


DA-IICT




IT313: Software Engineering

Modeling & Design

Saurabh Tiwari

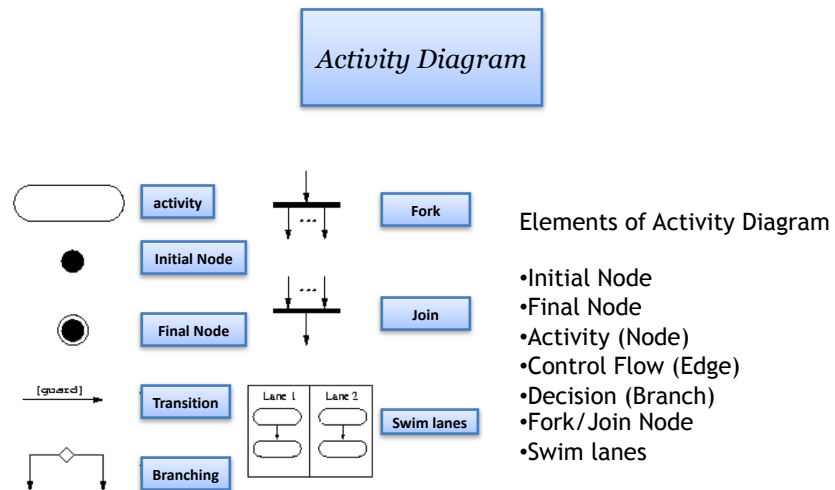
1



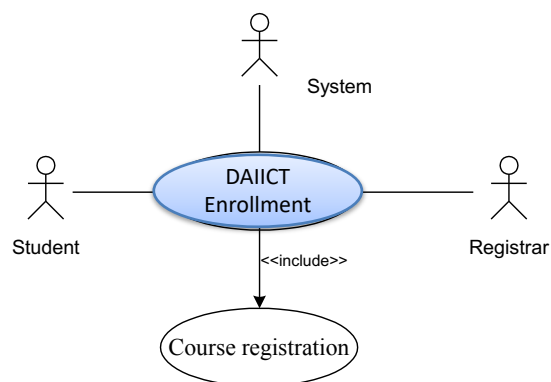
What is an Activity Diagram?

- Activity diagrams represent the dynamic (behavioral) view of a system
- Activity diagrams are typically used for business (transaction) process modeling and modeling the logic captured by a single use-case or usage scenario
- Activity diagram is used to represent the flow across use cases or to represent flow within a particular use case
- UML activity diagrams are the object oriented equivalent of flow chart and data flow diagrams in function-oriented design approach
- Activity diagram contains activities, transitions between activities, decision points, synchronization bars, swim lanes and many more...

Activity Diagram



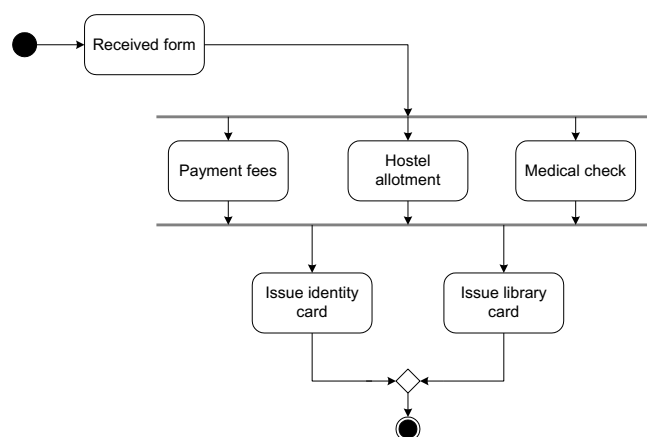
Student Enrollment in DAICT



SEDAIICT System

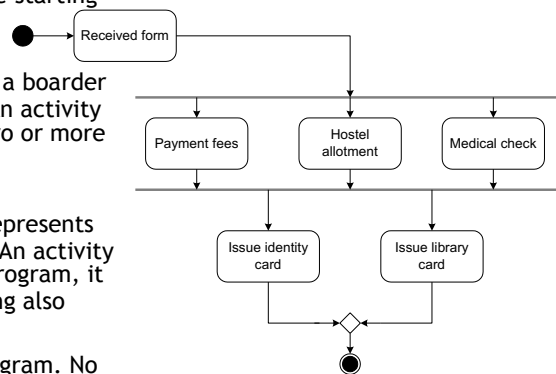
- Here different activities are:
 - Received enrollment form filled by the student
 - Registrar checks the form
 - Input data to the system
 - System authenticate the environment
 - Pay fees by the student
 - Registrar checks the amount to be remitted and prepare a bill
 - System acknowledge fee receipts and print receipt
 - Hostel allotment
 - Allot hostel
 - Receive hostel charge
 - Allot room
 - Medical check up
 - Create hostel record
 - Conduct medical bill
 - Enter record
 - Issue library card
 - Issue identity card

Activity Diagram for the Use Case



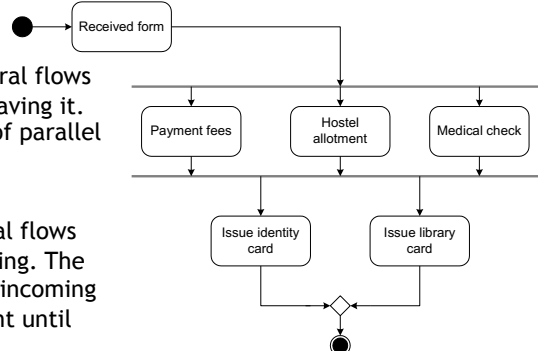
Basic Components in an Activity Diagram

- **Initial node**
 - The filled circle is the starting point of the diagram
- **Final node**
 - The filled circle with a boarder is the ending point. An activity diagram can have zero or more activity final state.
- **Activity**
 - The rounded circle represents activities that occur. An activity is not necessarily a program, it may be a manual thing also
- **Flow/ edge**
 - The arrows in the diagram. No label is necessary



Basic Components in an Activity Diagram

- **Fork**
 - A black bar (horizontal/vertical) with one flow going into it and several leaving it. This denotes the beginning of parallel activities
- **Join**
 - A block bar with several flows entering it and one leaving it. this denotes the end of parallel activities
- **Merge**
 - A diamond with several flows entering and one leaving. The implication is that all incoming flow to reach this point until processing continues

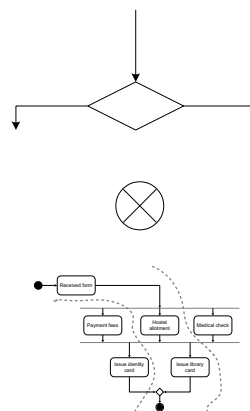


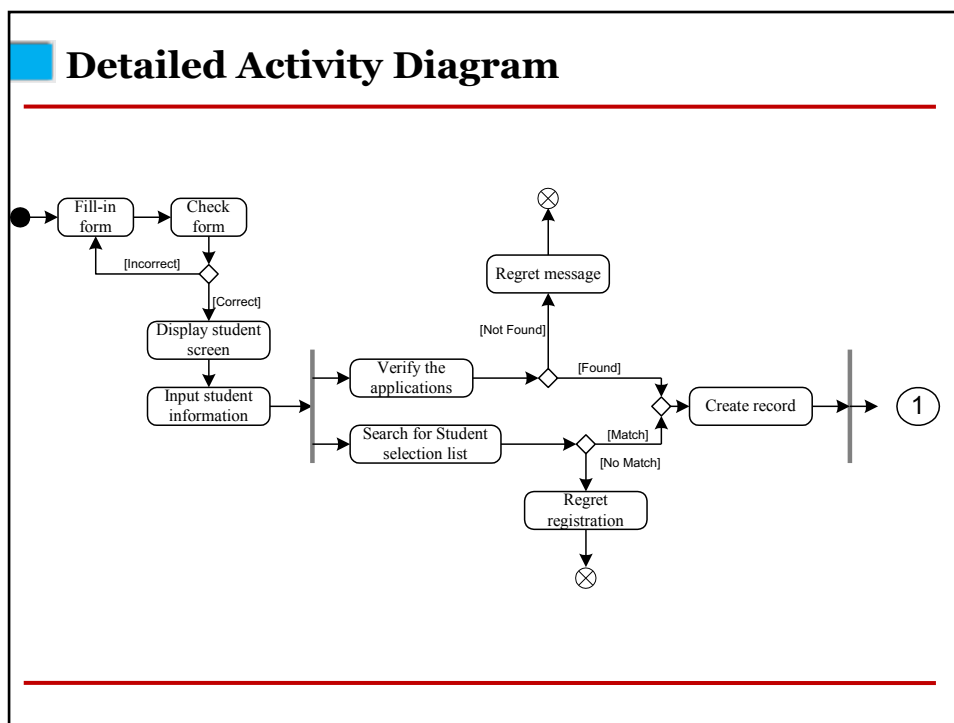
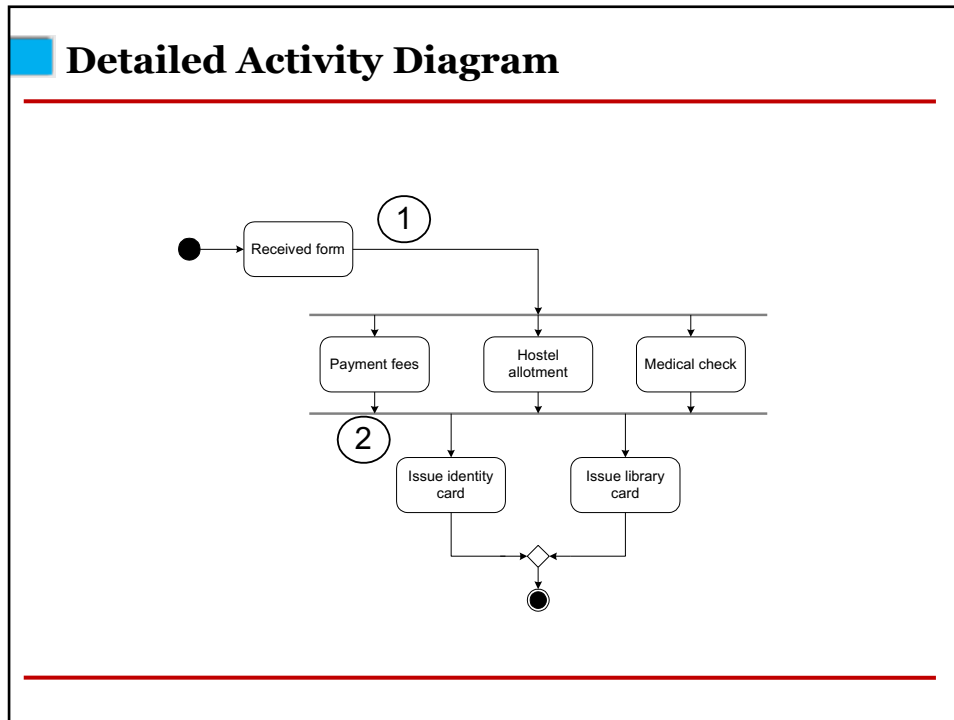
Basic Components in an Activity Diagram

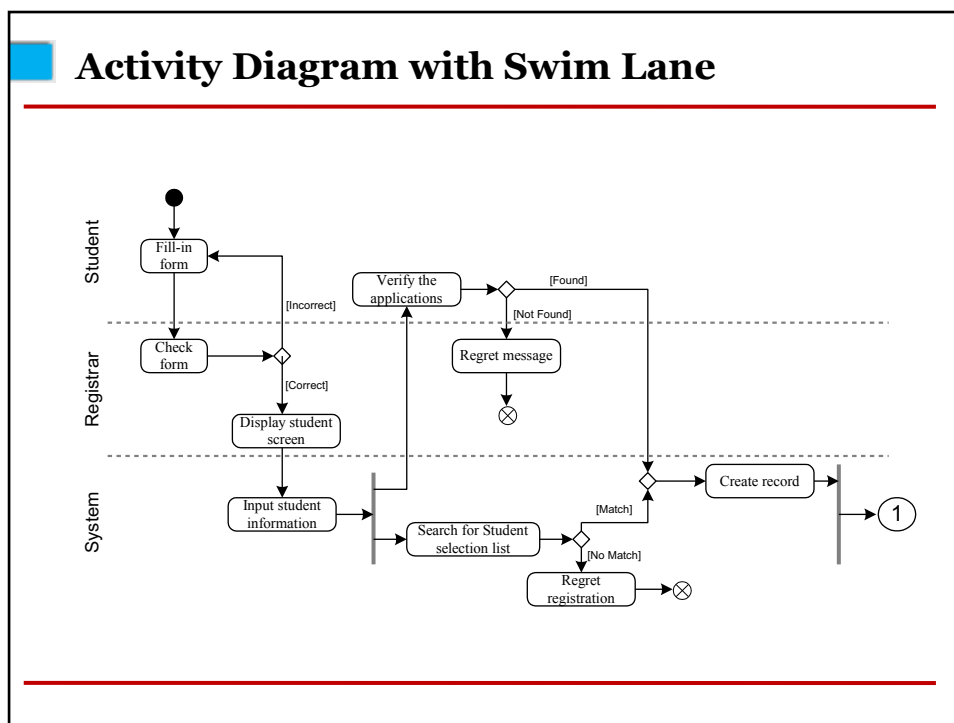
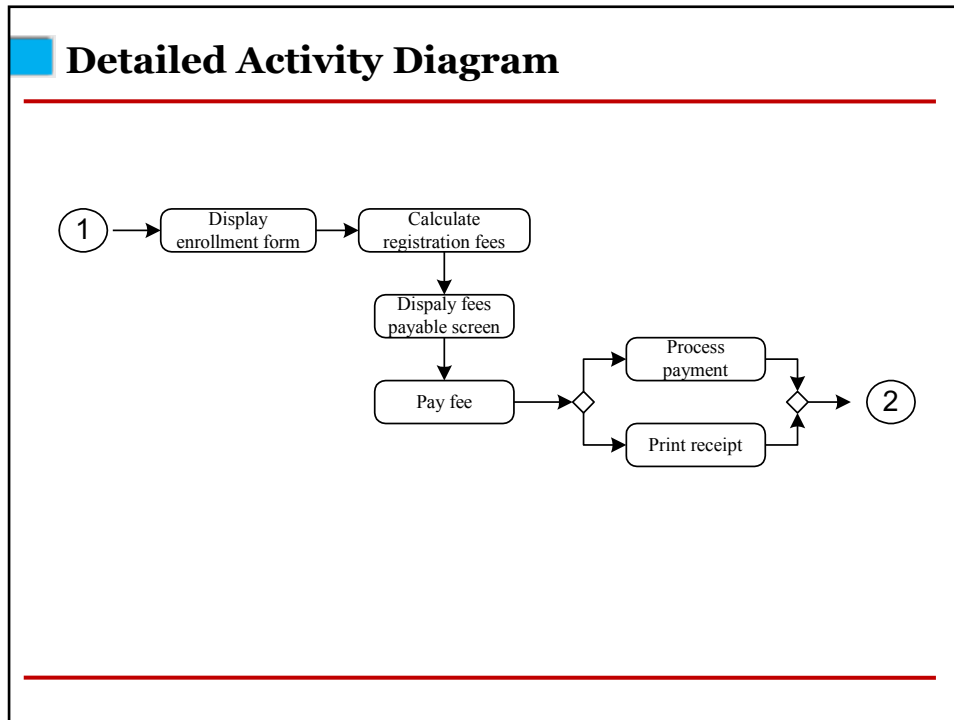
- Difference between **Join and Merge**
 - A join is different from a merge in that the join synchronizes two inflows and produces a single outflow. The outflow from a join cannot execute until all inflows have been received
 - A merge passes any control flows straight through it. If two or more inflows are received by a merge symbol, the action pointed to by its outflow is executed two or more times

Basic Components in an Activity Diagram

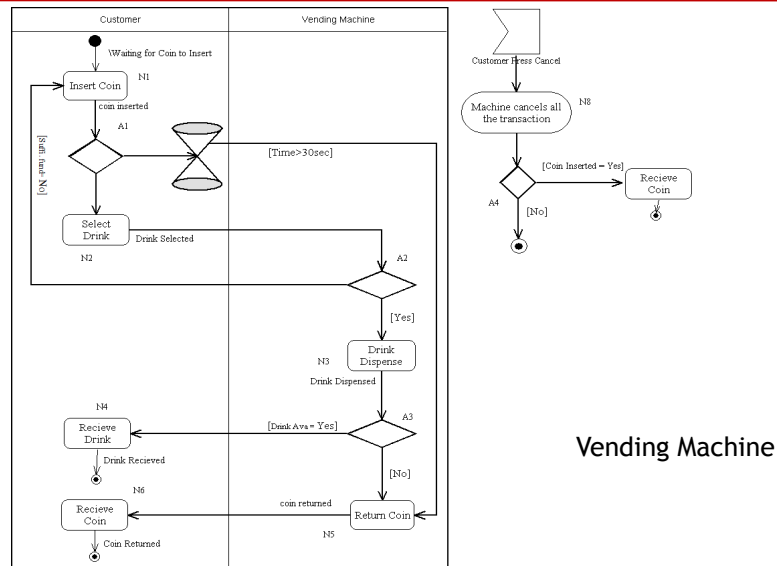
- **Decision**
 - A diamond with one flow entering and several leaving. The flow leaving includes conditions as yes/ no state
- **Flow final**
 - The circle with X through it. This indicates that Process stop at this point
- **Swim lane**
 - A partition in activity diagram by means of dashed line, called swim lane. This swim lane may be horizontal or vertical







Activity Diagram with UML 2.0



Problems to Ponder

- How activity diagram related to flow chart? How it defers from flow chart?
- How methods in classes and activities can be correlated?

Problems to Ponder

- Draw the activity diagrams for
 - Library Information System
 - Bank ATM
-

Exercises?

- Prepare an activity diagram for computing a restaurant bill. There should be a charge for each delivered item. The total amount should be subjected to a tax and a service charge of 18% for group of six or more. For smaller groups, there should be a blank entry for a gratuity according to the customer's discretion. Any coupons or gift certificates submitted by the customer should be subtracted.
-



Activities

- Total items
- Add tax
- Credit coupons and certificates
- Customer determines gratuity [less than six]
- Add 18% [six or more]

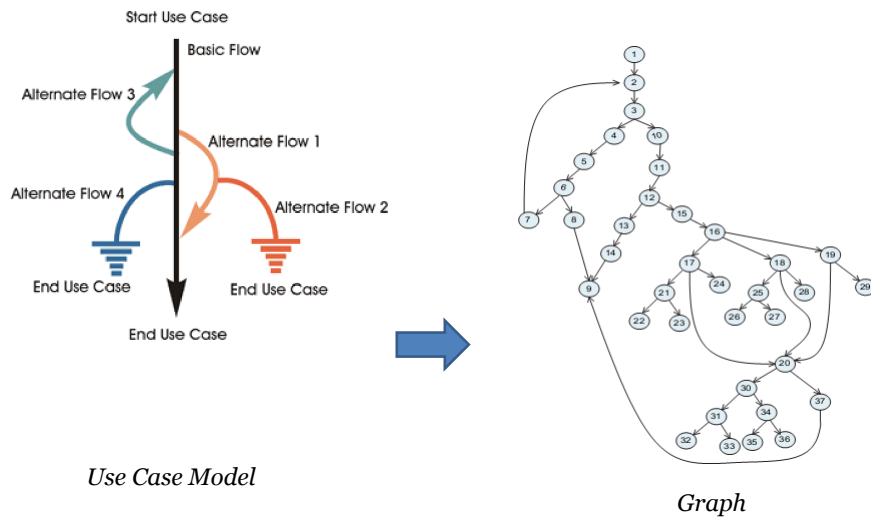
Activity diagram???



Exercises?

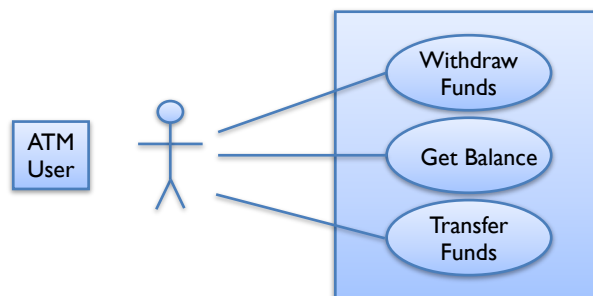
- Prepare an activity diagram that elaborates the details of logging into an email system. Note that entry of the user name and the password can occur in any order.
 - Draw the activity diagrams for
 - Library Information System
 - Bank ATM
-

Use Case to Activity: Graph - Example



21

Simple Use Case Example



- **Actors** : Humans or software components that use the software being modeled
- **Use cases** : Shown as circles or ovals
- **Node Coverage** : Try each use case once ...

Use case graphs, by themselves, are not useful for testing

Elaboration of ATM Use Case

- Use Case Name : Withdraw Funds
- Summary : Customer uses a valid card to withdraw funds from a valid bank account.
- Actor : ATM Customer
- Precondition : ATM is displaying the idle welcome message
- Description :
 - Customer inserts an ATM Card into the ATM Card Reader.
 - If the system can recognize the card, it reads the card number.
 - System prompts the customer for a PIN.
 - Customer enters PIN.
 - System checks the card's expiration date and whether the card has been stolen or lost.
 - If the card is valid, the system checks if the entered PIN matches the card PIN.
 - If the PINs match, the system finds out what accounts the card can access.
 - System displays customer accounts and prompts the customer to choose a type of transaction. There are three types of transactions, Withdraw Funds, Get Balance and Transfer Funds. (The previous eight steps are part of all three use cases; the following steps are unique to the Withdraw Funds use case.)

Elaboration of ATM Use Case – 2/3

- Description (continued) :
 - Customer selects Withdraw Funds, selects the account number, and enters the amount.
 - System checks that the account is valid, makes sure that customer has enough funds in the account, makes sure that the daily limit has not been exceeded, and checks that the ATM has enough funds.
 - If all four checks are successful, the system dispenses the cash.
 - System prints a receipt with a transaction number, the transaction type, the amount withdrawn, and the new account balance.
 - System ejects card.
 - System displays the idle welcome message.

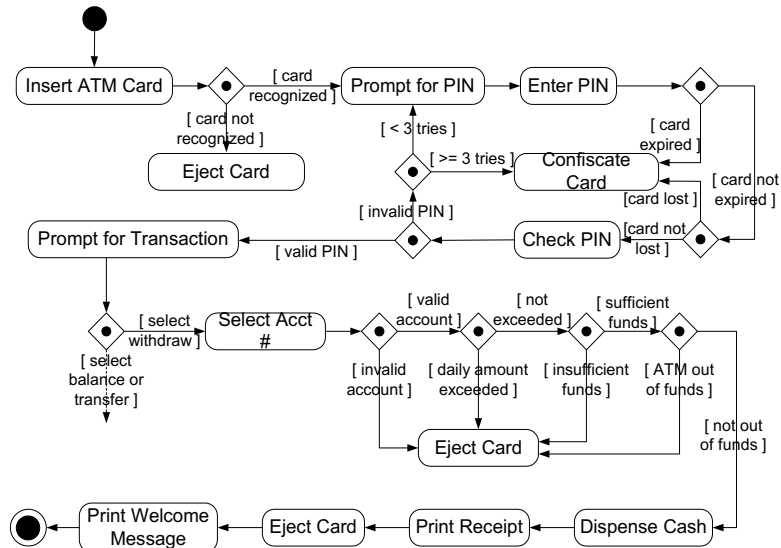
Elaboration of ATM Use Case – 3/3

- Alternatives :
 - If the system cannot recognize the card, it is ejected and the welcome message is displayed.
 - If the current date is past the card's expiration date, the card is confiscated and the welcome message is displayed.
 - If the card has been reported lost or stolen, it is confiscated and the welcome message is displayed.
 - If the customer entered PIN does not match the PIN for the card, the system prompts for a new PIN.
 - If the customer enters an incorrect PIN three times, the card is confiscated and the welcome message is displayed.
 - If the account number entered by the user is invalid, the system displays an error message, ejects the card and the welcome message is displayed.
 - If the request for withdraw exceeds the maximum allowable daily withdrawal amount, the system displays an apology message, ejects the card and the welcome message is displayed.
 - If the request for withdraw exceeds the amount of funds in the ATM, the system displays an apology message, ejects the card and the welcome message is displayed.
 - If the customer enters Cancel, the system cancels the transaction, ejects the card and the welcome message is displayed.
- Postcondition :
 - Funds have been withdrawn from the customer's account.

Use Cases to Activity Diagrams

- Activity diagrams indicate **flow among activities**
- Activities should model **user level steps**
- Two kinds of nodes:
 - **Action** states
 - **Sequential** branches
- Use case descriptions become **action state nodes** in the activity diagram
- Alternatives are **sequential branch nodes**
- Flow among steps are **edges**
- Activity diagrams usually have some helpful characteristics:
 - Few loops
 - Simple predicates

ATM Withdraw Activity Graph

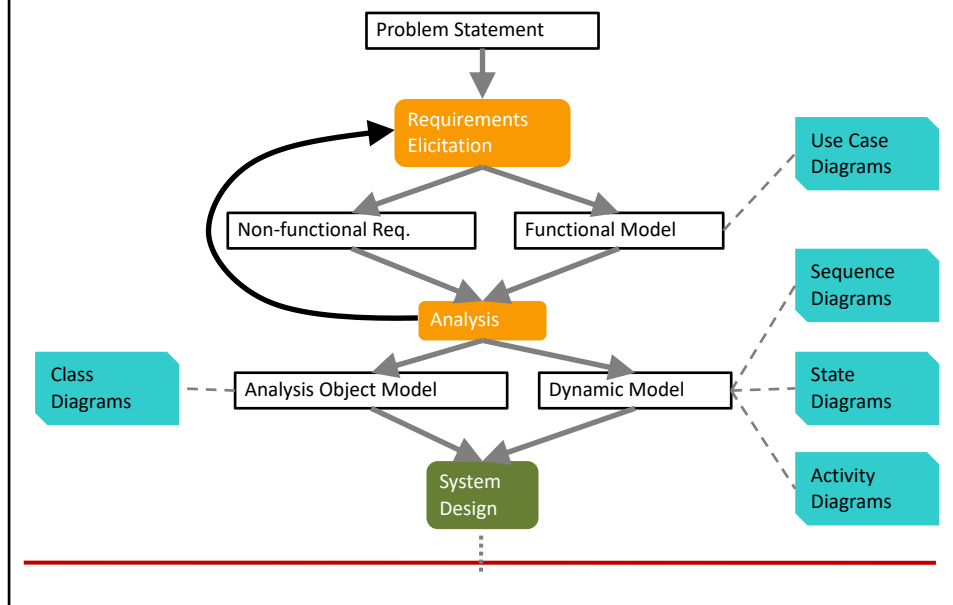


Exercise?

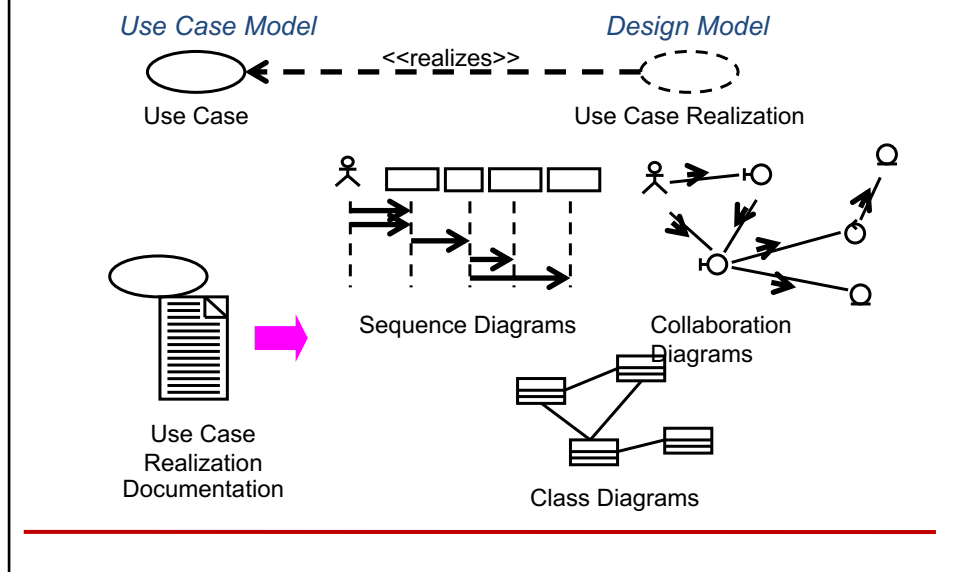
Use case Name	Process Sale	
Actors	Cashier, Catalog System, Inventory System	
	Steps	Actions
Basic Flow	1	Cashier starts a new sale.
	2	Cashier enters item identifier.
	3	POS System retrieve item information from the catalog system and, records sale line item and presents item description, price, and running total. Cashier repeats steps 2 until indicates done.
	4	POS System calculates and presents total price.
	5	Cashier tells Customer the total, and asks for payment.
	6	Customer pays and POS System handles payment.
	7	POS System records completed sale and sends sale information to the external Inventory system for stock update.
	8	POS System prints receipt.
	9	Customer leaves with receipt and goods.
Alternate Flow	Steps	Branching Actions
	2.1	IF the item entered by the Cashier is Invalid identifier THEN POS System Indicate error and Cashier enters the item manually.
	7.1	IF the items stock gets below a predefined minimum place a reposition order THEN Cashier deletes the item
	6.1	IF The Customer not have enough money THEN Customer asks the cashier to Cancel the transaction
	6.2	IF Customer says they intended to pay by cash but don't have enough cash THEN Customer uses an alternate payment method. The cashier tells customer to pay by card.
Precondition	Cashier is identified and authenticated.	
Postcondition	Sale is saved. Accounting and Inventory are updated. Receipt is generated. Payment authorization approvals are recorded.	

Prepare an activity diagram??

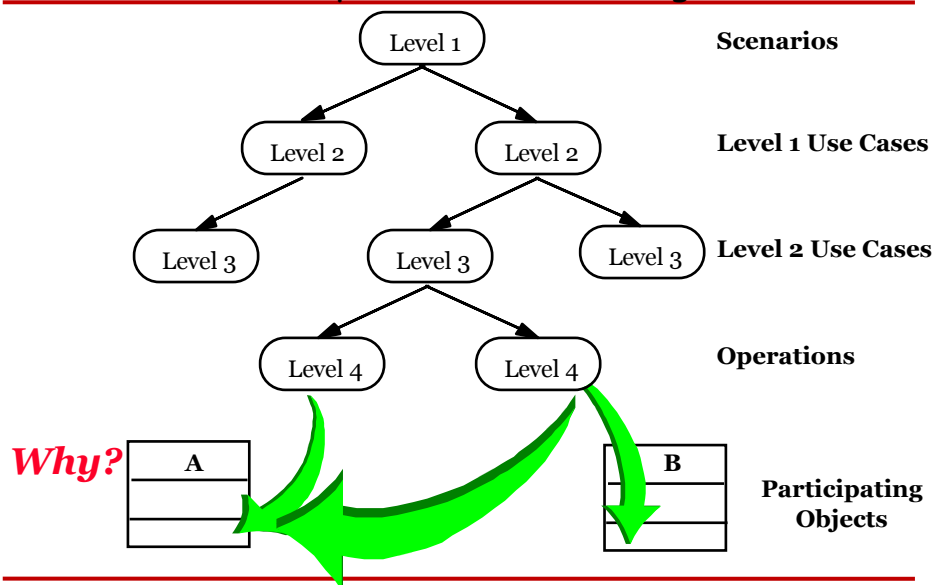
An overview of OOSE development activities and their products



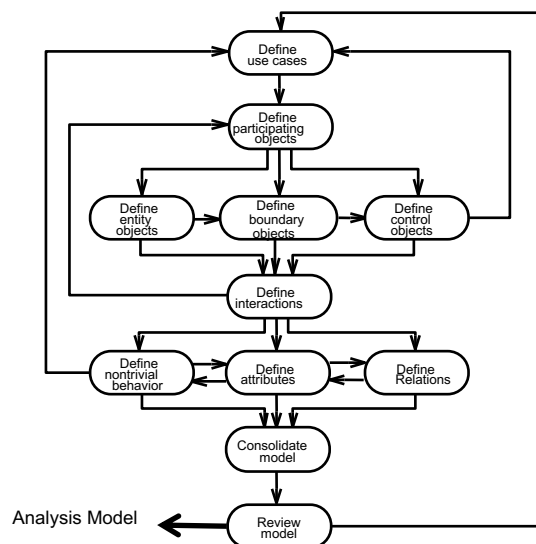
Use Case Realization






From Use Cases to Objects: Why Functional Decomposition is not Enough



Analysis Modeling

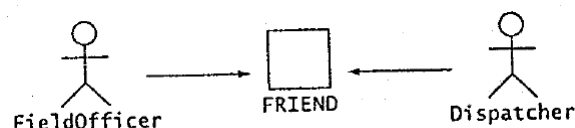


Types of Analysis Classes

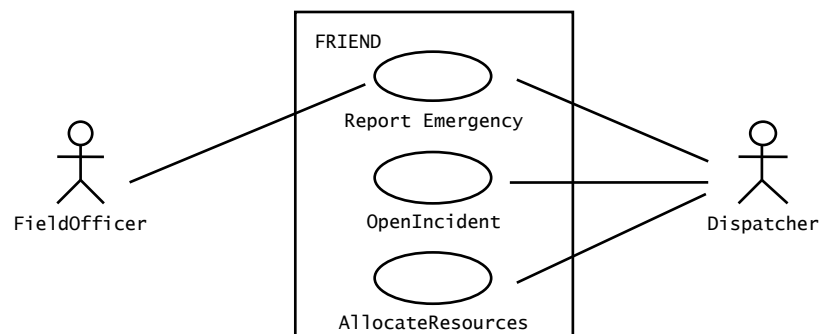
- **Entity classes:** 
 - Model persistent data, real world entities, e.g., roles, invoices, databases, file
- **Boundary classes:** 
 - Interaction between the system and its actors, e.g. receiving/presenting data
 - Examples: printer interfaces, terminals, sensors, APIs, forms, GUI items
 - Each boundary class should be related to at least one actor
- **Control classes:** 
 - Classes for coordination, sequencing, processing, transactions, control of other objects etc.

An Example

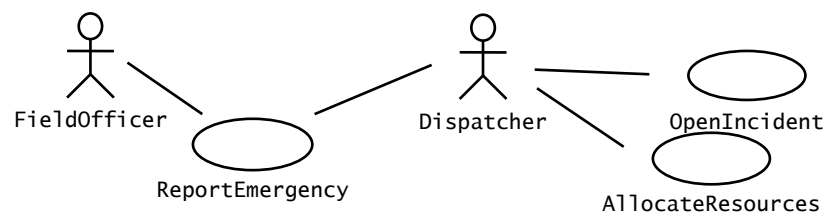
- **FRIEND**
 - A distributed information system for accident management. It includes many actors such as *FieldOfficer*, who represent the police, fire officers who respond to accidents, and *Dispatcher*, the police officer responsible for answering 911 calls and dispatching resources to an accident. FRIEND supports both actors by keeping track of incidents, resources, and task plans. The *FieldOfficer* and the *Dispatcher* interact through different interface – *FieldOfficer* interacts FRIEND through a mobile personal assistant, and *Dispatcher* access FRIEND through a workstation.



Example: Report Emergency use case in the Accident Management System (FRIEND)



Communication between actors and use cases in FRIEND.



Use case ReportEmergency

Use case name	ReportEmergency
Actors	FieldOfficer, Dispatcher
Entry condition	1. The FieldOfficer activate the "report Emergency" function of her terminal
Flow of event	<p>2. System responds by presenting a form with different details to filled-in</p> <p>3. FieldOfficer completes the form. She may also describe possible responses to the situation and request specific resources. She submits the form</p> <p>4. The Dispatcher reviews the information and creates an Incident in the DB by invoking OpenIncident use case. All the information received from FieldOfficer is then stored in the DB. The Dispatcher selects a response and allocates resources to the Incident (with AllocateResources use case) and acknowledge the Emergency Report by sending a FRIENDgram to the FieldOfficer</p>
Exit condition	5. The FieldOfficer receives the acknowledgement and the selected response

Identify Entity Objects

Heuristics

- Terms those are needed to clarify in order to understand the use case
- Recurring nouns
- Real-world entities
- Data sources or sinks

Use case ReportEmergency

- Entity Objects

- Dispatcher, EmergencyReport, FieldOfficer, and Incident

Dispatcher	Police officer who manage incidents. A Dispatcher opens, documents, and closes Incidents in response to emergency Report. Dispatchers are identified with batch numbers
Emergency Report	Initial incident information report
Field Officer	Police officer on duty and on the spot. She is responsible for reporting the incident. Recognized by her batch number.
Incident	All relevant information of the incident

Identifying Boundary Objects

- Represent system interface with the actors of the system
- Each actor must interact with at least one boundary object
- Need not to be much elaborated here

Identifying Boundary Objects

Heuristics

- Identify **user interfaces** for actor-system interactions
- Identify **forms** in which data is to be filled in
- Identify **communication/messages** between actor and systems
- When multiple actors are involved in a use case, identify **actor terminals**
- Do not model visual aspects of the interface
- Use user/customer language for describing interfaces

Use case Report Emergency

- Boundary Objects
 - AcknowledgementNotice, DispatcherStation, ReportEmergencyButton, EmergencyReportForm, FieldOfficerStation, IncidentForm

Acknowledgement Notice	
DispatcherStation	Computer used by the Dispatcher
ReportEmergency Button	
EmergencyReport Form	
FieldOfficerStation	Mobile computer used by the FieldOfficer
IncidentForm	This form is presented to the Dispatcher on the DispatcherStation when the EmergencyReport is received

Identify Control Objects

Heuristics

- Identify **one control object per use case**
- Identify **one control object per actor** in the use case
- Life span of a control object should cover the extent of the use case or extent of a user session

Use case Report Emergency

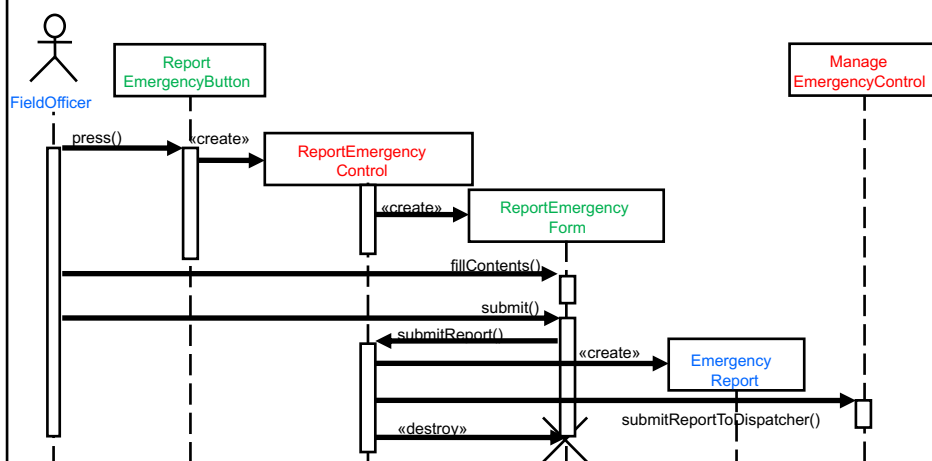
- Control Objects
 - **ReportEmergencyControl, ManageEmergencyControl**

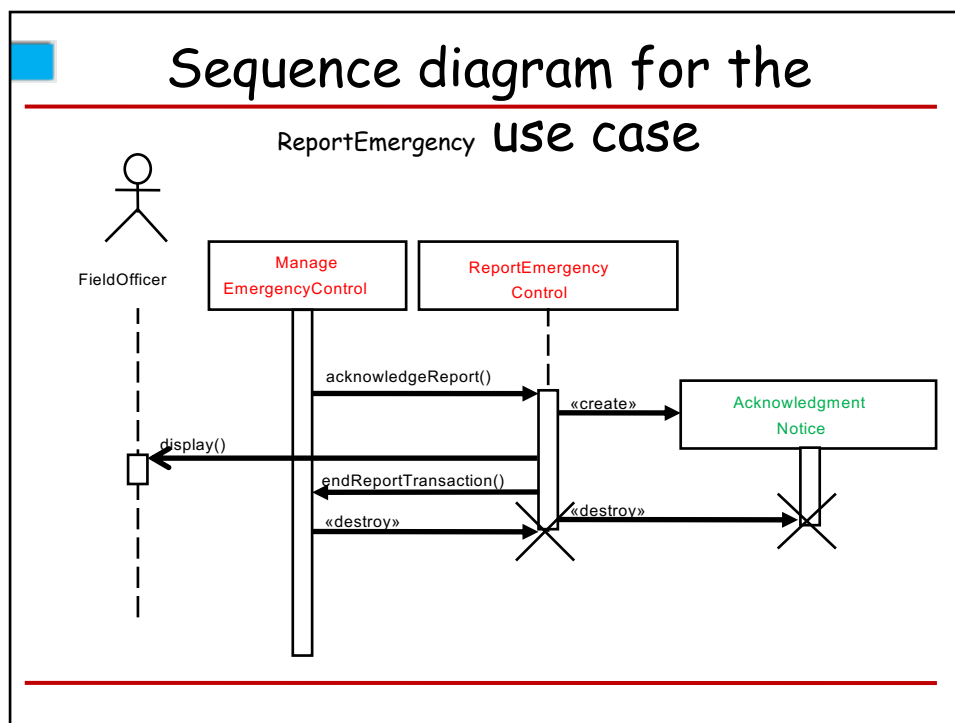
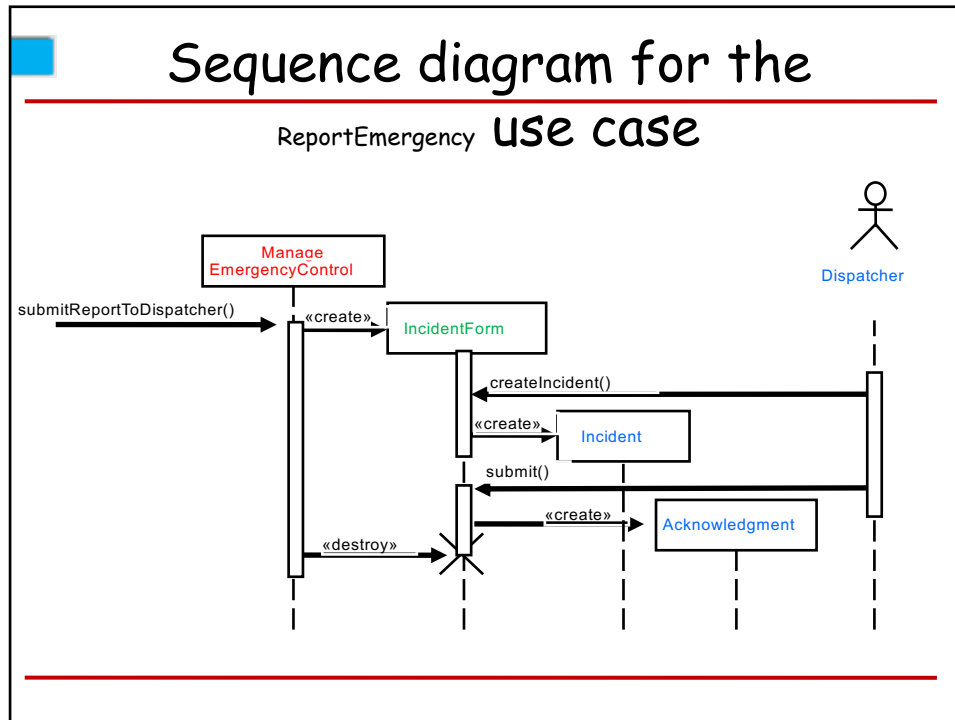
ReportEmergencyControl	The object is created when the FieldOfficer selects the "Report Emergency" button. It then creates an EmergencyReportForm and presents it to the FieldOfficer. After getting all the information, it forwards this information to the DispatcherStation. It then waits for an acknowledgement. When received, it creates an acknowledgementNotice and displays it to the fieldOfficer
ManageEmergencyControl	This object is created when an EmergencyReport is received. It then creates an IncidentForm and displays it to the Dispatcher. Once the Dispatcher has created an incident, allocated resources, and submitted an acknowledgement, it forwards the acknowledgement to the FieldOfficerStation

Mapping Use Cases to Object Interactions (Sequence Diagrams)

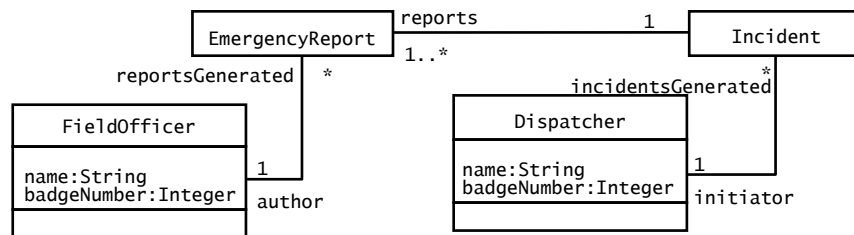
- To ensure the **completeness** of our model
- Sequence diagrams tie **use cases** with **objects** and **their interaction**
- **Not good** at the user level but is a step in transformation
- Allows to find **missing objects** or **grey areas** in requirement specification

Sequence diagram for the ReportEmergency Use case.

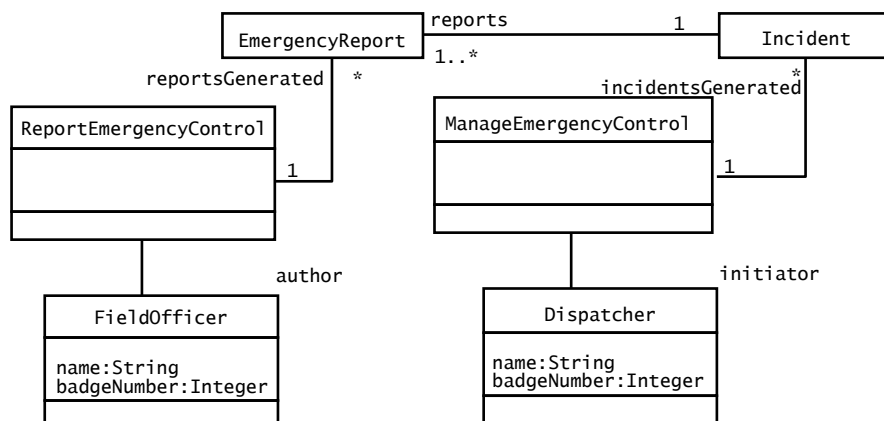




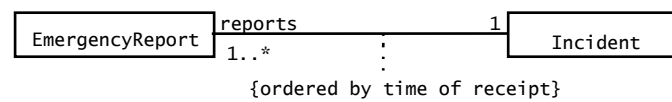
Classes that participate in the ReportEmergency use case.



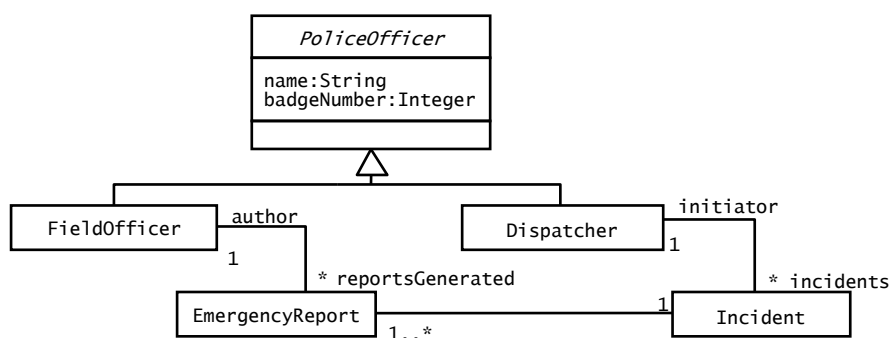
Classes that participate in the ReportEmergency use case.



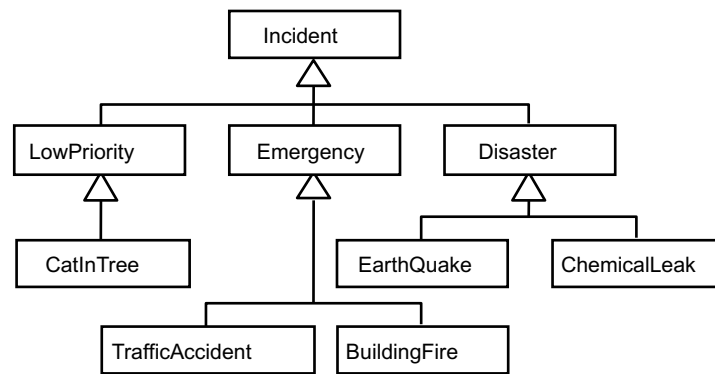
An example of constraint.





An example of a generalization



Another example of a generalization



Summary: Requirements Analysis


1. What are the transformations?  **Functional Modeling**
 - Create *use case diagram and scenarios*
 - Talk to client, observe, get historical records, do thought experiments
2. What is the structure of the system?  **Object Modeling**

Create *class diagrams*

Identify objects.

What are the associations between them? What is their multiplicity?

What are the attributes of the objects?

What operations are defined on the objects?
3. What is its behavior?  **Dynamic Modeling**

Create *sequence diagrams*

Identify senders and receivers

Show sequence of events exchanged between objects. Identify event dependencies and event concurrency.

Create *state diagrams*

Only for the dynamically interesting objects.