Specification example

- An executable specification-language enables:
  - Early verification
  - Precision
  - Automation
  - Documentation

- A good language/model match reduces:
  - Capture time
  - Comprehension time
  - Functional errors
Outline

- Capture an example’s model in a particular language
  - PSM model in the SpecCharts language

- Point out the benefits of a good language/model match

- Highlight experiments that demonstrate those benefits
Answering machine controller’s environment
Highest-level view of the controller

Controller

SystemOff

power='0'

SystemOn

power='1'
The SystemOn behavior

- System usually responds to the line
- Pressing any machine button gets immediate response
The RespondToMachineButton behavior

```
behavior RespondToMachineButton
type code is
begin
  if (play='1') then
    HandlePlay;
  elsif (fwd='1') then
    HandleFwd;
  elsif (rew='1') then
    HandleRew;
  elsif (memo='1') then
    HandleMemo;
  elsif (stop='1') then
    HandleStop;
  elsif (hear_ann='1') then
    HandleHearAnn;
  elsif (rec_ann='1') then
    HandleRecAnn;
  elsif (play_msgs='1') then
    HandlePlayMsgs;
  end if;
end;
```

(a) (b)
The RespondToLine behavior

- Monitors line for rings
- Answers line
- Responds to exceptions
  - Hangup
  - Machine turned off

![Diagram showing RespondToLine behavior]

ResponseToLine

Monitor

Answer

rising(hangup)

falling(machine_on)
The Monitor behavior

- Counts for required rings
- Requirements may change

Monitor

```plaintext
signal rings_to_wait : integer range 1 to 20 := 4;
function DetermineRingsToWait return integer is begin
  if ((num_msgs > 0) and (tollsaver='1') and (machine_on='1')) then
    return(2);
  elsif (machine_on='1') then
    return(4);
  else
    return(15);
  end if;
end;
```

```plaintext
variable I : integer range 0 to 20;
i := 0;
while (i < rings_to_wait) loop
  wait on tollsaver, machine_on;
  if (rising(ring)) then
    i := i + 1;
  end if;
end loop;
```

MaintainRingsToWait

CountRings

```plaintext
loop
  rings_to_wait <= DetermineRingsToWait;
  wait on tollsaver, machine_on;
end loop;
```

```plaintext
variable I : integer range 0 to 20;
i := 0;
while (i < rings_to_wait) loop
  wait on rings_to_wait, ring;
  if (rising(ring)) then
    i := i + 1;
  end if;
end loop;
```
The Answer behavior

behavior PlayAnnouncement type code is begin
    ann_play <= '1';
    wait until ann_done = '1';
    ann_play <= '0';
end;

behavior RecordMsg type code is begin
    ProduceBeep(1 s);
    if (hangup = '0') then
        tape_rec <= '1';
        wait until hangup='1' for 100 s;
        ProduceBeep(1 s);
        num_msgs <= num_msgs + 1;
        tape_rec <= '0';
    end if;
end;
The RemoteOperation behavior

- Owner can operate machine remotely by phone
- Owner identifies himself by four button ID

```vhdl
behavior CheckUserCode type code is
    begin
        code_ok <= true;
        for (i in 1 to 4) loop
            wait until tone /= "1111" and tone'event;
            if (tone /= user_code(i)) then
                code_ok <= false;
            end if;
        end loop;
    end;
```

**Diagram:**
(a) RemoteOperation
- Owner identifies himself by four button ID
- RespondToCmds

(b) Behavior CheckUserCode
- Code is validated by checking the entered code against the stored user codes.
The answering machine controller specification

**Controller**

- **SystemOff**
  - power= '1'
  - power= '0'

- **SystemOn**
  - InitializeSystem
  - RespondToMachineButton
    - rising(any button pushed)

- **RespondToLine**
  - Monitor
    - rising(hangup)
  - falling(machine on)

- **Answer**
  - PlayAnnouncement
    - tone= '0001'
  - RecordMsg
    - rising(hangup)
  - Hangup

- **RemoteOperation**
  - CheckUserCode
    - code ok
    - not code ok
  - RespondToCmds
    - tone= '0010'
    - ResetTape
      - other
    - MiscCmds
      - hangup= '1'
      - other

- **Controller**
  - **Announcement unit**
  - **Tape unit**
  - **Line circuitry**

- **Phone line**
  - **Announcement**
  - **Tape**
  - **Messages**
    - **Stop**
    - **Rew**
    - **Play**
    - **Fwd**
    - **Mic**
    - **Power**
    - **Record**
    - **Ann Done**
    - **Toll Saver**
    - **Hangup**
    - **Off Hook**
    - **Ring**
    - **Tone**

- **MiscCmds**
  - **ResetTape**
    - **Tape Fwd**
    - **Tape Rec**
    - **Tape Rew**
    - **Tape Play**
    - **Tape Cnt**
    - **Monitor**
      - rising(hangup)
      - falling(any button pushed)

- **System On**
  - **InitializeSystem**
  - **RespondToMachineButton**
    - rising(hangup)

- **Toll Saver**
  - **Hangup**
  - **Off Hook**
  - **Ring**
  - **Tone**

**Specifications example**
Executable specification use

- Precision
  - Readability/precision compete in a natural language
  - Executable specification encourages precision
  - Designer asks questions, specification answers them

- Language/model match (SpecCharts/PSM):
  - Hierarchy
  - State-transitions
  - Programming constructs
  - Concurrency
  - Exceptions
  - Completion
  - Equivalence of states and programs
Specification capture experiment

<table>
<thead>
<tr>
<th></th>
<th>VHDL</th>
<th>SpecCharts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average specification–time in minutes</td>
<td>40</td>
<td>16</td>
</tr>
<tr>
<td>Number of modelers</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Number of incorrect specifications first time</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Number of incorrect specifications second time</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- VHDL modelers required 2.5 times longer
- Two VHDL specifications possessed control errors
- SpecCharts were effective for state-transitions and exceptions
Comparison of SpecCharts, VHDL and Statecharts

Answering machine example

<table>
<thead>
<tr>
<th>Specification attributes</th>
<th>Conceptual model</th>
<th>SpecCharts</th>
<th>VHDL (hierarch.)</th>
<th>VHDL (flat)</th>
<th>Statecharts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program−states</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>32</td>
<td>80</td>
</tr>
<tr>
<td>Arcs</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>152</td>
<td>135</td>
</tr>
<tr>
<td>Control signals</td>
<td>0</td>
<td>84</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Lines/leaf</td>
<td>—</td>
<td>7</td>
<td>27</td>
<td>29</td>
<td>—</td>
</tr>
<tr>
<td>Lines</td>
<td>446</td>
<td>1592</td>
<td>963</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Words</td>
<td>1733</td>
<td>6740</td>
<td>8088</td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>

| Shortcomings             |                   |            |                  |             |             |
|--------------------------|                   |            |                  |             |             |
| No sequential program constructs |                   |            |                  | X           |             |
| No hierarchy             |                   | X          | X                |             |             |
| No exception constructs  |                   | X          | X                |             |             |
| No hierarchical events   |                   |            |                  | X           |             |
| No state−transition constructs |                   |            |                  | X X         |             |
### Design quality experiment

<table>
<thead>
<tr>
<th>Design attribute</th>
<th>Designed from English</th>
<th>Designed from SpecCharts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control transistors</td>
<td>3130</td>
<td>2630</td>
</tr>
<tr>
<td>Datapath transistors</td>
<td>2277</td>
<td>2251</td>
</tr>
<tr>
<td>Total transistors</td>
<td>5407</td>
<td>4881</td>
</tr>
<tr>
<td>Total pins</td>
<td>38</td>
<td>38</td>
</tr>
</tbody>
</table>

- No loss in design quality with an executable language
Summary

- Executable languages encourage precision and automation

- The language should support an appropriate model
  - Makes specification easy

- Strongly parallels programming languages
  - Structured vs. assembly languages
  - Object-oriented model and C++