Petri Nets

• More powerful and intuitive depiction of control flow
  strong on depiction of parallelism and concurrency
• A Petri Net structurally consists of
  – A finite number of places
  – A finite number of transitions
  – A finite set of arrows that connect places to transitions
    (or vice versa)
• If an arrow goes from a place to a transition, then place is
  said to be an input place of the transition.
• If an arrow goes from a transition to a place, then place is
  said to be an output place of the transition.

Marking and Firing Petri Nets

• A Petri Net place can be marked by the presence of a token
  – Any collection of places can be marked.
  – Any such marking is said to define a state of the Petri Net
• Petri Nets proceed from one state to another by means of a firing
  – Occurs only when every input place of a transition is marked with a token.
• The effect of the firing of a transition is to
  – Remove all of the tokens from the transition’s input places
  – Put tokens in all of the transition’s output places

Some Semantics

• PN = (Places, Transitions, Edges)
  – Places (PN) = \{place \_i\}
  – Transitions (PN) = \{transition \_j\}
  – Edges (PN) = (Inedges U Outedges)
    – Inedges (PN) = \{(place, transition)\} \times \{ Transition \_j \}
    – Outedges (PN) = \{(transition, place)\} \times \{ Places \_i \}
  where \_i \in Places (PN), \_j \in Transitions (PN)
• Marked: PN \times Places \rightarrow \{True, False\}
  – If Marked (PN, place \_i) = True we say that place \_i is marked
• A transition, \_t \in Transitions (PN) can fire if
  for all of its inedges, (place, \_t) Marked (place) = True
  – After a transition \_t \in Transitions (PN) fires
    – Marked (place) \rightarrow False for all places, \_p such that \(\_p, \_t\) \in Edges (PN)
    – Marked (place) \rightarrow True, for all places, \_p such that \(\_t, \_p\) \in Edges (PN)

Use Cases

• Specify “actors” and how they interact with various component parts of a system
  – This is an external “black box” view of a system
• System is a collection of “use cases” (ie. functional capabilities)
• Represented using diagrams and schemas
  – Diagrams show flow of “uses” between actors and use cases
  – Schemas are more formal non-pictorial definitions

Example Use Case diagram
Use Case schema

Use case: Heat Cooking Tank
Description: Heat a cooking tank to the temperature prescribed by the recipe of the juice to be mixed, and keep it at that temperature for the time prescribed by the recipe.

Read: Cooking tank, Batch, Recipe

Change:
- Operator: Batch ID
- Thermometer: Current temperature
- Heater: Switch on, switch off

Assumptions: Juice is present in tank

Transactions: When the heater is switched off, a message is sent to the operator.

Message Sequence Diagrams

• Sometimes called “ladder charts”
• Represent a particular sequence of messages exchanged between entities
• Popular in object-oriented methods to represent communications between objects
• Shows one particular communication sequence in one run of the system
• Shows behavior as well as communication
• Can be extended with conventions to represent looping, casing, timeouts, synchronization, global conditions across different entities, delayed message reception, etc.

Example Message Sequence Diagram

Class Diagram

• In widespread use. Consists of
  – Name
  – Attributes
  – Operations/Methods
  – Associations
    » Cardinalities
    » Annotations
    » Qualifiers
    » Interfaces
    » More….. (much more)

Class diagram for juice plant

Collaboration Diagrams

• Popular in object-oriented methods to represent message exchanges between objects
• Object specification augmented by annotations that represent dataflows between the communicating objects
• Differ from other notations
  – Nodes represent objects, not activities (as in DFDs, activity diagrams, activity charts, and block diagrams)
  – Nodes represent object instances, not object classes
• As in sequence diagrams, represent the sequence of messages in one particular scenario, not all possible communications scenarios.
Representing Other Types of Things

- Data, Objects, Artifacts
  - These are clearly secondary in all of the above diagrams
  - Often are more important than functional view
  - Harder to depict diagrammatically
- Process artifacts and views
  - Primary interest of management and customer stakeholders for much of the time
  - Typical questions:
    - What is the (development) plan? schedule?
    - Are we almost done?
    - What are we going to do next?
    - What if Joe quits?
  - Different representations are needed to reply effectively

Representation of Data/Objects

- Complement to emphasis on representation of activities
  - Foregoing representations all focussed on activities
  - Weak capabilities for describing data and objects
  - Seen mostly as effects of activities
  - Numerous places where data descriptions were needed
    - eg. Request List in elevator example
    - Supposed to be sorted (which way?)
    - Elements had fields (what types?)
  - Problems in doing this well
    - What information needed/what questions need answers?
      - Hierarchical decomposition of data
      - Legal actions on data
      - Typing information
    - What forms of representation will be useful?
      - Natural language
      - Diagrammatic
      - Formal language

Pert and CPM Charts

- Depict the process as a network of tasks
- Each step is a circle
- Incoming arrows are steps that must complete before this one
- Outgoing arrows are steps that might follow this one
- Each step has a time estimate
- No loops allowed
- So that maximum “flow time” can be computed
  - Along the “critical path”
- Early management tool
- Very naive and oversimplified view
  - no loops!!
  - Simplicity is its strength and weakness

Gantt Charts

- Familiar milestone charts, progress charts, ....
- Time represented along a horizontal axis
- Each task (person, ...) represented by a solid bar plotted against the time line
- Bar starts at “start time” and ends at “end time”
- Key Milestones represented by triangles placed along the bar
- Shows how tasks juxtapose
- Shows who should be doing what at all times
- Shows how product is supposed to evolve over time
- Effective for spotting schedule slippages

Example Gantt Chart
Data Representation Diagrams

Record Structures

Array Structures

Linked List Structures

Multiple Views

Plato’s Cave

RSL/REVS: A Very Old Example

- Reference: Bell, Bixler, Dyer, IEEE TSE SE-3 #1 pp. 49-59
- Enhanced DFD’s (called RNET’s)
  -- Functions called ALPHA’s
  -- ALPHA’s can be defined hierarchically
  -- Structured English as well as pictorial diagrams
  -- Diverse set of attributes used to enhance ALPHA def’s.
  -- Logical connectives on arrows
- Balanced by facilities for defining data (called DATA’s)
  -- Defined hierarchically
  -- Output from/input to specs are dual of ALPHA info.
- Supports consistency checking through redundancy
  -- Redundancy to check for consistency and quality
- Designed to help specify reactive systems
  -- Dataflow diagrams are particularly good at that (?)
- Other Features (more peripheral to this discussion)
  -- I/O specifications
  -- Requirements tracing (why is that requirement here?)

STATEMATE: A Newer Example

- Elaborate enhancement of FSM’s
  -- Augmented by other views (e.g. activity Diagrams)
- Key feature is maintenance of consistency among views
  -- Done by projecting views of text (language)-based model
- References
- Commercially available software system
  -- http://www-01.ibm.com/software/awdtools/statemate/
Multiple Views in Statemate

• Rationale for multiple views: Too much information in a single diagram creates clutter, confusion, defeats clarity.

• Advantage of multiple views: Each represents a different viewpoint, different model, with a different diagram—easier to grasp the model.

• Disadvantage: Reader needs to synthesize views, assure that they are really consistent with each other.

• Three principal views in Statemate:
  --Module Charts (a hierarchy representing capabilities)
  --Activity Charts (hierarchical dataflow charts)
  --Statecharts (hierarchical finite state machines)

• All charts are derived from single abstract view
  --facilitated by use of slick user interface

• Three Statemate views depict some different views, but also overlap with each other: facilitates cross-checking for consistency and easier comprehension.

Module Chart

• Hierarchy shown by nesting module-charts inside each other
  --How many levels of nesting without losing clarity?

• Modules are solid square boxes

• External modules and functions are dashed boxes external to the outermost module chart box

• Internal dashed boxes depict data stores

• Arrows connecting boxes depict data flow
  --Arrows can be forked, have embedded connectors.....

Activity Chart

• A Data Flow Diagram
  --Hierarchical
  --focus (depicted by solid boxes) on functions
  --Arrows depict data flows

• All of this helps user-reader to associate features of one with features of the other

• New dataflow diagram feature:
  --Control box (rounded): at most one per activity
  --Suggests need to depict how and when data will flow among functions—not just what

**Example: How to represent an activity consisting of a set of cases with DFD’s?
--Dashed arrows represent flow of control information (eg, signals, commands, status reporting/changing)
>>This is redundant with arrows in Module Charts

• This, in turn, anticipates new view represented using the third type of chart

Statemate Data/Object Specification

• Structuring and primitive typing supported

• Done through a forms-based textual input

• Data stored in a relational database

• Tools to support cross-checking with specifications on charts

• Querying of database allowed

• No pictorial/diagrammatic support
Statecharts

- Extension of basic notion of FSM
- FSMs are effective in modeling systems that are
  - clearly and accurately modelled as being in only one of a
  - finite number of states at a time
  - considered to move from state to state driven by events
drawn from a finite set of possibilities
- Statecharts add some features to what basic FSM's can represent
  - Hierarchy: >>Keeps charts from getting too big, hard to understand
  - ANDing and ORing of states: >> to model simultaneously being in >1 state
  - example: elevator in moving/not or doors_open/not
  - Elaborate specification of transition conditions
- Correlation with Activity Charts helps comprehensibility

Add Activities and Actions

- Activities
  - Associated with a state
  - Start when the state is entered
  - Take time to complete
  - Interruptible
- Actions
  - Associated with a transition
  - Take an insignificant amount of time to complete
  - Non-interruptible

Activities and Guards in Statecharts

- Activities
  - An activity can also send an event
- Transitions
  - A transition may have a guard conditions as well as an event specified
  - Transitions can also specify an action that happens in response to the receipt of an event

Statechart with Nested States

Statechart

- Tools to support drawing/changing diagrams
- Tools to support input of textual information through forms/templates
- Diagrams enhanced by use of color (?)
- Tools to generate simulations automatically
  - support "stepping through" the system
- System assures consistency among the diagrams
  - changes automatically depicted consistently in all diagrams
- Tools to automatically generate Ada code that emulate Statechart behavior
Statemate Weaknesses

- Does not seem to scale all that well
  --Hierarchy depicted by nesting all on one 2-dimensional surface
- Data still treated as secondary/incidental
- Focus still on functionality
  --Other characteristics and views are worth thinking about too:
  >>Speed
  >>Implementation approaches and issues
  >>...
UML (Unified Modeling Language): The Latest (?)

- Merger of Booch, Rumbaugh, Jacobsen work
  - "The three amigos"
  - All worked for Rational (now IBM)
- Comprehensive suite of diagrams
- Some semantics in place
  - But not all
  - International task forces (!) working on this
- Process for using them was developed too
  - Rational Unified Process (RUP)
- UML blew away the opposition
  - Not clear this was good

(Some) UML representations

- Class Diagrams
- Use Cases
- Sequence Diagrams
- Package Diagrams
- State Diagrams
- Activity Diagrams
- Collaboration Diagrams
- Deployment Diagrams

Different combinations used by different users for different projects

Major UML Problems/Objections

- What are semantics of all of these features of all of these diagrams?
  - Task forces working on them
  - Maybe there is just too much there (?)
- Diagram semantics overlap
  - Which diagram to use when
  - How to tell when they are inconsistent
- Extensibility
  - Use of "stereotype" feature
  - How to reconcile semantics of new features with existing ones

UML Tries to cover everything

- A diagram type for everything
- But they are not well connected to each other
- Few rules on what to use when
- Long reach with uncertain grasp

Evaluation of Diagrammatic Approach

- Pictures considerably aid clarity
- Significantly reduce possible ambiguity
- Increasingly strong semantics of increasingly intricate pictures yield increasing completeness and increasing assurance of consistency
- Increasingly intricate pictures are decreasingly clear, decreasingly modifiable
  - Modern approach is to provide tools to help
- In place of one intricate and complex diagram, many systems substitute a set of coordinated diagrams, each of which is relatively simple (eg. Statemate)
  - Leads to problems in assuring consistency of diagrams but tools can help here too

BUT ALSO:

- Most diagrams help depict functionality, but not other characteristics, (eg. data, process, etc.)