Where Have We Been?

- Scheduling - Periodic Tasks
- Scheduling - Sporadic Tasks
- Communication and Synchronization

What Now?

- Software Engineering
  - Software Life Cycle
  - Specification Methods
    - Flowcharts
    - State Transition Diagrams
    - Data Flow Diagrams
    - Structure Charts
    - Petri Nets
Software Life Cycle (Software Engineering)

- Ideal Waterfall Model

  Concept

  Requirements

  Design

  Programming

  Test

  Maintenance
Software Life Cycle Phases

• **Concept Phase**
  • need and goals
    • management directive, customer input, technology, etc.
    • features and feasibility (design and testing)

• **Requirements Phase**
  • Software Specification
    • timing, accuracy, user interface, etc.
    • software/hardware interfaces
    • testing plan
    • budget and schedule
  • Functional and Non-functional requirements
    • Complete, correct, consistent, testable
Software Phases (cont.)

- **Design Phase**
  - Convert Requirements to Detailed Design Spec.
  - Module partitioning

- **Programming Phase**
  - Convert to code, incremental test, documentation

- **Test Phase**
  - Formal test cases, meets the spec?

- **Maintenance Phase**
  - Customer support, bug fixes
The Real Software Life Cycle

Concept Phase

- Cannot Specify
- Feature not Feasible
- Error Detected in Concept

Maintenance Phase

- New Product
- Validate suspected problem
- Error Detected in Concept

Requirements Phase

- Feature not Feasible
- Error Detected in Requirements

Testing Phase

- Error in Design
- Error in Coding

Design Phase

- Feature not Feasible
- Design not Feasible
- Feature not Feasible
Software Specification and Design

- **Natural Languages**
  - Required, but should be redundant

- **Mathematical Specification**
  - Precise, maps to code, provable (?)

- **Flowcharts**
  - Well understood, but promotes use of GOTO
  - Parallel flow interaction not easily represented
  - Timing issues difficult to represent
When started, the print spooler must first initialize a queue of print objects (file pointers) and check the print queue backup file in case there was a system crash or other exit condition. If the backup file is not empty, load the queue from the backup file, maintaining order. If the queue could not be initialized, alert operator and exit. If the printer is not busy, set the busy flag to false. Assume a semaphore mechanism exists and block. (This could be a polling loop testing some pre-defined operating system flags.)

If it unblocks on a request to print, queue the job and update backup file. If printer is not busy, send the job to the printer, else block. If the queue is full, alert the requestor and block with no change to queue.

If it unblocks on printer done, check the queue and either send the next job and update backup file or set busy false. If printer error, alert operator. All cases block.

If unblock on exit request, quit.
State Transition Diagrams

• State (circle)
  • Represents “equilibrium points” of the system

• Transitions (arrow)
  • Changes of state
  • Occurs when “condition” is met
  • “Actions” are taken during transition

• System must have Discrete States

• Determine What Events Cause Changes in State

  Mealy = output during transition
  Moore = output within state
State Transition Diagram (Print Spooler)

- **Start**
  - Transition to **Init**
  - Transition to **Wait**

- **Init**
  - Transition to **Start**
  - Transition to **Wait**
  - Transition to **Queue**
  - Transition to **Print**
  - Transition to **Exit**

- **Wait**
  - Transition to **Init** with "success" label
  - Transition to **Wait** with "error" label
  - Transition to **Queue** with "queued, busy" label
  - Transition to **Print** with "sent, busy = true" label
  - Transition to **Exit** with "exit request" label

- **Queue**
  - Transition to **Wait** with "print request" label
  - Transition to **Init** with ""queue error" label

- **Print**
  - Transition to **Wait** with "no job, busy = true" label
  - Transition to **Queue** with "queued, printer not busy" label
  - Transition to **Exit** with "printer done" label

- **Exit**
Structure Charts

- “Calling Trees”
- Left-to-Right implies: Execution order
- Top-to-Bottom implies: Increasing detail
- o - denotes conditional execution
- * - denotes repetitive execution
- Represents program flow, not data flow

Example in [PL] text is poor
Structure Chart (Beverage Vending)

Dispense Beverage * on Demand

Accept coins
Accept choice
Dispense cup
Make Beverage

Brew Beverage
 Dispense Additives

Coffee 0
Decaf 0
Tea 0

Dispense Grounds
Dispense Hot H2O
Mix
Where Have We Been?

- Scheduling Algorithms
- Communication and Synchronization
- Software Engineering ([PL] Chapter 5)
  - Software Life Cycle
  - Software Specification
    - Natural Language
    - Flowcharts
    - State Transition Diagrams (FSM)
    - Structure Charts

Where are we headed?

- More Software Engineering
  - Software Specification
    - Dataflow Diagrams
    - Petri Nets
    - Statecharts
The View from Above

- Informal Specification Methods
  - Intuitive to non-technical people
  - Difficult to prove anything
    - Natural Language
    - Flowcharts
    - Structure Charts

- Semiformal Specification Methods
  - Easy to learn, but requires training
  - Some rigor
    - Dataflow Diagrams ([PL] construction)

- Formal Specification Methods
  - Difficult to learn
  - Rigorous and provable
    - State Transition Diagrams
    - Petri Nets
    - Statecharts
Dataflow Diagrams

- Analyze Data Flow Through System and Determine Major Functions
  - Emphasizes Flow of Data
  - Shows Concurrency Explicitly
  - Does Not Show Synchronization (implicit at best)

Diagram:
- Process
- Sink/Source
- Data Store
- Data Flow

(Data Transformation)
DFD Example [PL]

- Accel and Gyro Compensation are Concurrent
- Synchronization of Comp. Data is only Implied
- Hierarchies of DFDs Provide Process Details
- Describes What the System Does, not When
• Fixed some bugs in the [PL] diagram (magenta)
  • Use of Data Stores more closely match [JC] definition
  • Eliminated redundant stores
  • Acceleration input to “Torque Gyros” matches natural language spec.
• **Added Likely Information to Better Justify Data Stores**
  
  • This info is not explicitly in the spec but it makes the meaning clearer (left out to simplify the example)
  
  • Reinforces their existence as true Data Stores (not just temporary variables)
Control Flow Diagrams

- Analyze Control Flow (Events and Actions) and Determine Their Dynamics
  - Emphasizes Flow of Control
  - Control Transformations Represented by Attached State Transition Diagrams
  - Dynamic Behavior is Explicitly Indicated

Control Transformation  Events and Actions  Control Store
Real-Time DFD

• Link CFD with DFD

• Introduce Synchronization Prompts
  • Enable/Disable Prompt - on until off’ed
  • Trigger Prompt - one shot
  • Activator Prompt - active for duration of state

    L ← prompt label (E,T,A)

    (Note double-arrow meant continuous time in DFD)

• Prompts Couple CFD to DFD
Real-Time DFD Example

- **Accel**: Raw Accel to Comp. Accel
  - Comp. Accel
  - Acceleration

- **Comp. Gyro**: Raw Gyro to Comp. Gyro
  - Comp. Gyro
  - Gyros

- **Gyros**: Compensate Gyro

- **Control Transform**: 5ms Tick to Update Disp.

- **Calc. Pos.**: Compensate Accel
  - Calculate Position

- **Update Disp.**: Torque Gyros

- **STDs**: Screen Data

- **Display**: Position, Attitude
  - Update Display
  - Screen Data

- **Gyros**: Gyro Data
  - Gyro Cmds
  - Gyro

- **Position Attitude**: Comp. Gyros
  - Position, Attitude
  - Pos, Att

Software Engineering - 21
STDs for Real-Time DFD Example

- Other Topologies Possible
Petri Nets

- Formal Method for Multitasking and Synchronization

- “Initial Marking” Indicates How the Net Starts

- Transitions “Fire” if all Inputs have a Token

- Tokens are Consumed and Produced by Transitions
Petri Net Example [PL]

Table 1: Firing Table

<table>
<thead>
<tr>
<th>epoch</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>m₀</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>m₁</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>m₂</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- All Transitions are Synchronized
- Transitions Triggered by Event such as Clock
Multiple Transitions Example [PL]

- Multiple Transitions are Enabled Simultaneously
- Evaluate All Transitions to See Which Fire
Petri Nets Can Be Non-Deterministic

- What Transitions are Enabled?
- Can Both T2 and T3 Fire?
- Beware the Errant Token
• Complex Multiplication: \((a+bi)(a+bi) = a^2 - b^2 + 2abi\)
  • Tokens: intermediate results (variables)
  • Transitions: operations
  • Creation of tokens tied to external event
Escort Tokens and Inhibitors

- **Basic production facility** *(conveyor contention)*

- **Escort token to eliminate conveyor contention**
Escort Tokens and Inhibitors (cont.)

- Inhibitor to protect only conveyor (not entire station)

- To deal with finite time transitions, add “done” indicator that produces a token
Elements of Statecharts

- Finite State Automata
- Depth - states inside states (hierarchical)
- Orthogonality - concurrent processes
- Broadcast Communications - synchronization
Unified Modeling Language

- Aimed at Object-Oriented Systems Design
- Independent of Design Process
  - Modeling Language (semantics and notation)
  - Process of Application of that Language
- UML’s Characteristics
  - Semi-formal Language
  - Discrete Systems (vs. continuous)
    - based on finite state machines

Semantics ==> “meaning”
Syntax ==> representation of semantics
UML Components

• 3 Distinct Models
  • Requirements Model
    • Black box, hiding implementation
    • Actors and use cases (external to system)
  • Structural Model
    • Classes and how they collaborate
    • Associations among classes
    • Components (executables) and nodes (CPUs, I/O, etc)
  • Behavioral Model
    • Statecharts

• Refs:
  • Real-Time UML, Douglass, Addison Wesley