Automated Specification-Based Test Case Generation Using SCR

JPL IT Symposium
November 4, 2002

Allen Nikora
Quality Assurance Section
Jet Propulsion Laboratory

Constance L. Heitmeyer
Head, Software Engineering Section
Naval Research Laboratory

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology. This work was sponsored by the Software Engineering Technology element of JPL’s Center for Space Mission Information and Software Systems.
Automated Specification-Based Test Case Generation Using SCR

JPL IT Symposium
November 4, 2002

Allen Nikora
Quality Assurance Section
Jet Propulsion Laboratory

Constance L. Heitmeyer
Head, Software Engineering Section
Naval Research Laboratory
Agenda

- Overview
- Approach
- Work Accomplished
  - SCR Specification of FPE
  - Simulation of FPE, graphical interface for simulator
  - Test Cases
- Conclusion
- Future Work
Overview

- Generating test cases becomes increasingly difficult as the complexity of mission software increases.
  - Overlooked or misinterpreted requirements
  - Misunderstood or omitted interactions between requirements
- Until recently, available test case generators haven’t operated on specifications that can be analytically verified
  - AETG and other orthogonal test case generators create test cases based on a “specification” that describes a system’s functionality in terms of combinations of parameter values
  - Test Master and other EFSM-based generators
- Model checking techniques can be used to create test cases from a formal specification that can be analytically verified
  - Test cases based on specifications that exhibit desired functionality and satisfy desired properties
  - Test cases cover all possible executions described by the specification
- Goal – pilot use of a model-checking based test case generator on a “real” FSW component
Approach

- Identify test case generator
  - T-VEC
  - SCR

- Identify collaborating efforts
  - Fault Protection Engine for DI, Starlight

- Acquire and install SCR
  - Specification Editor
  - Simulator
  - Model Checker
  - Test Case Generator

- Transcribe the requirements for the selected areas into the SCR notation

- Use SCR test case generator to produce the test cases from the SCR specification

- Evaluate the test cases
Work Accomplished

- SCR Specification
- SCR Simulation
- Test Cases
SCR Specification Overview

- SCR specification of Fault Protection Engine based on:
  - Stateflow diagrams for FP engine at (http://alab.jpl.nasa.gov/FaultProtection.htm)
  - DI FP Engine design documentation – available at Deep Impact web site (deep-impact) in FP System Engineering area
SCR Specification Overview (cont’d)

• Simplifying Abstractions
  – No subresponses
  – Significantly smaller number of responses of each type than in “real” spacecraft
    • 3 non-interrupting
    • 3 interrupting
    • 2 ground requested
    Preserves interactions between response requests.
  – Simplified response deferral mechanism
SCR Specification Overview (cont’d)

STD for FPE specification

Date: 4 November, 2002

Software Engineering Technology
### SCR Specification Overview (cont’d)

<table>
<thead>
<tr>
<th>Begin State</th>
<th>Statechart</th>
<th>Transition Event</th>
<th>SCR Specification</th>
<th>Transition Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Run_Response</td>
<td>Received request to run a response OR there are one or more deferred responses</td>
<td>Idle</td>
<td>Run_Resp_NoWP</td>
</tr>
<tr>
<td></td>
<td>Idle</td>
<td>Received request to flush all responses</td>
<td>Run_Resp_NoWP, Run_Resp_WP, Run_Int_Resp</td>
<td>Received request to flush all responses</td>
</tr>
<tr>
<td>Run_Response, Run_IntERRUPTING_Response</td>
<td>Idle if stack is empty, Run_Response if stack is not empty</td>
<td>-</td>
<td>Run_Resp_NoWP</td>
<td>Idle</td>
</tr>
<tr>
<td>Run_Response</td>
<td>WayPoint</td>
<td>Waypoint encountered in response</td>
<td>Run_Resp_NoWP</td>
<td>Run_Resp_WP</td>
</tr>
<tr>
<td>NoWayPoint</td>
<td>WayPoint</td>
<td></td>
<td>Run_Resp_NoWP</td>
<td>Run_Resp_WP</td>
</tr>
</tbody>
</table>

### Comparison of SCR Specification to FPE statechart
### SCR Specification Overview (cont’d)

<table>
<thead>
<tr>
<th>Begin State</th>
<th>Statechart</th>
<th>Transition Event</th>
<th>SCR Specification</th>
<th>Transition Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Run_Resp_NoWP</td>
<td>Run_Int_Resp</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Run_Interru</td>
<td>WayPoint</td>
<td>Interrupting or</td>
<td>Run_Int_Resp</td>
<td>Run_Resp_WP</td>
</tr>
<tr>
<td>ting_Resp-</td>
<td></td>
<td>ground requested</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>response completes.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Run_Int_Resp</td>
<td>Run_Resp_NoWP</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Waypoint encountered in response AND there are one or more deferred interrupting or ground-requested responses.

Interrupting or ground requested response completes prior to expiration of waypoint.

Waypoint expired prior to completing interrupting or ground requested response.

### Comparison of SCR Specification to FPE statechart (cont’d)
### SCR Specification Overview (cont’d)

<table>
<thead>
<tr>
<th>Begin State</th>
<th>Statechart</th>
<th>Transition Event</th>
<th>SCR Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>WayPoint</td>
<td>NoWayPoint</td>
<td>Waypoint has expired.</td>
<td>Run_RESP_WP</td>
</tr>
<tr>
<td></td>
<td>Run_Interrupting_Response</td>
<td>Request for interrupting or ground requested response received OR there are one or more deferred interrupting or ground-requested responses.</td>
<td>Run_RESP_WP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Begin State</th>
<th>End State</th>
<th>Transition Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run_RESP_WP</td>
<td>Run_RESP_NoWP</td>
<td>Waypoint has expired.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Begin State</th>
<th>End State</th>
<th>Transition Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run_RESP_WP</td>
<td>Run_Int_RESP</td>
<td>Request for interrupting or ground requested response received.</td>
</tr>
</tbody>
</table>

### Comparison of SCR Specification to FPE statechart (cont’d)

Date: 4 November, 2002
SCR Specification Overview (cont’d)

<table>
<thead>
<tr>
<th>Statechart</th>
<th>SCR Specification</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Monitored Variables</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IntResp</td>
<td>mRespRequest</td>
<td>mRespRequest is a variable whose value is a three-digit number. The least significant digit represents the ID of a non-interrupting response, the next least significant digit represents the ID of an interrupting response, and the most significant digit represents the ID of a ground-request response. These could have been specified as three separate monitored variables. Since more than one response can be requested at any given time, however, specifying the variable in this manner simplified the specification.</td>
</tr>
<tr>
<td>NonIntResp</td>
<td>mRespRequest</td>
<td>See above</td>
</tr>
<tr>
<td>ReqResp</td>
<td>mRespRequest</td>
<td>See above</td>
</tr>
<tr>
<td>IsDone</td>
<td>MrespDone</td>
<td>A signal indicating that the currently executing response has completed. In the SCR specification, this signal is viewed as coming from the sequencer that actually executes the instructions within a response. The functionality and behavior of the sequencer are not included in the SCR specification.</td>
</tr>
<tr>
<td>FlushAll</td>
<td>mFlushAllResps</td>
<td>A signal to the FPE to terminate the currently-executing response and cancel all deferred response requests.</td>
</tr>
</tbody>
</table>

Monitored and Controlled Variables
SCR Specification Overview (cont’d)

<table>
<thead>
<tr>
<th>Statechart</th>
<th>SCR Specification</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitored Variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnterWayPoint</td>
<td>mWayPoint</td>
<td>A signal to the FPE indicating that a (non-interrupting) response has encountered a waypoint. In the SCR specification, this is viewed as a signal from the sequencer actually executing the response’s instructions.</td>
</tr>
<tr>
<td>ExitWayPoint</td>
<td>mTimeOut</td>
<td>These data items signal the end of a waypoint within a (non-interrupting) response. To make the timeout more visible, we defined separate signals for entering a waypoint and waypoint timeout.</td>
</tr>
<tr>
<td>Controlled Variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RespInit</td>
<td>cResp_Request</td>
<td>This variable indicates the ID and type of the request that should be executed next. In the SCR specification, the variable is represented as a three-digit non-zero number, where exactly one digit is non-zero, the position of the non-zero digit indicates the response type, and the digit value indicates the response ID.</td>
</tr>
</tbody>
</table>

Monitored and Controlled Variables (cont’d)
SCR Specification Overview (cont’d)

Logic of the FPE Specification

- The term \( t_{\text{Current}_\text{Req_ID}} \) is assigned a value as follows:
  
  - If the FPE is in \( \text{Idle} \) or \( \text{Run Resp WP} \) modes and one or more new requests for high-priority responses are received (indicated by a change in \( m_{\text{Request Resp}} \) and either of \( t_{\text{GR_ID}} \) and \( t_{\text{IR_ID}} \) is non-zero), then
    
    - \( t_{\text{Current}_\text{Req_ID}}' = t_{\text{GR_ID}} \) if \( t_{\text{GR_ID}} \) is non-zero
    - \( t_{\text{Current}_\text{Req_ID}}' = t_{\text{IR_ID}} \) if \( t_{\text{GR_ID}} \) is zero and \( t_{\text{NR_ID}} \) is non-zero
  
  - If the FPE is in \( \text{Idle} \) mode and only a new request for a non-interrupting response is received (indicated by a change in \( m_{\text{Request Resp}} \) and both \( t_{\text{GR_ID}} \) and \( t_{\text{IR_ID}} \) are zero while \( t_{\text{NR_ID}} \) is non-zero), then
    
    - \( t_{\text{Current}_\text{Req_ID}}' = t_{\text{NR_ID}} \)
  
  - If the FPE is in \( \text{Run Resp NoWP} \) mode and if the currently executing response is completed or a waypoint is encountered when the currently executing response is a non-interrupting response, then
    
    - \( t_{\text{Current}_\text{Req_ID}}' \) is assigned the ID of the longest deferred request in the queue of ground requests if the queue is non-empty \( (t_{\text{GRq_len}} > 0) \)
    - \( t_{\text{Current}_\text{Req_ID}}' \) is assigned the longest deferred element of the queue of interrupting requests if the queue is non-empty \( (t_{\text{IRq_len}} > 0) \) and if the queue of ground requests is empty \( (t_{\text{GRq_len}} = 0) \)
SCR Specification Overview (cont’d)

Logic of the FPE Specification (cont’d)

- The term tCurrent_Req_ID is assigned a value as follows:
  - If the FPE is in Run_Resp_NoWP mode and the currently executing response is completed, then
    - tCurrent_Req_ID’ is assigned the ID of the longest deferred element of the queue of non-interrupting requests if the queue is non-empty (tNRq_len > 0) and if the other queues are empty
  - If the FPE is in Run_Int_Resp mode and the currently executing response is completed and the time-out has not expired, then
    - tCurrent_Req_ID’ is assigned the longest deferred element of the queue of ground requests if the queue is non-empty (tGRq_len > 0)
    - tCurrent_Req_ID’ is assigned the longest deferred element of the queue of interrupting requests if the queue is non-empty (tIRq_len > 0) and the queue of ground requests is empty (tGRq_len = 0)
  - If the FPE is in Run_Int_Resp mode and the currently executing response is completed and the time-out has expired, then
    - tCurrent_Req_ID’ is assigned the ID (saved earlier in tSaveNR_ID) of the non-interrupting request whose execution was postponed by a waypoint
SCR Specification Overview (cont’d)

Logic of the FPE Specification (cont’d)

- The term tCurrent_Req_ID is assigned a value as follows:
  - If an input to FlushAll is received (mFlushAllResps becomes true), or if the FPE is in Run_Resp_NoWP mode and all queues are empty, or if the FPE is in Run_Resp_NoWP mode and a waypoint is encountered when the high priority queues are empty, or if the FPE is in Run_Int_Resp mode and the high-priority queues are empty and the time-out has not expired, then
  - tCurrent_Req_ID’ is assigned the value zero (no new or deferred response request is available).

- The term tCurrent_Req_Type is assigned values using the same logic. The value of the controlled variable cResp_Request is computed using the values of tCurrent_Req_ID and tCurrent_Req_Type.
SCR Specification Overview (cont’d)

- Final version of specification can be found in the “SCR Specifications for Fault Protection Engine” row at http://eis.jpl.nasa.gov/~anikora/WPAs_and_Task_Descriptions/SCR-Spec-Based-Testing-attachment.html#ControlledRecords
  - Current specification is at the end of the list.
  - Specification opens as text file using Notepad, Word, etc.
- Walk through final specification using SCR tool
- Specification also included as Appendix A in final report
Simulator Overview

- SCR toolset includes facilities for generating a simulation for a specification
- Created a simulation of the FPE specification to better understand FPE behavior
- If time allows, four scenarios will be demonstrated
  - One Non-Interrupting, One Interrupting Response
  - Two Non-Interrupting Responses
  - One Non-Interrupting, Two Interrupting Responses
  - One Non-Interrupting, Two Interrupting, Two Ground-Requested Responses
Simulator Overview (cont’d)

FPE Simulator GUI
Test Cases Overview

- Generated according to mode transition table defined in specification
- Test cases expressed in terms of externally-visible inputs and outputs
- Test cases cover all transitions defined in mode transition table
  - Nominal behavior
  - Some error behavior
Extending SCR To Automatic Test Set Generation

Our approach to software testing:
- specification-based
- blackbox—does the software satisfy the requirements specification?
Some Basics

Test Case: Sequence of inputs, each paired with a set of outputs
Test Suite: A collection of test sequences

Goals of Test Set Generation:

- The number of test cases in the test suite should be as small as possible
- The test suite should “cover” all errors that any implementation may contain

Our approach to generating test cases: Use a model checker

- To construct the test input data (sequence of inputs)
- As an oracle--given a sequence of inputs, we use the model checker to compute the set of expected outputs

How to generate counterexamples?
USE TRAP PROPERTIES
Constructing Trap Properties From System Properties

Suppose the SCR spec of the FPE satisfies the following property:

\[
@T(m\text{FlushAllResps}) \text{ WHEN } FPE\text{Mode} = \text{Run}\_\text{Int}\_\text{Resp} \Rightarrow FPE\text{Mode}' = \text{Idle}
\]

How do we generate a test sequence from this property?

- Translate the SCR specification into the lang. of a model checker, say Spin
- Translate the negation of the above property's hypothesis into Promela

\[
\text{NOT ( } \overline{@T(m\text{FlushAllResps}) \text{ WHEN } FPE\text{Mode} = \text{Run}\_\text{Int}\_\text{Resp}} \text{ )}
\]

The above property is an example of a TRAP PROPERTY
Example: The Mode Transition Table From The SCR Spec Of The FPE

<table>
<thead>
<tr>
<th>Old Mode</th>
<th>Event</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>@C(mResp_Request) AND...</td>
<td>Run_Resp_NoWP</td>
</tr>
<tr>
<td>Run_Resp_NoWP</td>
<td>@T(mWayPoint) when tCurrentReqType=NR and tIRq_len=0 and tGRq_len=0</td>
<td>Run_Resp_WP</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Run_Resp_WP</td>
<td>@T(mFlushAllResps)</td>
<td>Idle</td>
</tr>
</tbody>
</table>

*Table Defining the Value of FPEMode*
Example: The Mode Transition Table From The SCR Spec Of The FPE (cont’d)

if $FF\_Mode = Idle$
    $\land \neg @C(mResp\_Request) \land \ldots$  $\rightarrow$  $FPE\_Mode' = Run\_Resp\_NoWP$
    $\neg FPE\_Mode = Run\_Resp\_NoWP$  $\rightarrow$  $FPE\_Mode' = Run\_Resp\_WP$
    $\land @T(mWayPoint)$ when $tCurrentReqType=NR \land tIRq\_len=0 \land tGRq\_len=0$

$\land FPE\_Mode = Run\_Resp\_WP$
    $\land @T(mFlushAllResps)$  $\rightarrow$  $FPE\_Mode' = Run\_Resp\_NoWP$
$\neg (else)$  $\rightarrow$  $FPE\_Mode' = FPE\_Mode$

fi

Total function that the table defines (single else clause)
Constructing Test Cases From A Mode Transition Table (1)

Alternate Representation of the Function with the else Clause Distributed

if
  FPEMode = Idle
  if
    \( C(m_{\text{Resp\_Request}}) \) & ... \( \rightarrow \) FPEMode' = Run\_Resp\_NoWP \( C1 \)
    \( (\text{else}) \)
  \( \rightarrow \) FPEMode' = FPEMode \( C1\text{else} \)
fi

if
  FPEMode = Run\_Resp\_NoWP
  if
    \( C(m_{\text{Resp\_Done}}) \) & ... \( \rightarrow \) FPEMode' = Idle \( C2 \)
    \( @T(\text{WayPoint}) \) & ... \( \rightarrow \) FPEMode' = Run\_Resp\_WP \( C3 \)
    \( @T(\text{WayPoint}) \) & ... \( \rightarrow \) FPEMode' = Run\_Int\_Resp \( C4 \)
    \( (\text{else}) \)
  \( \rightarrow \) FPEMode' = FPEMode \( C2\text{else} \)
fi

if
  FPEMode = Run\_Resp\_WP
  if
    ... \( \rightarrow \) FPEMode' = ...
    \( (\text{else}) \)
  \( \rightarrow \) FPEMode' = FPEMode
fi

if
  FPEMode = Run\_Int\_Resp
  if
    ... \( \rightarrow \) FPEMode' = ...
    \( (\text{else}) \)
  \( \rightarrow \) FPEMode' = FPEMode
fi
Constructing Test Cases From A Mode Trans. Table (2)

- Each part of the function definition is called a case
- Each case defines a set of state transitions
- Because each function is total, the set of test cases cover the entire state space
- Because the cases are mutually exclusive, each case is an equivalence class of system executions with the same two final states

For example, case $C1$ defines the set of executions whose final two states satisfy the following property:

\[
\text{FPEMode} = \text{Idle} \land \@C(\text{mResp Request}) \land \ldots \\
\implies \text{FPEMode'} = \text{Run Resp NoWP}
\]
## Test Cases Overview (cont’d)

<table>
<thead>
<tr>
<th>Source Mode</th>
<th>Events</th>
<th>Destination Mode</th>
<th>Test Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>@C(mResp_Request) AND (mResp_Request’ &gt; 0 AND ((tNR_ID’ &gt; 0 AND tNR_ID’ &lt;= MaxID) OR (tIR_ID’ &gt; 0 AND tIR_ID’ &lt;= MaxID) OR (tGR_ID’ &gt; 0 AND tGR_ID’ &lt;= MaxID))) ELSE</td>
<td>Run_Resp_NoWP</td>
<td>C1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Idle</td>
<td>C1else</td>
</tr>
<tr>
<td>Run_Resp_NoWP</td>
<td>@C(mResp_Done) AND (mResp_Done’= cResp_Request AND tNoReqsQd) OR @T(mFlushAllResps) ELSE</td>
<td>Idle</td>
<td>C2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run_Resp_NoWP</td>
<td>C2else</td>
</tr>
<tr>
<td>Run_Resp_NoWP</td>
<td>@T(mWayPoint) WHEN (tCurrent_Req_Type = NR AND tIRq_len = 0 AND tGRq_len = 0) ELSE</td>
<td>Run_Resp_WP</td>
<td>C3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run_Resp_NoWP</td>
<td>C3else</td>
</tr>
<tr>
<td>Run_Resp_NoWP</td>
<td>@T(mWayPoint) WHEN (tCurrent_Req_Type = NR AND (tIRq_len &gt; 0 OR tGRq_len &gt; 0)) ELSE</td>
<td>Run_Int_Resp</td>
<td>C4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run_Resp_NoWP</td>
<td>C4else</td>
</tr>
<tr>
<td>Run_Int_Resp</td>
<td>@C(mResp_Done) AND (mResp_Done’= cResp_Request AND tIRq_len = 0 AND tGRq_len = 0 AND TimeOut=false) ELSE</td>
<td>Run_Resp_WP</td>
<td>C5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run_Int_Resp</td>
<td>C5else</td>
</tr>
</tbody>
</table>

**Correspondence Between Test Cases and Mode Transitions**

Date: 4 November, 2002
Test Cases Overview (cont’d)

<table>
<thead>
<tr>
<th>Source Mode</th>
<th>Events</th>
<th>Destination Mode</th>
<th>Test Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run_Int_Resp</td>
<td>@C(mResp_Done) AND (mResp_Done'= cResp_Request AND tTimeOut=true) ELSE</td>
<td>Run_Resp_NoWP</td>
<td>C6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run_Int_Resp</td>
<td>C6else</td>
</tr>
<tr>
<td>Run_Int_Resp</td>
<td>@T(mFlushAllResps) ELSE</td>
<td>Idle</td>
<td>C7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run_Int_Resp</td>
<td>C7else</td>
</tr>
<tr>
<td>Run_Resp_WP</td>
<td>@C(mTimeout) WHEN (tIRqlen = 0 AND tGRqlen = 0) ELSE</td>
<td>Run_Resp_NoWP</td>
<td>C8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run_Resp_WP</td>
<td>C8else</td>
</tr>
<tr>
<td>Run_Resp_WP</td>
<td>@C(mResp_Request) AND (tGR_ID' != tGR_ID AND tGR_ID' &gt; 0) OR (tIR_ID' != tIR_ID AND tIR_ID' &gt; 0)) ELSE</td>
<td>Run_Int_Resp</td>
<td>C9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run_Resp_WP</td>
<td>C9else</td>
</tr>
<tr>
<td>Run_Resp_WP</td>
<td>@T(mFlushAllResps) ELSE</td>
<td>Idle</td>
<td>C10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Run_Resp_WP</td>
<td>C10else</td>
</tr>
</tbody>
</table>

Correspondence Between Test Cases and Mode Transitions
### Test Cases Overview (cont’d)

#### Individual Test Cases

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Individual Steps</th>
<th>Individual Steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>mResp_Request 1</td>
<td>cResp_Request 1</td>
</tr>
<tr>
<td></td>
<td>mResp_Done 1</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td>C2</td>
<td>mResp_Request 1</td>
<td>cResp_Request 1</td>
</tr>
<tr>
<td></td>
<td>mWayPoint TRUE</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td>C3</td>
<td>mResp_Request 1</td>
<td>cResp_Request 1</td>
</tr>
<tr>
<td></td>
<td>mResp_Request 10</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td>C4</td>
<td>mResp_Request 1</td>
<td>cResp_Request 1</td>
</tr>
<tr>
<td></td>
<td>mResp_Request 10</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td>C5</td>
<td>mResp_Request 1</td>
<td>cResp_Request 1</td>
</tr>
<tr>
<td></td>
<td>mResp_Request 21</td>
<td>cResp_Request 20</td>
</tr>
<tr>
<td></td>
<td>mWayPoint TRUE</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td>C6</td>
<td>mResp_Request 2</td>
<td>cResp_Request 2</td>
</tr>
<tr>
<td></td>
<td>mResp_Request 4</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td></td>
<td>mWayPoint TRUE</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td></td>
<td>mTimeOut TRUE</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td></td>
<td>mResp_Done 10</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td>C7</td>
<td>mResp_Request 9</td>
<td>cResp_Request 10</td>
</tr>
<tr>
<td></td>
<td>mResp_Request 3</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td></td>
<td>mWayPoint TRUE</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td></td>
<td>mFlushAllResps TRUE</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td>C8</td>
<td>mResp_Request 1</td>
<td>cResp_Request 1</td>
</tr>
<tr>
<td></td>
<td>mWayPoint TRUE</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td></td>
<td>mTimeOut TRUE</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td>C9</td>
<td>mFlushAllResps TRUE</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td></td>
<td>mResp_Request 1</td>
<td>cResp_Request 1</td>
</tr>
<tr>
<td></td>
<td>mResp_Request 11</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td></td>
<td>mWayPoint TRUE</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td></td>
<td>mTimeOut TRUE</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td></td>
<td>mResp_Done 10</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td>C10</td>
<td>mResp_Request 4</td>
<td>cResp_Request 4</td>
</tr>
<tr>
<td></td>
<td>mWayPoint TRUE</td>
<td>cResp_Request 0</td>
</tr>
<tr>
<td></td>
<td>mFlushAllResps TRUE</td>
<td>cResp_Request 0</td>
</tr>
</tbody>
</table>

**NOTES**

- Test case C1 may be eliminated because it is contained in test case C2.
- In many cases, for example, the first step of test case C3, an input does not generate a change in a controlled variable (above, no change is represented by `<>`).
- The second input of test case C7 produces changes in two controlled variables.
- Some of the test cases are not the shortest possible tests. For example, the first two steps of test case C9 could be deleted, since they have no effect on the state or on the controlled variables.
Test Cases Overview (cont’d)
Individual Test Cases (cont’d)

-----C1else----- -----C1else----- Eliminate -- OVERLAPPED BY C9
mFlushAllResps TRUE <>

-----C2else----- -----C2else-----
mResp_Request 1 <>
cResp_Request 1
mResp_Request 2

-----C3else----- -----C3else-----
mResp_Request 1 <>
cResp_Request 1
mResp_Request 3

-----C4else----- -----C4else-----
mResp_Request 1 <>
cResp_Request 1
mResp_Request 7
mResp_Request 3
mErrMsgBadID = ID_Out_of_Range

-----C5else----- -----C5else-----
mResp_Request 4 <>
cResp_Request 4
mWayPoint TRUE
mResp_Request 10
mResp_Request 500

-----C6else----- -----C6else----- Eliminate -- OVERLAPPED BY C6
mResp_Request 2 <>
mResp_Request 4
mResp_Request 11
mWayPoint TRUE
mTimeOut TRUE
cResp_Request 10

-----C7else----- -----C7else-----
mResp_Request 3 <>
cResp_Request 3
mResp_Request 10
mWayPoint TRUE
mResp_Request 2

-----C8else----- -----C8else-----
mResp_Request 1 <>
cResp_Request 1
mWayPoint TRUE
mResp_Request 3

-----C9else----- -----C9else-----
mFlushAllResps TRUE <>
mResp_Done 1
mResp_Request 1 <>
cResp_Request 1
mWayPoint TRUE
mFlushAllResps

-----C10else----- -----C10else-----
mResp_Request 1 <>

Notes
- Test cases C1else and C6else may be eliminated because they are contained in test cases C9 and C6, respectively.
Conclusions

What have we done

- Demonstrated feasibility of constructing a set of test sequences from an operational req. specification using a model checker
- Have done so in a manner that "covers" all possible system executions described by the requirements specification
- Demonstrated how one can construct from the spec a set of two-state properties (i.e., cases) that describe all possible system behaviors
Conclusions (cont’d)

- Almost all effort is in the development of the specification
- After gaining familiarity with SCR, development of specs is fairly rapid
  - Mechanics of translating statecharts to SCR specifications is straightforward
  - Information not specified in statecharts must be gathered by interviewing developers (e.g., FP response priorities)
- FP Engine represents a type of system to which SCR has not previously been applied
  - More complex
  - Does not satisfy Synchronous Hypothesis (i.e., inputs are completely consumed before another input is received)
Conclusions (cont’d)

- SCR specification captures the required behavior in an understandable way
  - Easy to change when errors are detected
  - Easy to change when one needs a different version of the FPE algorithm
  - People can be easily taught to understand the spec language

- The SCR specification is executable, allowing
  - Automatic checking for syntax and type errors, missing cases, unwanted non-determinism, circular definitions
  - Automatic construction of a simulator model of the FPE, which is useful for demonstrating and validation the spec
  - Automatic verification/refutation using model checkers/theorem provers (future)
Future Work

- Improving the scalability of the method
  - Apply abstraction methods to model checking
  - Develop an algorithm to directly build a test sequence from a property
- Method currently builds one test sequence per property: how can more than one effective test sequence be built from a single property
  - Statistical methods
  - Case splitting
  - A method such as that of Weyuker et al. [TSE, May94].
- Consider fault-tolerant behavior
- SCR from statecharts
  - Would complement work by P. Pingree in translating statecharts to Promela
- Verification of autonomous systems
  - MDS
  - DS-1Remote Agent
  - ...