

# *Automated Specification-Based Test Case Generation Using SCR*

Allen Nikora  
Quality Assurance Office (512)  
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 area of JPL's Center for Space Mission Information and Software Systems (CSMISS).

# Agenda

- Task Objectives
- Success Criteria
- Approach
- Work Accomplished
  - SCR Specification of FPE
  - Simulation of FPE, graphical interface for simulator
  - Test Cases
- Issues
- Conclusions
- Future Work
  - Deployment
  - Extension of FPE specification
  - Complete FPE requirements document/user's manual

# Task Objectives

- Pilot the use of the SCR-based test case generator on real development efforts. Initial work will be accomplished using Deep Impact FP Engine
- Determine how to most effectively use the test case generator and other SCR capabilities on JPL projects

# Success Criteria

- Accurate evaluation of using SCR test case generation for critical flight software. Understand applicability and limitations.
- Conduct evaluation using real mission software component – Deep Impact Fault Protection Engine.
- Successful transformation of FP Engine specifications to SCR notation
- Develop guidelines for generating effective test cases with SCR.

# Approach

- Acquire and install SCR toolset from Naval Research Laboratory
- Identify appropriate areas of collaborating efforts for which SCR will be used to generate test cases
  - Deep Impact/Starlight FP Engine
  - Deep Impact/Starlight FP Monitors and Responses (as time and availability allow)
  - Critical MER components (if time and effort allow)
- Transcribe the requirements for the selected areas into the SCR notation
- Use the test case generator to produce the test cases from the SCR specification
- Evaluate the test cases

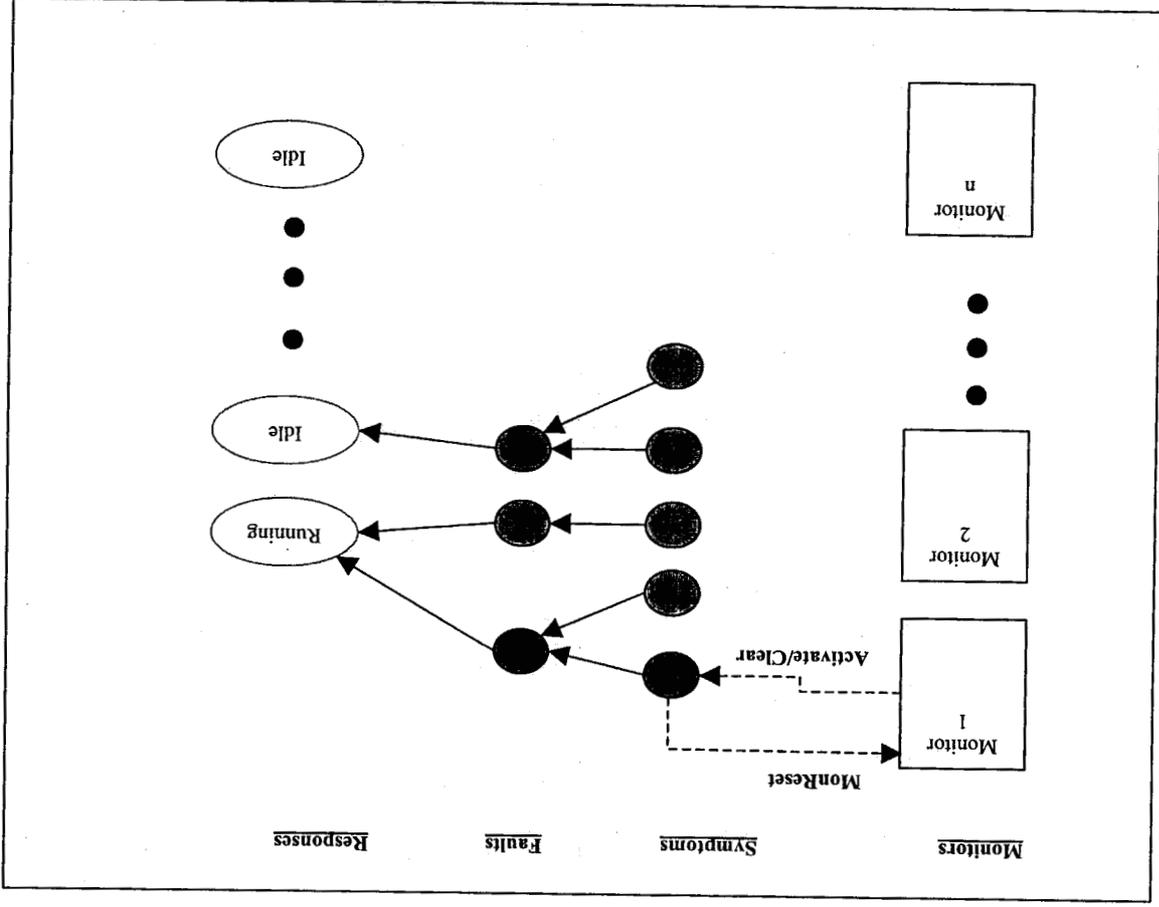
# Work Accomplished

- SCR Specification
- SCR Simulation
- Test Cases

# SCR Specification Overview

- SCR specification of Fault Protection Engine based on:
  - Final report of an effort to model the Fault Protection Engine using SDL.
  - Stateflow diagrams for FP engine available from JPL-internal website
  - Deep Impact FP Engine design documentation – available from the on-line Deep Impact project library

# FPE Symptom/Fault/Response Mapping





# SCR Specification Overview (cont'd)

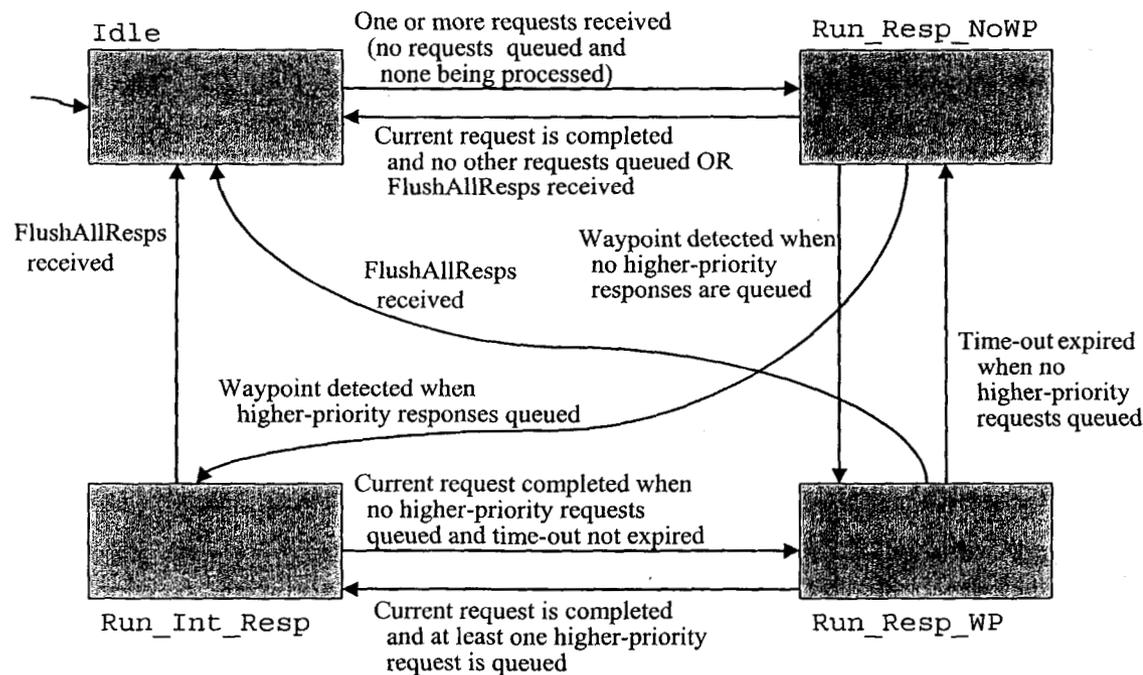
- Simplifying Abstractions

- No sub-responses
- The FPE as described in the available documentation allows a response to call sub-responses. This behavior is shown as two transitions on Slide 9:
  - “SubResp/RespInit”, from Run\_Response to Run\_Response. When the FPE encounters a call to a sub-response, it will suspend the currently-running response and cause the named sub-response to begin executing.
  - “SubResp/RespInit”, from Run\_Interrupting\_Response to Run\_Interrupting\_Response. The behavior is the same as for the transition identified above.

To simplify this version of the SCR specification, we did not include in it this aspect of the FPE.

- The maximum number of responses of each type is significantly smaller in the specification than it would be in a real spacecraft.
  - Real space missions can have 20 or more different fault responses.
  - Lower number of responses of each type is adequate to accurately model the interactions between the different types of response requests that the FPE could encounter.
- The response deferral mechanism in the implemented FPE is somewhat more complicated than what is shown in the specification.
  - In the SCR specification, no response deferral queue can have two responses having the same ID.
  - In the implemented FPE, a given fault cannot queue the same response twice.

# SCR Specification Overview (cont'd)



## STD for FPE specification

# SCR Specification Overview (cont'd)

| Statechart                              |  |  | SCR Specification                        |               |  |
|---|--|--|--|---------------|--|
| Begin State                             | End State  | Transition Event   | Begin State                              | End State     | Transition Event   |
| Idle                                    | Run_Response   | Received request to run a response OR there are one or more deferred responses | Idle                                     | Run_Resp_NoWP | Received request to run a response OR there are one or more deferred responses |
| Run_Response, Run_Interrupting_Response | Idle   | Received request to flush all responses  | Run_Resp_NoWP, Run_Resp_WP, Run_Int_Resp | Idle          | Received request to flush all responses  |
| Run_Response                            | Idle if stack is empty, Run_Response if stack is not empty | -  | Run_Resp_NoWP                            | Idle          | Response completed   |
| NoWayPoint                              | WayPoint   | Waypoint encountered in response   | Run_Resp_NoWP                            | Run_Resp_WP   | Waypoint encountered in response   |

## Comparison of SCR Specification to FPE statechart

# SCR Specification Overview (cont'd)

| Statechart                |           |  | SCR Specification |               |   |
|---------------------------|-----------|--|-------------------|---------------|---|
| Begin State               | End State | Transition Event                                     | Begin State       | End State     | Transition Event  |
| -                         | -         | -  | Run_Resp_NoWP     | Run_Int_Resp  | Waypoint encountered in response AND there are one or more deferred interrupting or ground-requested responses. |
| Run_Interrupting_Response | WayPoint  | Interrupting or ground requested response completes. | Run_Int_Resp      | Run_Resp_WP   | Interrupting or ground requested response completes prior to expiration of waypoint.                            |
| -                         | -         | -  | Run_Int_Resp      | Run_Resp_NoWP | Waypoint expired prior to completing interrupting or ground requested response.                                 |

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

# SCR Specification Overview (cont'd)

| Statechart  |                           |   | SCR Specification |               |   |
|-------------|---------------------------|---|-------------------|---------------|---|
| Begin State | End State                 | Transition Event  | Begin State       | End State     | Transition Event  |
| WayPoint    | NoWayPoint                | Waypoint has expired.   | Run_Resp_WP       | Run_Resp_NoWP | Waypoint has expired.   |
| WayPoint    | Run_Interrupting_Response | Request for interrupting or ground requested response received<br>OR there are one or more deferred interrupting or ground-requested responses. | Run_Resp_WP       | Run_Int_Resp  | Request for interrupting or ground requested response received. |

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

# SCR Specification Overview (cont'd)

| Statechart                 | SCR Specification | Comments   |
|----------------------------|-------------------|--|
| <b>Monitored Variables</b> |                   |  |
| IntResp                    | mRespRequest      | 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. |
| NonIntResp                 | mRespRequest      | See above  |
| ReqResp                    | mRespRequest      | See above  |
| IsDone                     | MrespDone         | 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.  |
| FlushAll                   | mFlushAllResps    | A signal to the FPE to terminate the currently-executing response and cancel all deferred response requests.   |

## Monitored and Controlled Variables

# SCR Specification Overview (cont'd)

| Statechart                  | SCR Specification | Comments   |
|-----------------------------|-------------------|--|
| <b>Monitored Variables</b>  |                   |  |
| EnterWayPoint               | mWayPoint         | 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.  |
| ExitWayPoint                | mTimeOut          | 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.  |
| <b>Controlled Variables</b> |                   |  |
| RespInit                    | cResp_Request     | 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. |

## Monitored and Controlled Variables (cont'd)

# SCR Specification Overview (cont'd)

## Logic of the FPE Specification

- The term `tCurrent_Req_ID` is assigned a value as follows:
  - If the FPE is in `Idle` or `Run_Resp_WP` modes and one or more new requests for high-priority responses are received (indicated by a change in `mRequest_Resp` and either of `tGR_ID` and `tIR_ID` is non-zero), then
    - `tCurrent_Req_ID' = tGR_ID` if `tGR_ID` is non-zero
    - `tCurrent_Req_ID' = tIR_ID` if `tGR_ID` is zero and `tNR_ID` is non-zero
  - If the FPE is in `Idle` mode and only a new request for a non-interrupting response is received (indicated by a change in `mRequest_Resp` and both `tGR_ID` and `tIR_ID` are zero while `tNR_ID` is non-zero), then
    - `tCurrent_Req_ID' = tNR_ID`
  - If the FPE is in `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
    - `tCurrent_Req_ID'` is assigned the ID of the longest deferred request in 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 if the queue of ground requests is empty (`tGRq_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)

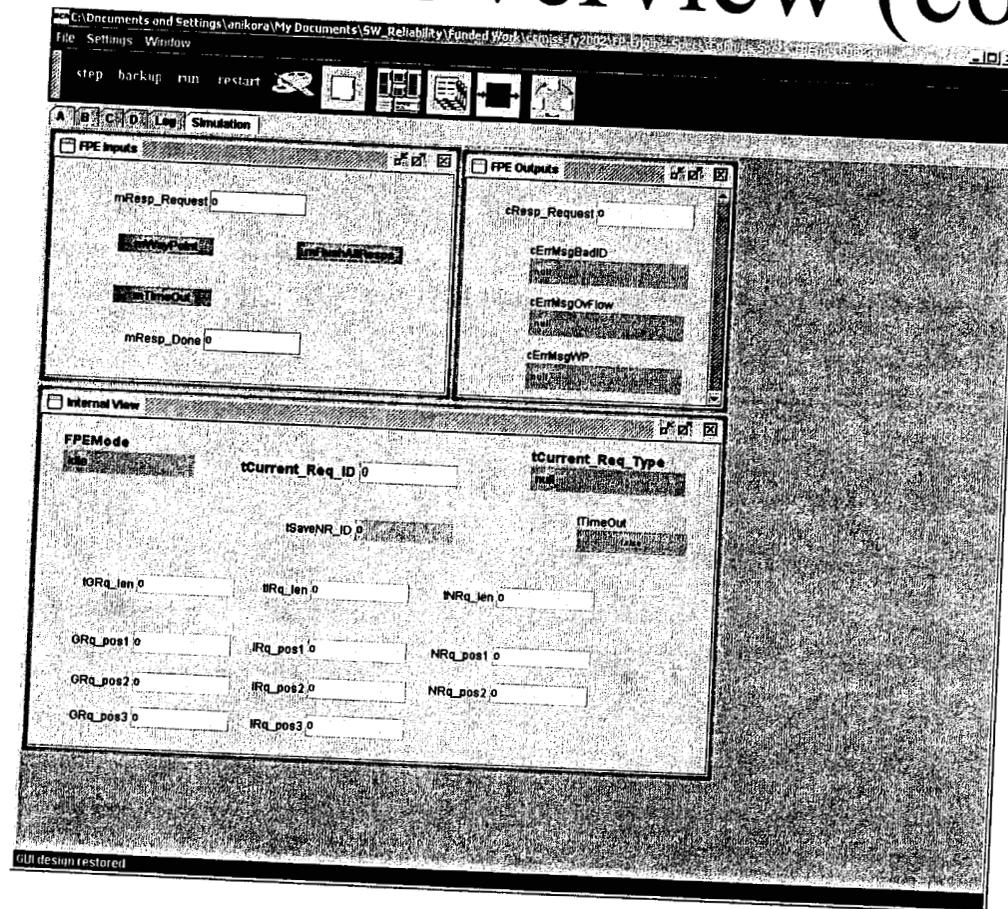
## Logic of the FPE Specification (cont'd)

- 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`.

# 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
- 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)

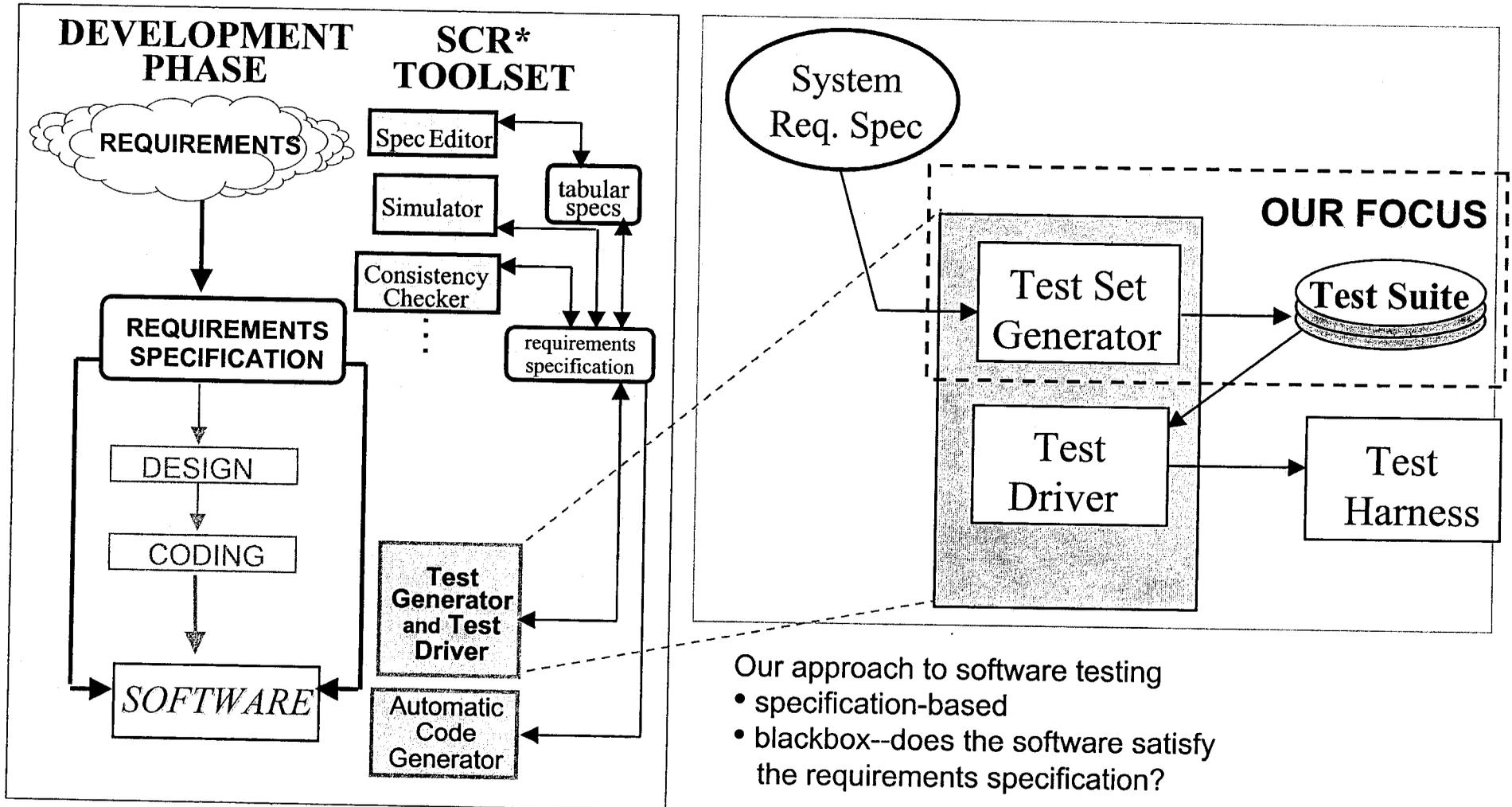


## FPE Simulator GUI

# 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



# 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:

$$\text{@T(mFlushAllResps) WHEN FPEMode = Run\_Int\_Resp} \\ \Rightarrow \text{FPEMode}' = \text{Idle}$$

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

$$\text{NOT ( @T(mFlushAllResps) WHEN FPEMode = Run\_Int\_Resp)}$$

The above property is an example of a **TRAP PROPERTY**

## Example: The Mode Transition Table From The SCR Spec Of The FPE

| Old Mode      | Event   | New Mode      |
|---------------|---|---------------|
| Idle          | @C(mResp_Request) AND...  | Run_Resp_NoWP |
| Run_Resp_NoWP | @T(mWayPoint) when tCurrentReqType=NR and tIRq_len=0 and tGRq_len=0 | Run_Resp_WP   |
| ...           | ...   | ...           |
| Run_Resp_WP   | @T(mFlushAllResps)  | Idle          |

← *Table Defining the Value of FPEMode*

## Example: The Mode Transition Table From The SCR Spec Of The FPE (cont'd)

```

if FPEMode = Idle
    ^ @C(mResp_Request) AND ... → FPEMode' = Run_Resp_NoWP
    □ FPEMode = Run_Resp_NoWP      → FPEMode' = Run_Resp_WP
    ^ @T(mWayPoint) when
      tCurrentReqType=NR &
      tIRq_len=0 & tGRq_len=0
    ...
    □ FPEMode = Run_Resp_WP
      ^ @T(mFlushAllResps) → FPEMode' = Run_Resp_NoWP
    □ (else)                → FPEMode' = FPEMode
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(mResp_Request) & ... → FPEMode' = Run_Resp_NoWP      C1
    □ (else) → FPEMode' = FPEMode      C1else
  fi
  □ FPEMode = Run_Resp_NoWP
  if
    □ @C(mResp_Done) & ... → FPEMode' = Idle      C2
    □ @T(WayPoint) & ... → FPEMode' = Run_Resp_WP      C3
    □ @T(WayPoint) & ... → FPEMode' = Run_Int_Resp      C4
    □ (else) → FPEMode' = FPEMode      C2else
  fi
  □ FPEMode = Run_Resp_WP
  if
    □ ... → FPEMode' = ...
    □ (else) → FPEMode' = FPEMode
  fi
  □ FPEMode = Run_Int_Resp
  if
    □ ... → FPEMode' = ...
    □ (else) → FPEMode' = FPEMode
  fi
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:

$$\begin{aligned} \text{FPEMode} = \text{Idle} \wedge @C(\text{mResp\_Request}) \ \& \ \dots \\ \implies \text{FPEMode}' = \text{Run\_Resp\_NoWP} \end{aligned}$$

# Test Cases Overview (cont'd)

| Source Mode   | Events   | Destination Mode | Test Case |
|---------------|--|------------------|-----------|
| Idle          | @(CmResp_Reqst) AND<br>(mResp_Reqst > 0 AND ((NR_ID' > 0 AND (NR_ID' <= MaxID) OR<br>(GR_ID' > 0 AND (GR_ID' <= MaxID)<br>OR ((GR_ID' > 0 AND (GR_ID' <= MaxID))<br>ELSE)) | Run_Resp_NoWP    | C1        |
| Run_Resp_NoWP | @(CmResp_Done) AND<br>(mResp_Done = cResp_Reqst AND<br>(NoRespQ) OR @(tmFlushMResps)<br>ELSE   | Idle             | C1else    |
| Run_Resp_NoWP | @(tmWayPoint) WHEN<br>(cCurrent_Req_Type = NR AND<br>(Rq_Len = 0 AND (GRq_Len = 0)<br>ELSE   | Run_Resp_NoWP    | C2else    |
| Run_Resp_NoWP | @(tmWayPoint) WHEN<br>(cCurrent_Req_Type = NR AND<br>(Rq_Len > 0 OR (GRq_Len > 0))<br>ELSE   | Run_Resp_WP      | C3        |
| Run_Resp_NoWP | @(tmWayPoint) WHEN<br>(cCurrent_Req_Type = NR AND<br>(Rq_Len > 0 OR (GRq_Len > 0))<br>ELSE   | Run_Resp_NoWP    | C3else    |
| Run_Int_Resp  | @(tmResp_Done) AND<br>(mResp_Done = cResp_Reqst AND<br>(Rq_Len = 0 AND (GRq_Len = 0 AND<br>(TimeOut = false)<br>ELSE   | Run_Int_Resp     | C4        |
| Run_Int_Resp  | @(tmResp_Done) AND<br>(mResp_Done = cResp_Reqst AND<br>(TimeOut = true)<br>ELSE  | Run_Resp_NoWP    | C4else    |
| Run_Int_Resp  | @(tmResp_Done) AND<br>(mResp_Done = cResp_Reqst AND<br>(TimeOut = true)<br>ELSE  | Run_Resp_WP      | C5        |
| Run_Int_Resp  | @(tmResp_Done) AND<br>(mResp_Done = cResp_Reqst AND<br>(TimeOut = true)<br>ELSE  | Run_Int_Resp     | C5else    |
| Run_Int_Resp  | @(tmFlushMResps)<br>ELSE   | Run_Resp_NoWP    | C6        |
| Run_Int_Resp  | @(tmFlushMResps)<br>ELSE   | Run_Int_Resp     | C6else    |
| Run_Resp_WP   | @(tmTimeOut) WHEN (Rq_Len<br>> 0 AND (GRq_Len = 0)<br>ELSE   | Idle             | C7        |
| Run_Resp_WP   | @(tmTimeOut) WHEN (Rq_Len<br>> 0 AND (GRq_Len = 0)<br>ELSE   | Run_Int_Resp     | C7else    |
| Run_Resp_WP   | @(tmTimeOut) WHEN (Rq_Len<br>> 0 AND (GRq_Len = 0)<br>ELSE   | Run_Resp_NoWP    | C8        |
| Run_Resp_WP   | @(tmTimeOut) WHEN (Rq_Len<br>> 0 AND (GRq_Len = 0)<br>ELSE   | Run_Resp_WP      | C8else    |
| Run_Resp_WP   | @(CmResp_Reqst) AND<br>(GR_ID' != (GR_ID AND (GR_ID' ><br>0) OR (NR_ID' != (NR_ID AND<br>(NR_ID' > 0))<br>ELSE   | Run_Int_Resp     | C9        |
| Run_Resp_WP   | @(tmFlushMResps)<br>ELSE   | Run_Resp_WP      | C9else    |
| Run_Resp_WP   | @(tmFlushMResps)<br>ELSE   | Idle             | C10       |
| Run_Resp_WP   | @(tmFlushMResps)<br>ELSE   | Run_Resp_WP      | C10else   |

Correspondence Between Test Cases and Mode Transitions

# Test Cases Overview (cont'd)

## Individual Test Cases

-----C1-----      -----C1-----  
 mResp\_Request 1      cResp\_Request 1

-----C2-----      -----C2-----  
 mResp\_Request 1      cResp\_Request 1  
 mResp\_Done 1          cResp\_Request 0

-----C3-----      -----C3-----  
 mFlushAllResps TRUE       $\diamond$   
 mResp\_Request 1          cResp\_Request 1  
 mWayPoint TRUE          cResp\_Request 0

-----C4-----      -----C4-----  
**mResp\_Request 1**      **cResp\_Request 1**  
**mResp\_Request 10**       $\diamond$   
**mWayPoint TRUE**      **cResp\_Request 10**

-----C5-----      -----C5-----  
 mResp\_Request 1      cResp\_Request 1  
 mResp\_Request 21       $\diamond$   
 mWayPoint TRUE      cResp\_Request 20  
 mResp\_Done 10      cResp\_Request 0

-----C6-----      -----C6-----  
 mResp\_Request 2      cResp\_Request 2  
 mResp\_Request 4       $\diamond$   
 mResp\_Request 11       $\diamond$   
 mWayPoint TRUE      cResp\_Request 10  
 mTimeOut TRUE       $\diamond$   
 mResp\_Done 10      cResp\_Request 2

-----C7-----      -----C7-----  
**mResp\_Request 9**      **cErrMsgBadID = ID\_Out\_of\_Range**  
**mResp\_Request 3**      **cErrMsgBadID = null**  
                          **cResp\_Request 3**  
                           $\diamond$   
**mResp\_Request 13**      **cResp\_Request 10**  
**mWayPoint TRUE**      **cResp\_Request 0**  
**mFlushAllResps TRUE**      **cResp\_Request 0**

-----C8-----      -----C8-----  
 mFlushAllResps TRUE       $\diamond$   
 mResp\_Request 1      cResp\_Request 1  
 mWayPoint TRUE      cResp\_Request 0  
 mTimeOut TRUE      cResp\_Request 1

-----C9-----      -----C9-----  
 mFlushAllResps TRUE       $\diamond$   
 mResp\_Done 1       $\diamond$   
 mResp\_Request 3      cResp\_Request 3  
 mWayPoint TRUE      cResp\_Request 0  
 mResp\_Request 10      cResp\_Request 10

-----C10-----      -----C10-----  
 mResp\_Request 4      cResp\_Request 4  
 mWayPoint TRUE      cResp\_Request 0  
 mFlushAllResps TRUE       $\diamond$

### 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  $\diamond$ ).
- 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 --  
 mFlushAllResps TRUE    <>    OVERLAPPED BY C9

-----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    cErrMsgBadID = ID\_Out\_of\_Range

-----C5else-----    -----C5else-----  
 mResp\_Request 4    cResp\_Request 4  
 mWayPoint TRUE    cResp\_Request 0  
 mResp\_Request 10    cResp\_Request 10  
 mResp\_Request 500    <>

-----C6else-----    -----C6else-----    Eliminate -- OVERLAPPED BY C6  
 mResp\_Request 2    cResp\_Request 2  
 mResp\_Request 4    <>  
 mResp\_Request 11    <>  
 mWayPoint TRUE    cResp\_Request 10  
 mTimeout TRUE    <>

-----C7else-----    -----C7else-----  
 mResp\_Request 3    cResp\_Request 3  
 mResp\_Request 10    <>  
 mWayPoint TRUE    cResp\_Request 10  
 mResp\_Request 2    <>

-----C8else-----    -----C8else-----  
 mResp\_Request 1    cResp\_Request 1  
 mWayPoint TRUE    cResp\_Request 0  
 mResp\_Request 3    <>

-----C9else-----    -----C9else-----  
 mFlushAllResps TRUE    <>  
 mResp\_Done 1    <>  
 mResp\_Request 1    cResp\_Request 1  
 mWayPoint TRUE    cResp\_Request 0  
 mFlushAllResp    <>

-----C10else-----    -----C10else-----  
 mResp\_Request 1    cResp\_Request 1  
 mWayPoint TRUE    cResp\_Request 0  
 mResp\_Request 2    <>

### NOTES

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

# Issues

- Unspecified Behavior

- The priority of the different types of responses was not specified in the available documentation.
- Although non-interrupting responses are the only type of response intended to have waypoints, there is no on-board enforcement mechanism. This allows the following possibilities:
  - An interrupting or a ground-requested response could implement way-points
  - A sub-response to an interrupting or non-interrupting response could have waypoints.

Under these circumstances, the behavior of the FPE is not defined.

- The descriptive material used as the basis for the specification developed for this task does not define the behavior of the FPE if it receives a request for a non-existent response. A diagnostic message will be displayed during simulation runs if this situation arises.

# 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

## What next?

- How to improve the scalability of the method
  - Apply abstraction methods to model checking
  - Develop an algorithm to directly build a test sequence from a property
- Our 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

# Conclusions (cont'd)

- SCR test generation facility appears to be appropriate for components or systems at the FPE level of complexity
  - Need to see how the test case generator would react to the addition of symptoms and responses
- 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

- Deployment
  - Work with DI, other projects to identify appropriate components to which technique could be applied
- Extend FPE specification to include:
  - Additional priority levels for responses
  - Priority aging for responses
  - Waypoints in higher priority responses
  - Detecting/terminating responses that haven't completed
  - Multithreading

# Future Work (cont'd)

- FPE Requirements/User's Manual
  - Any use requires knowledge of SCR; implies formal specification/SCR training will be needed
  - Actual requirements spec will include
    - Precise verbal description of FPE and other components
    - Abstract description of queues
    - What Ids of requests actually are (rather than the placeholders that are currently used)
  - Critical properties that the FPE must satisfy (e.g., deadlock free, etc.)
    - Actual time-outs
    - Subresponses
  - Likely areas of change
    - Other classes of requests besides Ground, Non-Interrupting, and Interrupting
    - Queue Lengths