

Analysis Part II

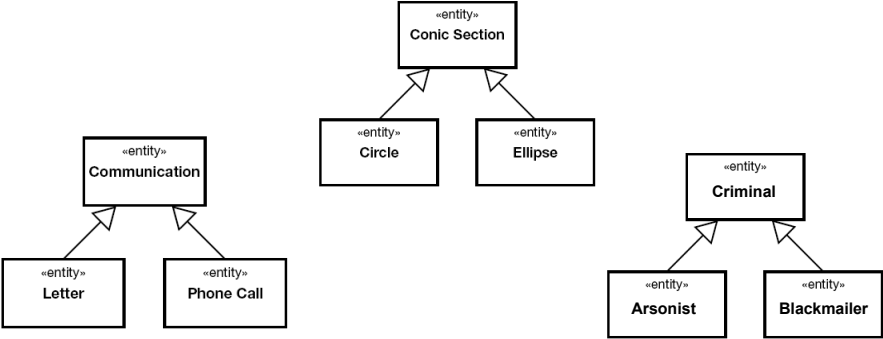
First Model continued:
Generalization, State and Activity

Generalization

- Finding
- Assessing
- Depicting
- Pitfalls

Generalizations

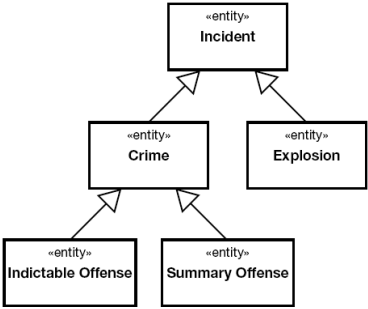
- Model entities that have some, but not all features in common
 - Relationship represented by an open triangular arrowhead



3

Generalizations

- Can have multiple levels
 - generalization of a generalization
- Has many names
 - generalization / specialization
 - a.k.a. gen-spec
 - is-a-kind-of
 - also provides a good test
 - super-type / sub-type
 - inheritance
 - best left as a design term



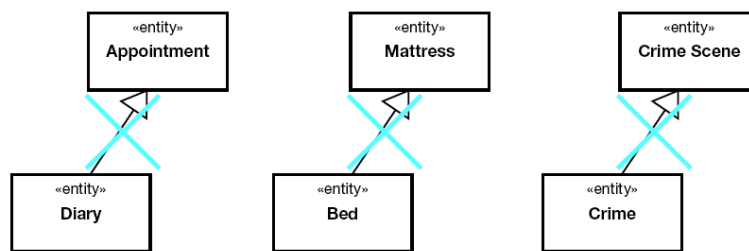
4

Incorrect Generalizations

- Fails the "is-a-kind-of" test



- Examples of "has-a" relationship

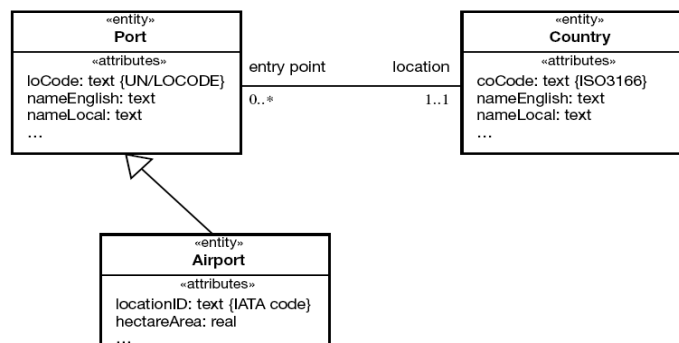


Should be aggregations

5

Generalization and "Inheritance"

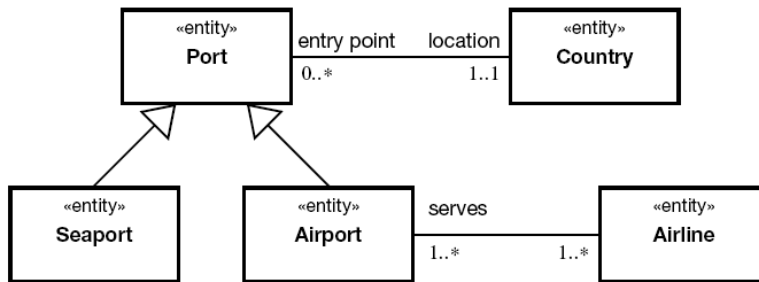
- Specialized entity shares **all** characteristics of the generalized entity
 - attributes **and** associations
 - nothing *private* at this level of modeling



6

Generalization and "Inheritance"

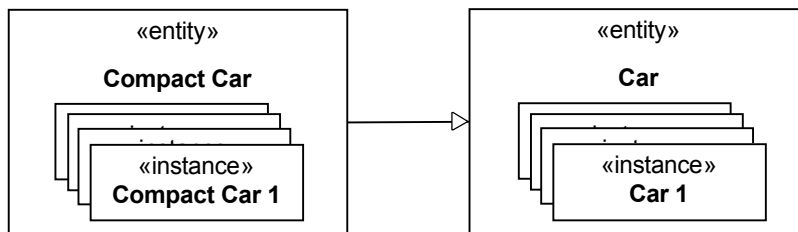
- Specialized entity shares **all** characteristics of the generalized entity
 - attributes **and** associations
 - nothing *private* at this level of modeling



7

Generalization vs Instances

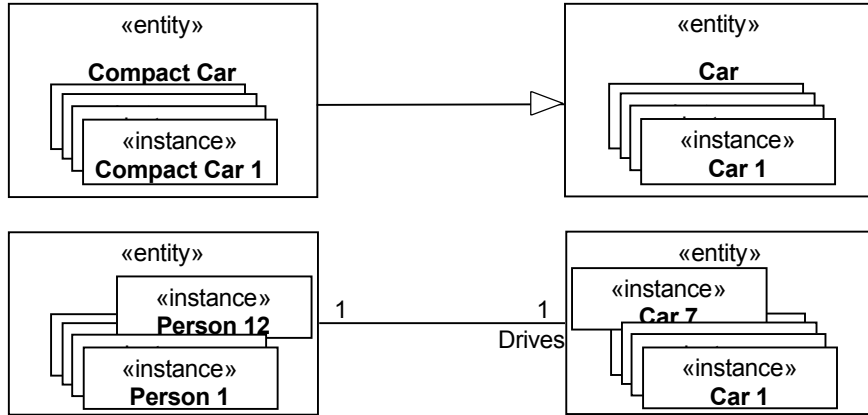
- The instances of an entity set (or entity class)
 - Are identical
- Generalization are more abstract
 - the specialized entities under a generalization has more features / behaviours then the generalized entity



8

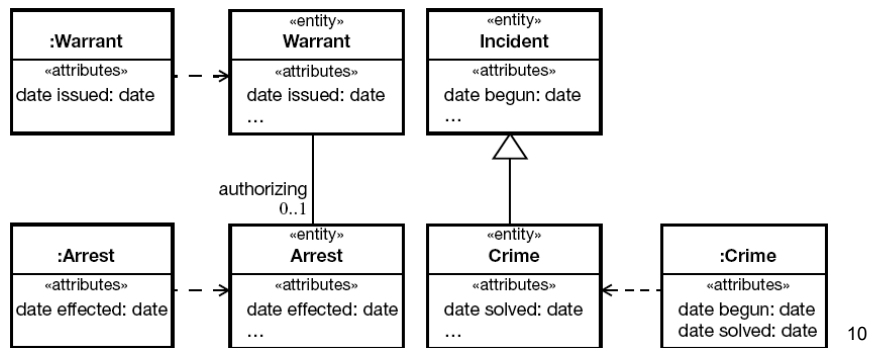
Generalization vs Instances

- Generalization relationships relate whole sets together
 - No linkage of instances (substitute one set for another)
 - Whereas association and aggregation relate instances together



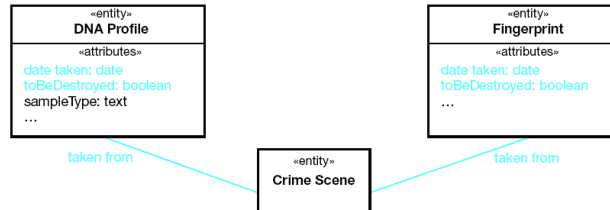
Generalization vs Instances

- In subject matter model
 - generalization entities have instances
- In design models,
 - generalizations should be kept abstract (no instances)



Finding Generalizations

- Generalizations promote normalization
 - look for commonalities



- Go from specializations to generalizations
 - Other way usually produces too many levels
 - Keep generalization hierarchy as flat as you can
 - Einstein on models:
 - "Everything should be made as **simple as possible** ... **but not simpler**"

11

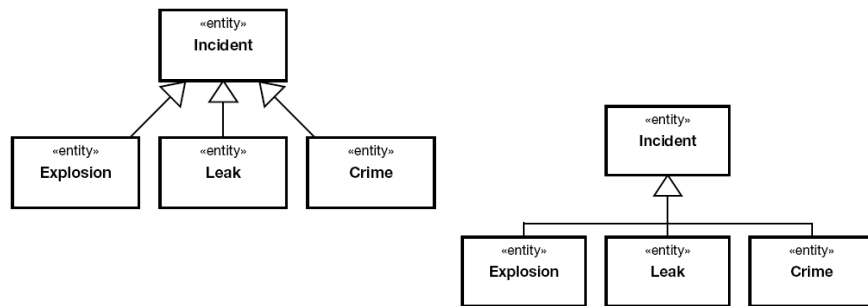
Assessing Generalizations

- Are you respecting the substitutability principle?
- Does **each instance** of the specialization **really** have **all** of the characteristics of the generalization?
- Is the relationship not just a coincidence?
- It is not an association between instances of a "specialized" entity to instances of a "generalized" entity
 - instances should substitute if generalization, not be linked
- It is a real generalization and not just less work

12

Depicting Generalizations

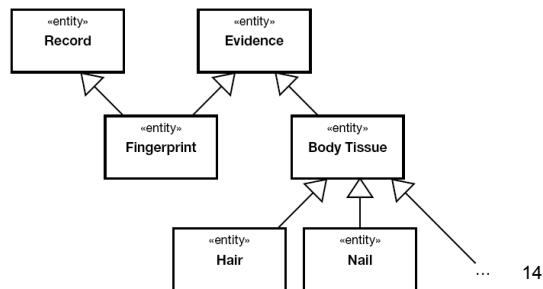
- There are normally no labels or other embellishments
- The “shared target” (or “comb”) is often shown in the literature
 - It can make diagrams harder to maintain however



13

Multiple Generalization

- Natural in Subject Matter Model
 - can be problematic in design and technical model
 - various languages implement "multiple inheritance" in different ways
 - If fits your model, use it and worry about design details later



14

Pitfalls of Generalization

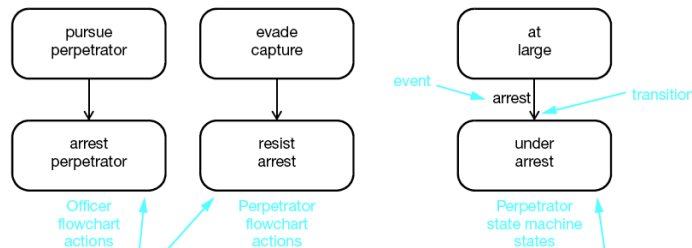
- expecting that generalization relationships will always **directly** translate into design relationships
- not realizing that the generalization relationship is completely different to the association and aggregation relationships
- spending too much time too early on generalization relationships that might never bear fruit
 - a contributor to "analysis paralysis"
- starting at the top and supposing that there might be useful specializations
 - should start at the bottom and looking for actual commonality among accurately defined entity sets

15

State Machines

Actions vs Changes of State: what to study?

- Actions \Rightarrow state changes and events
 - Unfortunately, actions in the subject matter do not match that of the software
- Significant events and state changes \Rightarrow actions
 - The state changes of the subject matter *do* match the state changes of their software counterparts

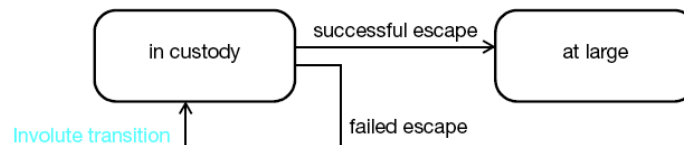


Noting these actions doesn't help us design the software as much as does noting these states.

16

State machines

- Are in one and exactly one of a finite number of states
- Events occur
- If an event labels a transition from the current state, the state machine transitions to the transition's destination state
- The destination state can be the same state
 - Thus recognizing the event without changing state of being



19

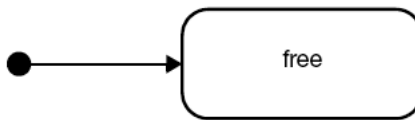
Basic State Machine Elements

- States
 - "history" or memory
- Events
 - the input
- Transition
 - events cause transitions from state to state
 - transition labeled by the event that caused the transition

20

Basic State Machine Elements

- Initial state
 - The birth state of an entity instance
 - The constructed, initial state of a software instance

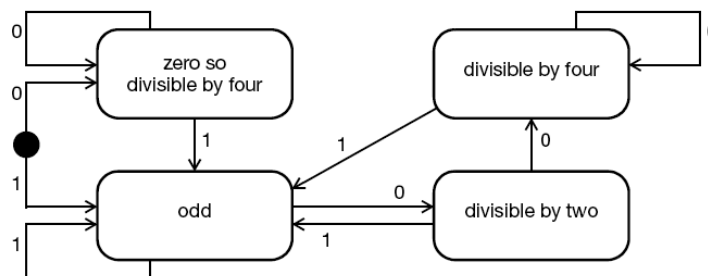


- "Dot" is not a state
- Points to the state

21

Simple and Classic Example

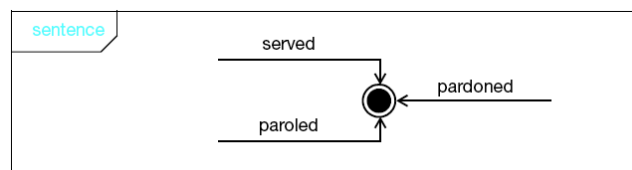
- **Problem:** classify a binary number as divisible by 4
 - all binary numbers with 2 '0's at the end is divisible by 4
 - zero is divisible by 4
- **States:** 2 states - {even, odd}
- **Events:** binary string input
- **Transitions:**



22

Basic State Machine Elements

- Final state
 - The state in which an entity instance will never sense another event
 - An object state in which all messages are unexpected and erroneous



Note: – Frame around the state transition diagram should be used
– it reminds the reader of which entity is undergoing the state transitions

23

Finding States, Events and Transitions

Understanding of the life pattern of an entity or object can begin with state or event

- “A crime is open or solved; what events move us between those states?”
- “There are arrest, convictions, reopening, ...; in what states do crimes exist?”

24

Naming States and Events

- We said that a basic state machine is always in exactly one state
- So transitions must be instantaneous
- And time passes only in states
- So name states and events accordingly
 - State names indicate states of being
 - Present participles (-ing) are often good
 - Event names indicate their instantaneous, their momentary nature

25

Events and Messages

- You can't equate analysis state machine events with messages
 - Because then you might be led to ask "from whom?"
 - And that question is often unanswerable and inevitably its answer is unhelpful
- An event is simply something that happens in the context of the entity, to which it is sensitive

26

State Machines and Generalization

- Theoretically, it's possible to abstract common state, event and transition patterns
 - Into generalization state machine
 - That specialization state machines inherit from
- However, getting self-contained state machines correct is difficult enough
- Adding the also complicated and challenging notion of inheritance into the mix
 - Seems to be asking for near-insuperable trouble

27

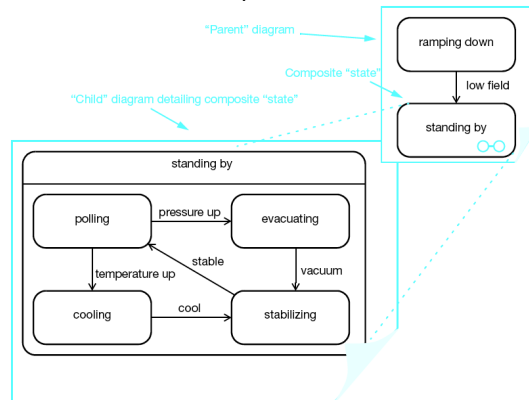
State Groups

- The difficulty we have in abstracting state machines,
 - Means that they tend to be large
 - They fail to “chunk”; they don't respect our 7 ± 2 brain limits
- A composite state is a group of states portrayed with a single state-like symbol for the purpose of understanding
 - The detail can then be relegated to a separate, “child” diagram

28

State groups (composite states)

- State groups allow for the simplification of a complex state diagram
 - Allow us to “chunk”, thus respect our 7 ± 2 brain limits
- Forms a hierarchy
 - remember true state machine is the “expanded” version



29

Entering Group States

- Which "nested" state is being pointed to?

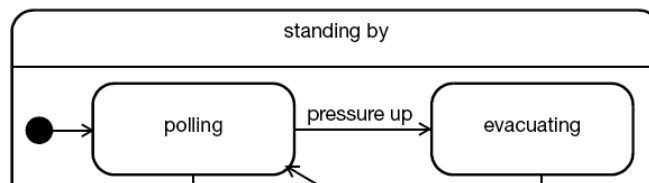


- Three possibilities
 - using initial state markers
 - using named entry points
 - using history indicators

30

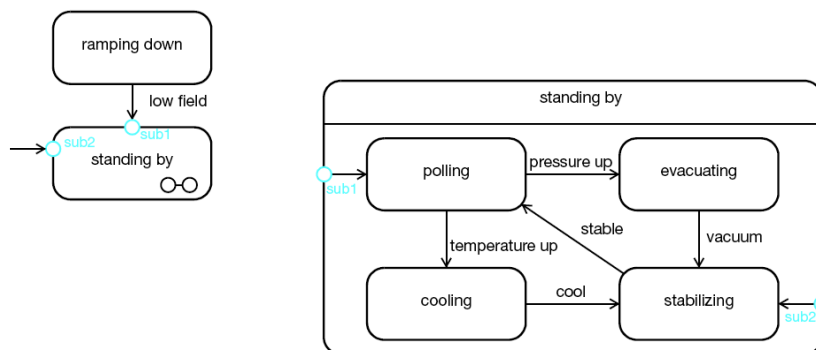
Using initial state markers

- Use initial state symbol
 - however can be confused with actual initial state
 - can only point to one state
 - possible that different transitions can enter different states



31

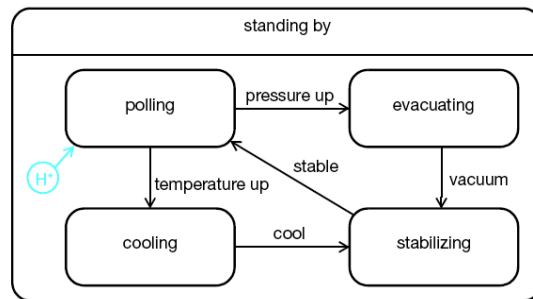
Using entry points



32

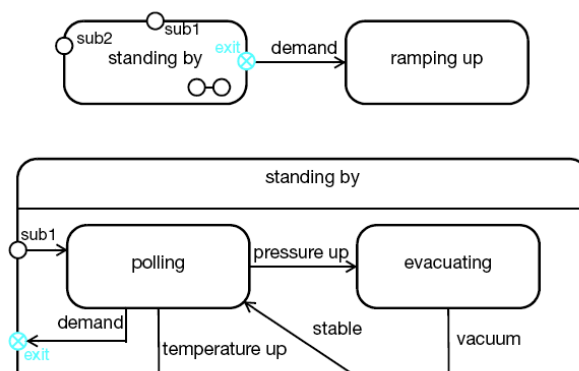
Using History Indicators

- Enter the same state that the group was in when the group was last exited
- symbol H* is used
- H* points to the state to go to when the group is first entered (it is not a state itself)



33

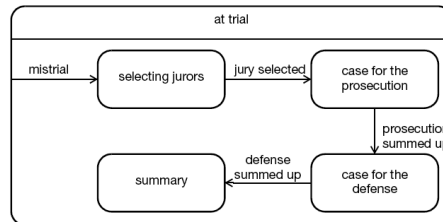
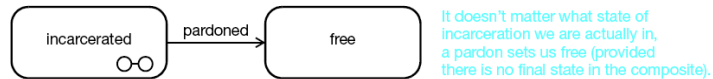
Exiting Group States



34

Exiting Group States

- If every state in group can exit with the same transition, can skip exit point
 - if exit point is not there... assume there is one everywhere
 - excellent at decluttering



35

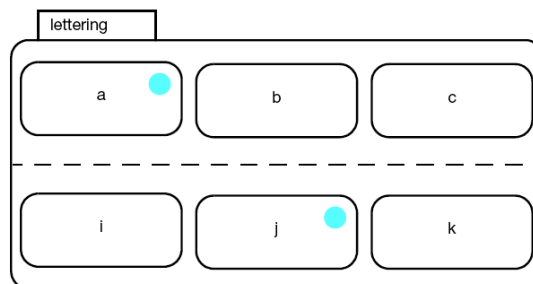
Concurrent State Groups

- There's another way to group states
- A state in a state machine can already be considered to be an equivalence partition
 - Partitioning up actual values ("micro" states) into more abstract, "macro" states
- We can go further
 - And partition up the states of a state machine into partitions – concurrently executing partitions

36

... Concurrent State Groups

- Instead of our machine always being in one state
 - Instead of “marking” one state as the state that the machine is “in”
- The concurrent region of our machine is always in n states
 - Where n is the number of concurrent partitions



We don't actually draw the tokens of course; but if we did, an ordinary state machine would always have one and exactly one state marked as the current state. A concurrent machine has one state from each region "marked".

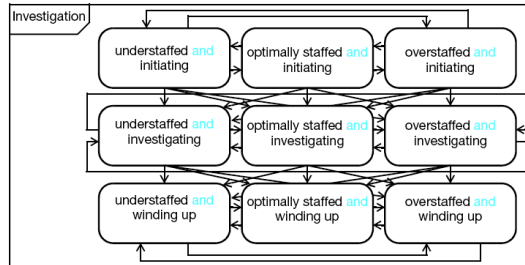
37

... Concurrent State Groups

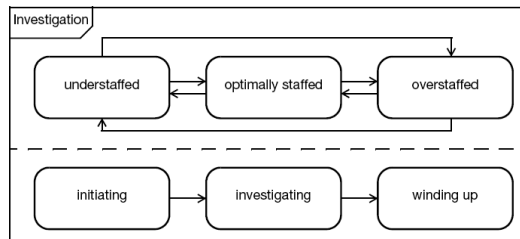
- On the face of it, that would seem to complicate things
- However, it simplifies things when the life described by a state machine
 - Has unconnected aspects each of which can be in one of a number of states
- Without concurrency, honest names need to be permuted names
 - E.g. At work and on call, at work and off call, at home and on call, at home and off call

38

Concurrent state groups



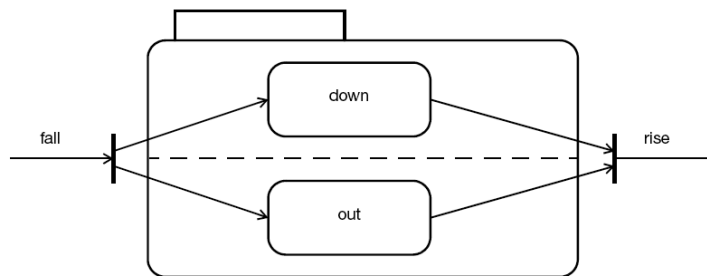
- Can be simplified as



39

Entering and exiting concurrent state groups

- Think of forking concurrent processes
 - use same diagrammatic symbols
- Note: no concurrency actually involved
 - refer back to permutation states from last slide



40

Guards and Post Conditions

- guarding conditions for a transition
 - guard is a Boolean condition
 - refers to micro-state values
 - often attribute values

evacuate [safety officer = false]
→

- post conditions can also be specified
 - placed after a "/" character

evacuate [safety officer = false] / [outside = true]
→

41

State Machine Traps and Pitfalls

- Don't imagine every entity (or type or class) needs a state machine
 - If you "need" a state machine, you might be too complicated
 - Most entities, types and classes can get by without one
- If you do need a state machine, keep it simple
 - Avoid actions
 - Entity state machines show states of being and order the significant events in an entity's life
 - (Object state machines can take the natural picture that an event is a message and a message implies a method)
 - Lavish care and attention
 - A 85% correct state machine is 15% useful
 - A 95% correct state machine is 50% useful
 - A 99% correct state machine is 99% useful

42

... State Machine Traps and Pitfalls

It isn't possible to have a single interpretation of state machines that works for subject matter models and implementation models, e.g.

- Typical analysis
 - Events are significant happenings
 - Events not accepted in a state might not be errors
- Typical design
 - Events are messages
 - Messages not accepted in a state are probably errors

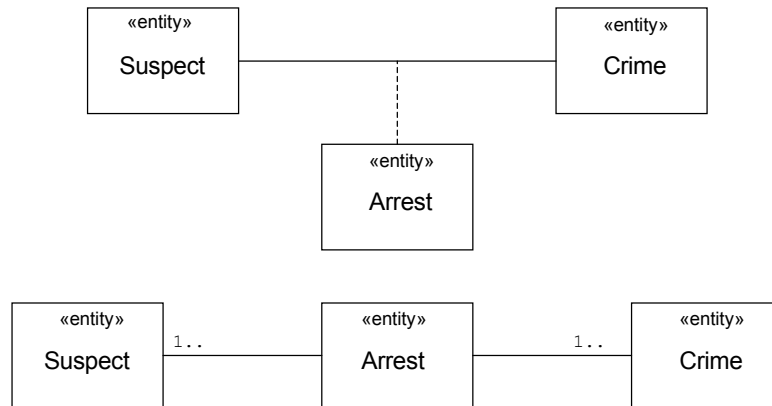
43

Association Entities

- As well as being existence-dependent on another entity,
 - An entity can be existence-dependent on an association
 - Characterizing the association
 - This works better than allowing associations to have attributes, which other approaches allowed
- If you like, it's a specific and iconic depiction of a common pattern of multiplicities

44

... Association Entities



45

Constraints

- Carefully constructed and validated constraints
 - Can significantly improve the accuracy of the model
 - E.g. “only students in full-time education and under 21 years of age can take out the FW3 loan”
- Attempting to show constraints in diagrams,
 - Often looks ambiguous
 - And must therefore be fully and formally entered into the model repository,
 - Using an appropriate constrain language
 - Like the UML’s OCL (object constraint language)

46

Activity Diagrams

Dealing with algorithms

- The nature of activity diagrams
- The place of activity diagrams
- The elements of an activity diagram

47

Dealing With Algorithms

- Given an appropriate granularity
 - Enough good, cohesive entities
 - When the time comes,
 - In the next model
 - Methods' algorithms shouldn't usually be too challenging to describe
 - We will return to this later
- What would we do with,
 - An apparently complicated, large and irreducible subject matter algorithm
 - Whose details must be captured now; for whatever reason it won't be feasible to study them during the workup of the next model?
- What if we were doing some business systems analysis
 - Wanting to understand a process rather than feed the design of a computer system?

48

The Nature of Activity Diagrams

- They have changed radically
 - Mostly for the better
 - In UML 2
- Their presence denormalized UML 1
 - They overlapped with state machines
- In UML 2 they have made their mind up as to their nature
 - They take flow viewpoint
 - They show the flow of data and the flow of control
 - They can do what flowcharts, Petri nets and dataflow diagrams did in past generations of software methods

49

... The Nature of Activity Diagrams

- ... they take a flow viewpoint
- We need to be careful with activity diagrams
- Their use in business systems analysis and in workflow
 - I.e. documenting systems in general
 - Or doing classical systems analysis prior to computerization
 - Is quite acceptable
- They have only the tiniest smidgen of object-orientation
 - And on a path developing an object-oriented computer system,
 - Any more object-oriented alternatives would be vastly preferable
 - Such as a sequence diagram
- The evidence is incontrovertible,
 - That obtaining good objects from flow-based models is very, very difficult

50

The Elements of An Activity Diagram

- Actions
 - Like the steps of flowcharts
 - easily describable, fairly atomic thing that is to be done
 - Predefined and low level



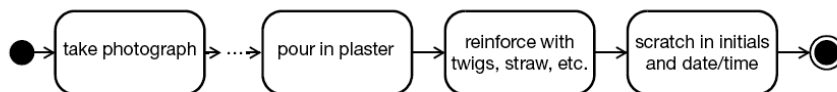
- Flows
 - Of control – the stuff flowcharts
 - Which can be concurrent – the stuff of Petri nets
 - Of action-provoking data – the stuff of dataflow diagrams



51

Initial and final nodes

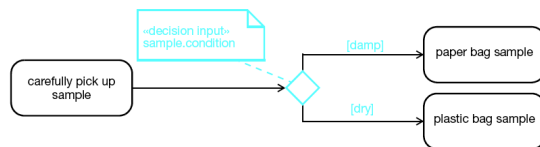
- The start here and end here symbols of flowcharts
- Think of
 - initial node as token producer
 - final node as token consumer



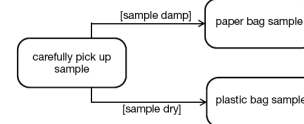
52

Decisions

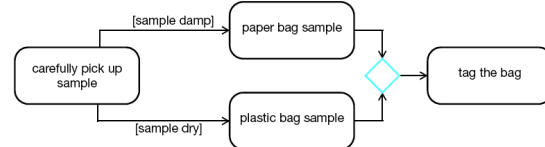
- Slightly easier to draw, but essentially the old diamond of flowcharting
- Use guards (square brackets) to label choice paths



- Can skip the diamond – implicit choice



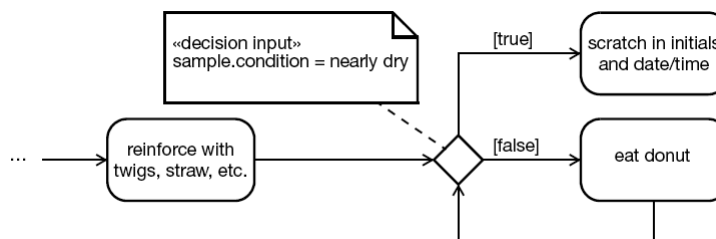
- Use the diamond to merge back together



53

Iterating

- Specific loop describing symbol – as yet undefined
- So for now just loop back to a previous event with a condition determining the number of times



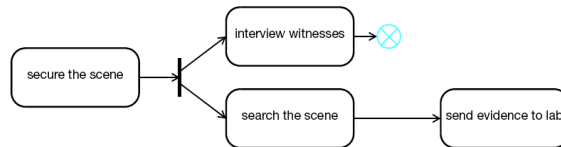
54

Concurrency

- Forks
 - Starting and ending concurrency – the main Petri net facilitator



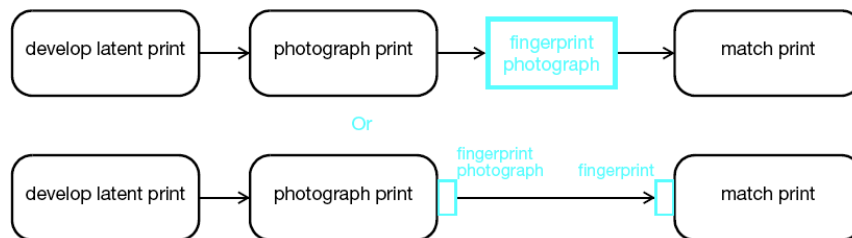
- Flow final nodes
 - The end of just one of some concurrently active flows



55

Object nodes

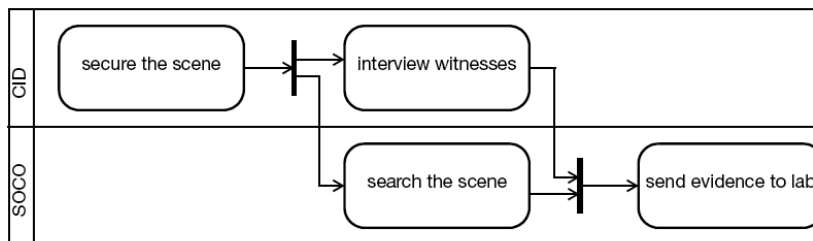
- Not objects as we might know them
- But part of the dataflow stuff



56

Partitions

- Partitioning the diagram into regions depicting the people, departments or object carrying out the actions
- Each partition called a "swimlane"



(CID is Criminal Investigation Department and SOCO is Scene of Crime Officer)

57

Useful in Analysis?

Having already been careful with activity diagrams in analysis, we have a few more worries

- Composition relationships
 - Containment, physical composition, coincident lifetime
 - Not a distinction one can usefully make in subject matter models
 - Useful in a later model
- Use relationships
 - Just about no predicting power
 - Useful in a later model
- Methods
 - Just about no predicting power
 - Useful in a later model
- Messages
 - Just about no predicting power
 - Useful in a later model

58

First Model Summary

- We have greater, and consensus, understanding of the subject matter
- We have good ideas for types of object instances (but not yet for classes)
- We have good ideas for query services (but not yet for instance variables)
- We have good ideas for characterizing relationships
- We might have state transition constraints
- We have hints as to possible generalization and conformance structures