Dynamic Aspects of UML
Dynamic Aspects of UML

- Additional UML diagrams
  - Collaboration diagram
  - Sequence diagram
  - Activity diagram
  - State diagram

- One or more of these diagrams would be part of a design document which expresses behavior prior to writing code.
In a sequence diagram the objects’ behaviors are shown as **vertical lines** and messages cross between these lines.

The time sequence is shown by vertical position, flowing down the diagram.

Time is not generally to scale.

Bars indicate periods of activity.
Sequence Diagram for Buying Insurance Policy

Jones : Buyer

1: buy(Apex, Thomas, 100k)

Smith : Broker

1.1: buy(Thomas, 100k)

1.2: issue(Thomas, 100k)

Apex : Company

1.3: payCommission()

[Thomas is the insured.]
Added Facets of Sequence and Collaboration Diagrams

- Objects can send messages to **themselves** (e.g. one method implemented using another, including recursion).

- **Pre-conditions** can exist on message arcs: 
  [... pre-condition ...

- Timing can be indicated on the diagram

- Object destruction indicated by large X
Timing Annotation

\[ b - a \leq 500 \text{ ms} \]
Message Types

- No specific detail of control passing

- Synchronous: Nested flow, as in typical method calls; **caller waits** for reply before resuming

- Asynchronous: No explicit reply before sender resumes.
Collaboration Diagrams

- Show objects, not classes, collaborating
- “Message” flows between objects, e.g. a method call:

```
buyer : Buyer

broker : Broker
```

sequence number (dewey decimal) ----> message

1.1: buy(Apex, Thomas, 100k)

note: Arrow does not connect boxes; it shows the direction of flow on an association.
Message Flow for Buying an Insurance Policy

1: buy(Apex, Thomas, 100k)

1.1: buy(Thomas, 100k, Jones)

1.2: issue(Thomas, 100k)

1.3: payComission()
Significance of Numbering

- Numbering shows the sequencing of messages.

- Dewey-Decimal: 1.1, 1.2 are messages introduced in handling of message 1, etc.
Activity Diagrams

- Activity diagrams are extensions of conventional flowcharts.
- Parallel execution is possible.
- Synchronization is representable.
Activity Diagram Example

Print document

Create Postscript file

Send Postscript file

Email confirmation

Parallel Activities

Printer1: Printer

Receive Postscript file

"signal"

Print Postscript file
Synchronization of Activities (fork-join subset of “Petri nets”)
“And” vs. “Or” Synchronization
Branching of an Activity
(same as in traditional flowchart)
“Swimlanes” = Threads or Processes in Activity Diagram

3 Swimlanes
State Diagrams

- Essentially **Statecharts**, as invented by David Harel (founder of I-logix).
- Statecharts are a structured form of finite-state machine.
- States can be decomposed into parallel or nested sub-systems.
- This is more economical than *enumerating* all states.
Every ordinary finite-state machine diagram is a statechart.

States are mutually exclusive.

States may involve program variables not represented by diagram structure.
Events vs. Condition/Action pairs

- Events designate some named event occurring, such as an external event.
- Condition/Action represents an event triggered by a condition on program variables and an action that may assign to those variables.
- The null condition is the same as “true”.
- The null action is that all variables are unchanged.
Hierarchy
Word descriptions signaling that an aggregate state might be appropriate:

- Phase
- Mode
- Region
- Interval
Statechart Example

- Combination FM/AM Radio and Cassette Tape Player (Auto-reverse)
  - Either the radio or the tape player is playing, but not both.
  - The unit can play only if power is on.
  - The tape plays iff a tape is inserted.
  - The radio can play in AM or FM.
Radio/Tape Player State Diagram

Top-Level View

toggle power switch

off -> on

toggle power switch
Radio/Tape Player State Diagram

First-Level Expansion

toggle power switch

off

on

radio mode

tape not inserted

insert tape

tape inserted

eject tape

toggle power switch

tape mode
Radio/Tape Player State Diagram

Second-Level Expansion

- toggle power switch

- off

- on
  - tape not inserted
  - tape is inserted

- radio mode
  - FM
  - AM
  - band toggled

- tape mode
  - forward
  - reverse
  - start reached
  - end reached

- eject tape
- insert tape

- toggle power switch
Possible Design Flaws
Detected from Diagram
Redesign to Remedy Flaws

- Tape mode
  - Forward
    - Direction: fwd
    - Start reached: direction = fwd
    - End reached: direction = rev
  - Reverse
    - Direction: rev
    - Start reached: direction = rev
    - End reached: direction = fwd

- Radio mode
  - FM
    - Band: fm
    - Band toggled: band = fm
  - AM
    - Band: am
    - Band toggled: band = am

- On
  - Tape not inserted
  - Tape inserted
Another Nuance: Concurrency in StateCharts

Dashed line indicates concurrent subsystems
Augment the Radio/Tape Design to Permit Recording from Radio

- The Radio and Tape can now operate concurrently.
- The tape may record in either direction.
- We can no longer let the presence of the tape dictate the mode; must use buttons.
Partially-Developed Radio/Tape Player/Record

- **Radio Mode:**
  - **On:**
    - **Off**
    - **FM**
    - **AM**
    - **Band Toggled**
  - **Off Pressed:**
  - **On Pressed:**

- **Tape Mode:**
  - **Tape Not Inserted**
  - **Insert Tape**
  - **Eject Tape**
  - **Mode: (Per Buttons)**
    - **Play**
    - **Stop**
    - **Record**
  - **Radio Not Off:**
  - **Start Reached**
  - **End Reached**

- **Functions:**
  - **Forward**
  - **Reverse**

- **States:**
  - **Tape Inserted**
Concurrency in StateCharts provide representational economy

- A statechart with concurrency provides for the representation of $M \times N$ states with only $M + N$ bubbles.

- The combinatorial explosion of states is thereby reduced.
Data Flow Diagrams (DFDs)

- Classical software engineering tool
- Not part of UML (yet!)
- Data flow can be loosely handled as objects, but object diagrams are ambiguous as to whether they are representing true data flow.
The main idea:

- Represent system as a directed graph of software modules (nodes)
- Arcs between nodes represent the “flow” of data from one module to another
Data Flow Diagrams

- Applications that lend themselves to this model:
  - “Pipe” composition, as in Unix pipes
  - Problems treatable as functional decompositions on of streams of data
  - File- and document-processing problems: The “flow” is represented by intermediate immediate files that are written by one module and read by another.
Advantage of DFDs

- DFDs have an intuitive feel that is easily understood by the customer (vs. Objects, which might be less intuitive), so could be used as a sort of specification language.
- Good for “assembly-line” processing of commodities or documents.
- Data flow makes programming into plumbing.
DFD for Order Processing

Order details + blank order form → Complete order form → Validate order → Record order → Send to supplier

Order amount + account details → Adjust available budget

Completed order form → Signed order form → Signed order form

Orders file → Budget file

Checked and signed order + order notification

source: Ian Sommerville, Software Engineering, 5th Ed.
More DFD points

- DFD capable of showing various levels of abstraction.
- A processing box can be expanded into its own DFD, etc., enabling “zooming” into the structure of the system.
- Does not particularly lend itself to showing interactive behavior, since output is separated from input.
Data Dictionaries
Data Dictionaries

- Glossary or “database” of names and associated descriptions of entities, associations, attributes used in the system.
- Represents a project-wide shared repository of system information
- Provides:
  - A mechanism for name management. As a system model may be developed by different people, there is potential for name clashes
  - A link from analysis to design and implementation
# Data Dictionary Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>has_labels</td>
<td>1:N relation between entities of type Node or Link and entities of type Label.</td>
<td>Relation</td>
<td>5.10.93</td>
</tr>
<tr>
<td>Label</td>
<td>Holds structured or unstructured information about nodes or links. Labels can be text or can be an icon.</td>
<td>Entity</td>
<td>8.12.93</td>
</tr>
<tr>
<td>Link</td>
<td>Represents a relation between design entities represented as nodes. Links are typed and may be named.</td>
<td>Relation</td>
<td>8.12.93</td>
</tr>
<tr>
<td>name (label)</td>
<td>Each label has a name which identifies the type of label. The name must be unique within the set of label types used in a design.</td>
<td>Attribute</td>
<td>8.12.93</td>
</tr>
<tr>
<td>name (node)</td>
<td>Each node has a name which must be unique within a design. The name maybe up to 64 characters long.</td>
<td>Attribute</td>
<td>15.11.93</td>
</tr>
</tbody>
</table>

source: Ian Sommerville, Software Engineering, 5th Ed
Data Dictionaries

- Ideally a database in their own right
- Remain intact for the lifetime of the project
- Useable by management and developers
- Useable by customer
  - if source-access is provided
  - legacy file formatting
  - as a description of processing entities (view technical details)
DD Examples on the Web

- www.cris.state.nc.us/datafrm.html
- nccs.urban.org/STalmDD.htm

- Use your browser for many other examples
Info in Data Dictionaries

- Can be handled by UML tool, e.g. in class diagrams and their documentation.

- Report generator may be able to give conventional data dictionary format.
SysML

- The SysML (Systems Modeling Language) is a general-purpose modeling language for systems engineering applications that is defined as a dialect (Profile) of UML 2. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems. These systems may include hardware, software, information, processes, personnel, and facilities. SysML was originally developed as an open source specification project initiated in 2003.

- Many systems engineering processes tend to be document-intensive (a.k.a. document centric) and employ a motley mix of diagram techniques that are frequently imprecise and inconsistent. In a manner similar to how software engineers sought a general-purpose modeling language (UML) to precisely specify software-intensive systems during the last decade, systems engineers are now seeking a domain-specific modeling language to specify complex systems that include non-software components (e.g., hardware, information, processes, personnel, and facilities). UML cannot satisfy this need because of its software bias; hence the motivation for SysML. Even though SysML is based on UML, it reduces UML's size and software bias while extending its semantics to model requirements and parametric constraints. These latter capabilities are essential to support requirements engineering and performance analysis, two essential systems engineering activities.
SysML vs. UML

- Activity Diagram
- Sequence Diagram
- State Machine Diagram
- USE Case Diagram
- Block Definition Diagram
- Internal Block Diagram
- Package Diagram

- Same as UML 2
- Modified from UML 2
- New diagram type

SysML Diagram

Behavior Diagram

Requirement Diagram

Structure Diagram

Parametric Diagram
**SysML vs. UML**

- **In SysML, but not UML:**

<table>
<thead>
<tr>
<th>Diagram Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parametric diagram</td>
<td>Show parametric constraints between structural elements. Useful for performance and quantitative analysis.</td>
</tr>
<tr>
<td>Requirement diagram</td>
<td>Show system requirements and their relationships with other elements. Useful for requirements engineering.</td>
</tr>
<tr>
<td>Allocation tables*</td>
<td>Show various kinds of allocations (e.g., requirement allocation, functional allocation, structural allocation). Useful for facilitating automated verification and validation (V&amp;V) and gap analysis.</td>
</tr>
</tbody>
</table>

* dynamically derived tables, not really a diagram type
SysML vs. UML

- In UML, but not SysML:

  - Component diagram
  - Communication diagram
  - Deployment diagram
  - Interaction overview diagram
  - Object diagram
  - Timing diagram