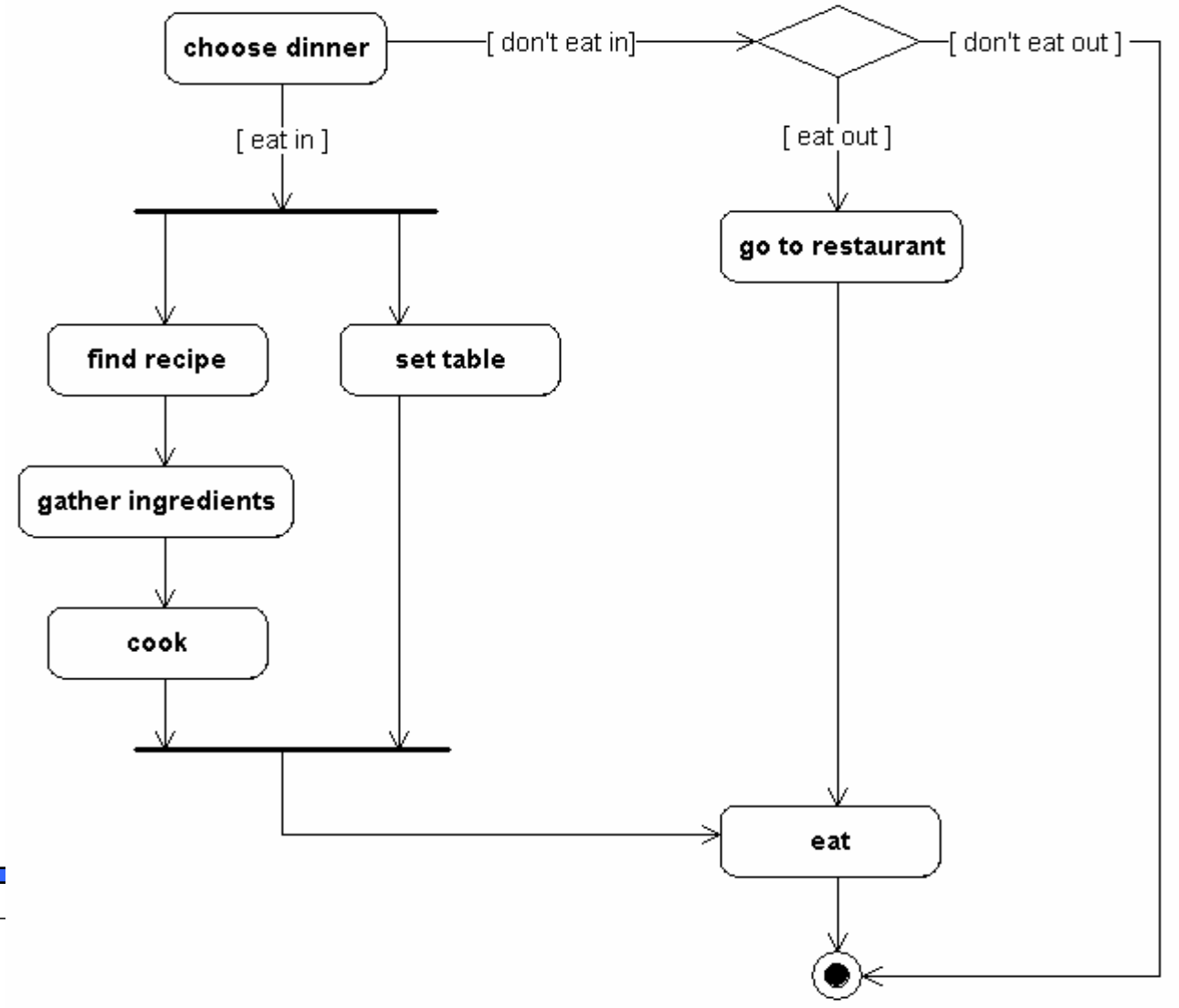


    - A single incoming edge branches into several outgoing edges. Typically constraints (also called guard conditions) are used on the outgoing edges to determine which edge should be followed
  - Merge

    - This node represents a point in an activity where several incoming edges come together into a single outgoing edge

Behavioral Object Analysis
- Activity Diagrams -

- Example



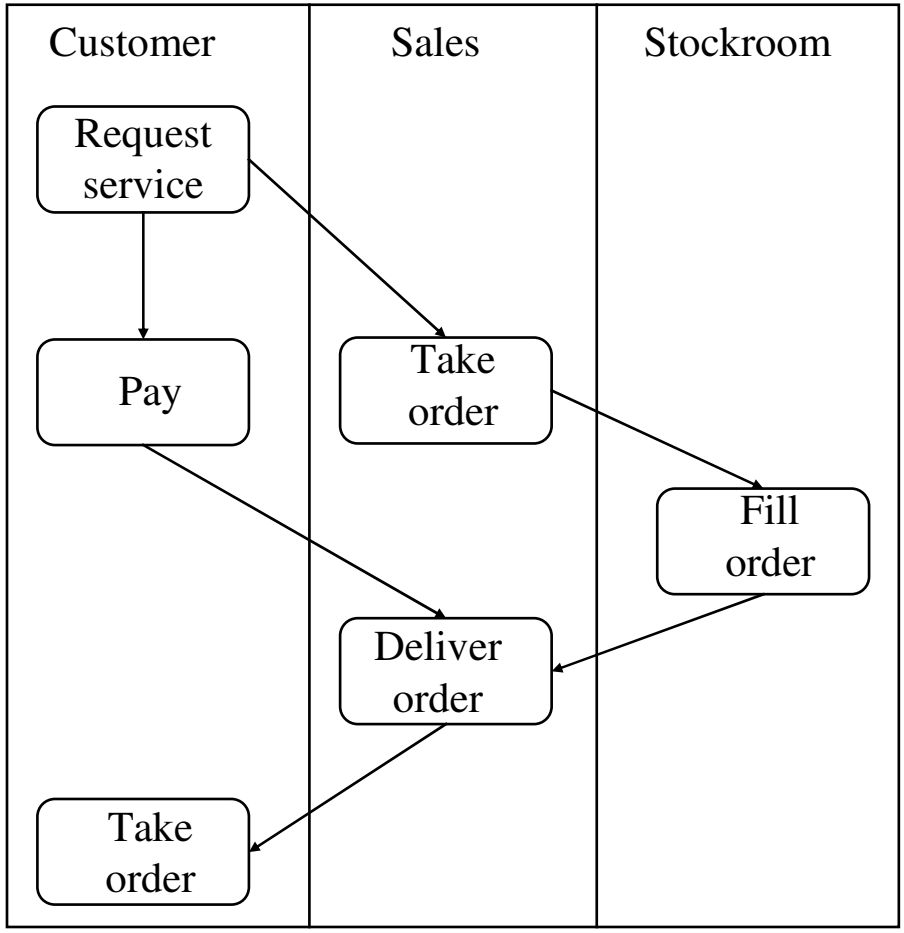
Behavioral Object Analysis
- Activity Diagrams -

- Call behaviors
  - type of action that can be used to reference behaviors in other activity, state machine, or interaction diagrams in a model.

Call behavior that references a behavior in an activity diagram	Call behavior that references a behavior in a state machine diagram	Call behavior that references a behavior in an interaction diagram
		

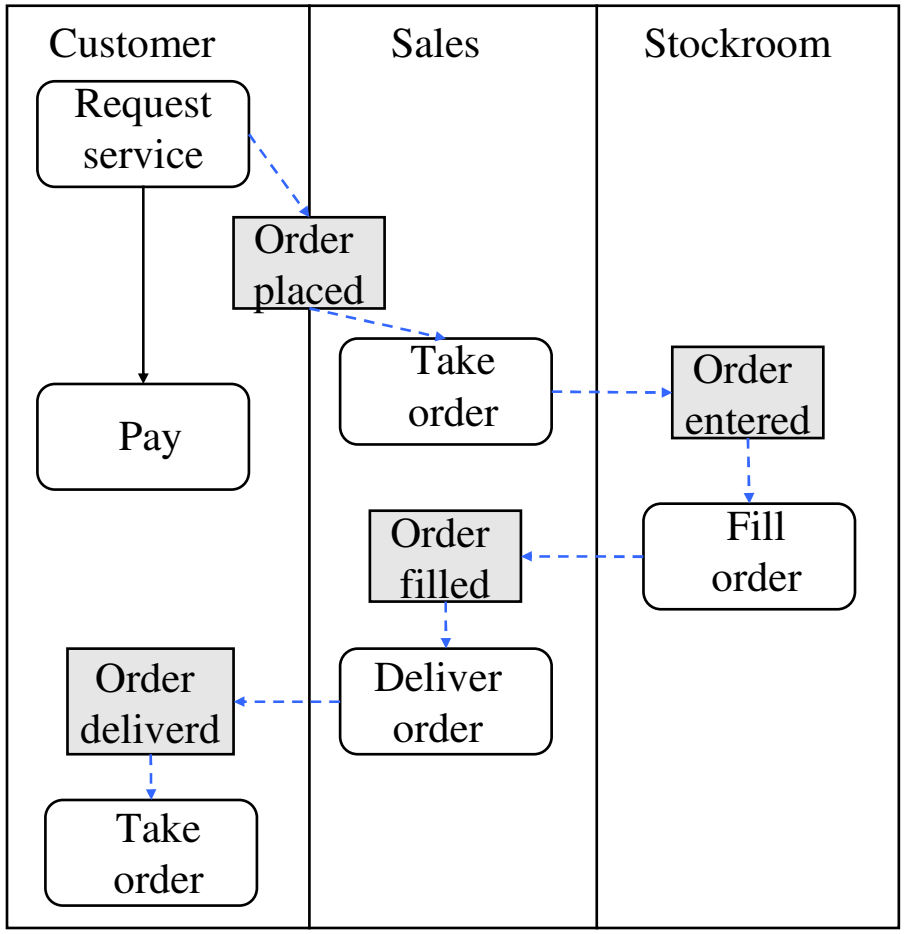
Behavioral Object Analysis
- Activity Diagrams -

- Swim lanes
  - A kind of package for organizing responsibility for activities within a class.



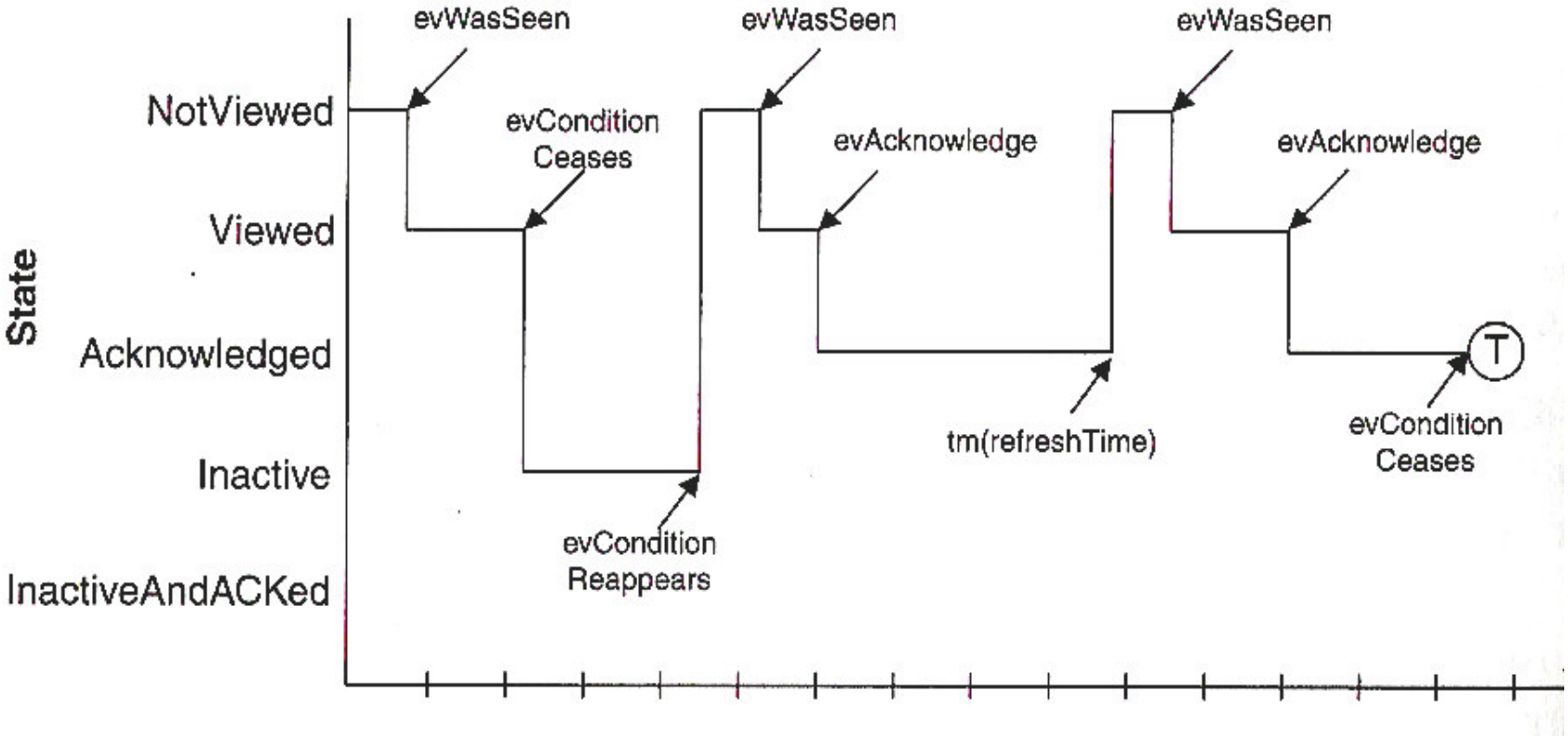
Behavioral Object Analysis
- Activity Diagrams -

- Action-Object Flow Relationship
  - Activities operate by and on objects. Two kinds of relationships can be shown
    - The kinds of objects that have primary responsibility for performing an action and
    - The objects whose values are used or determined by the action.



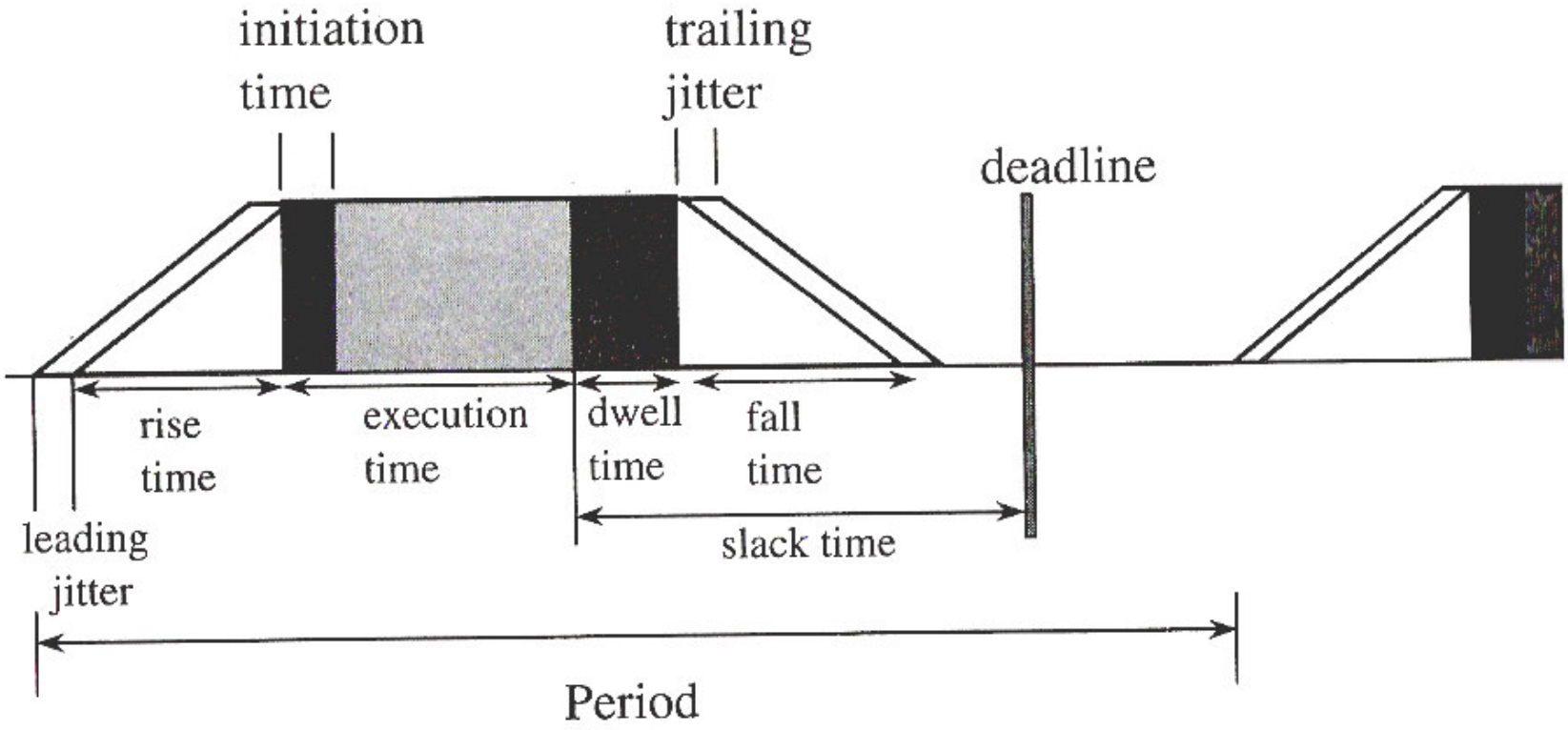
Behavioral Object Analysis
- Timing Diagrams -

- Simple Timing Diagram

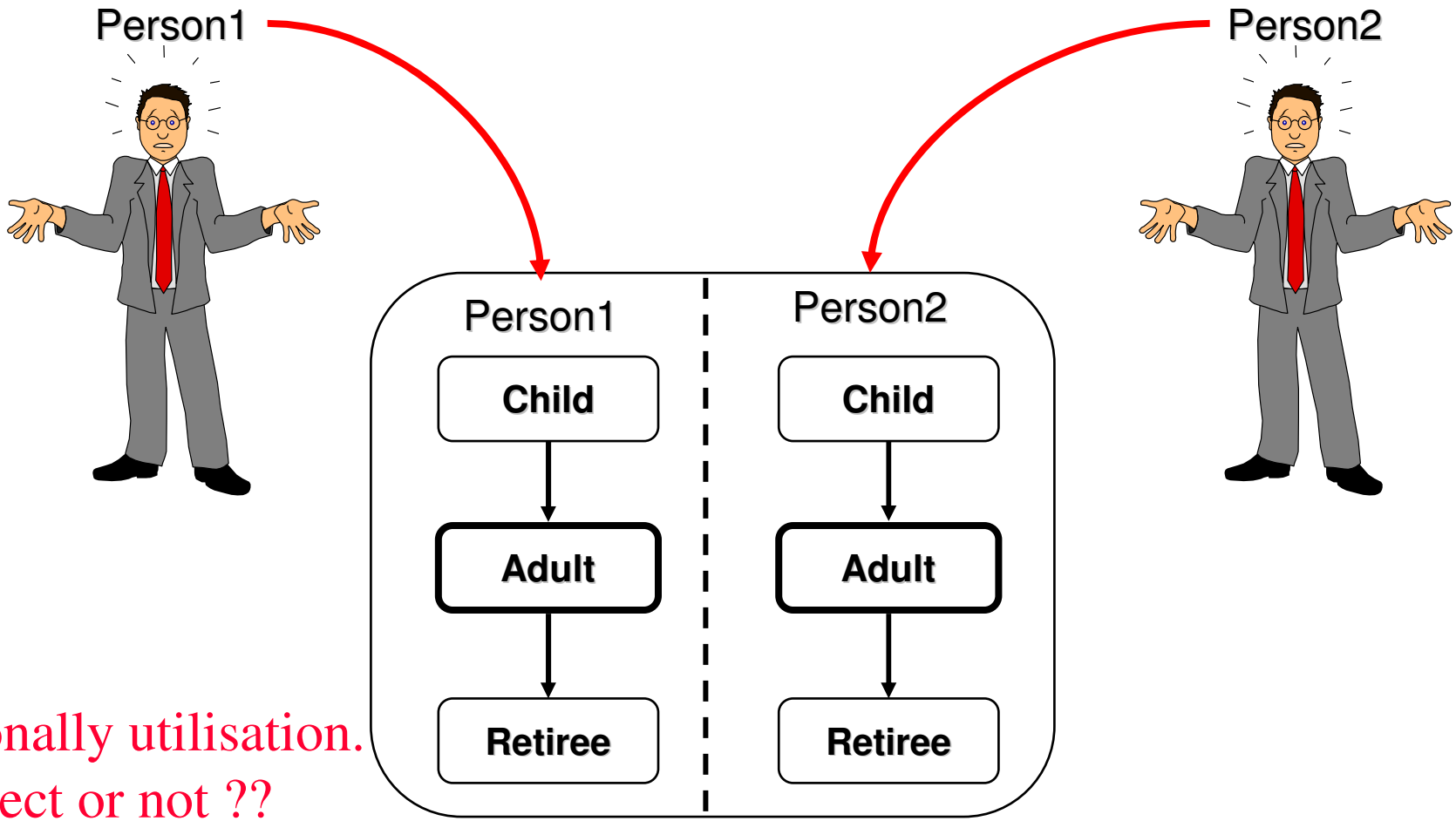


Behavioral Object Analysis
- Timing Diagrams -

- Complex Timing Diagram



Behavioral Object Analysis
- Exercise -



Orthogonally utilisation.
Correct or not ??

Summary

- Real-time systems analysis
  - Important stage, allowing the development of systems starting from a high abstraction level
  - Defines the “what” of the system
- Analysis may be thought of as discovering, in detail, the requirements of the final product, the object structure and their behaviour
- UML provides a set of tools to capture systems at high level
  - Intuitive wile still allowing designer to select detailed representation