

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

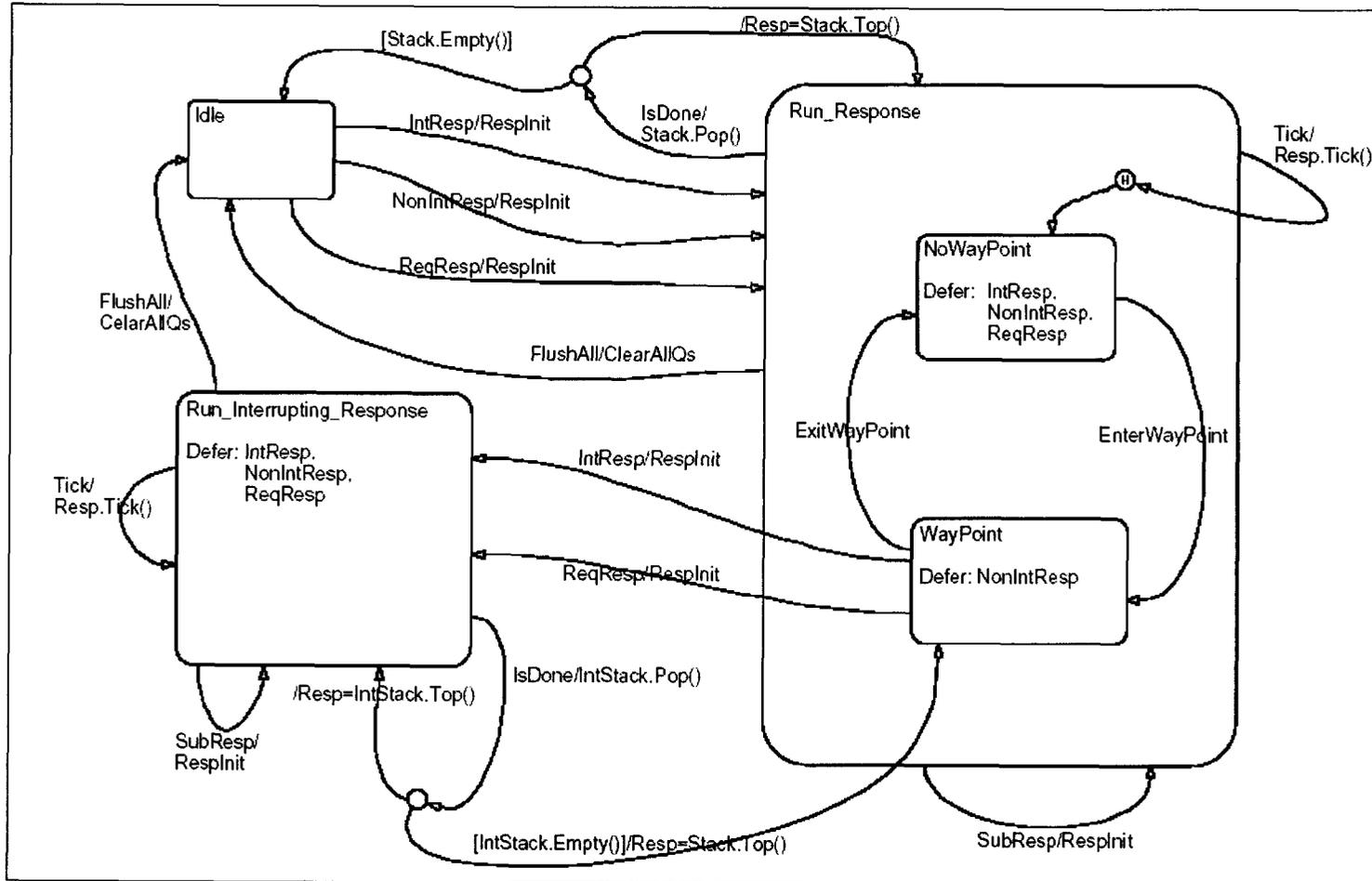
Work Accomplished

- SCR Specification
- SCR Simulation
- Test Cases

SCR Specification Overview

- SCR specification of Fault Protection Engine based on:
 - “SDL for Flight Software”, Final Report, Garth Watney, September 28, 2001
 - 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

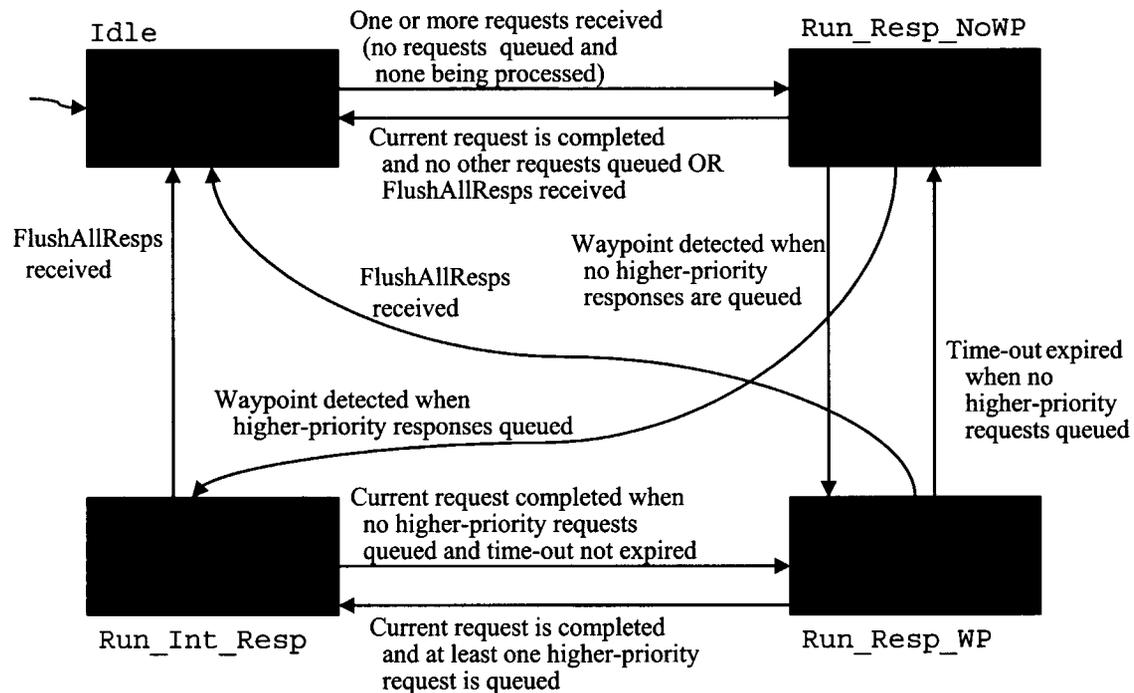
Fault Protection Engine



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

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 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
Monitored Variables		
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
Monitored Variables		
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.
Controlled Variables		
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`.

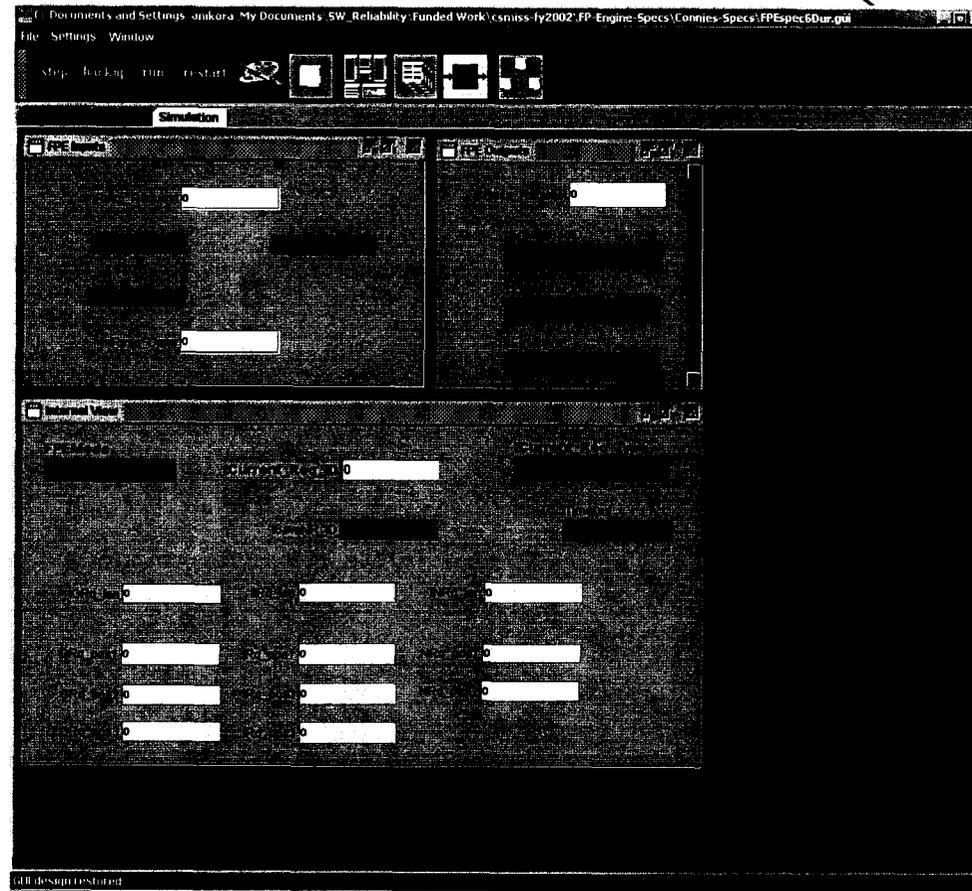
SCR Specification Overview (cont'd)

- 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)

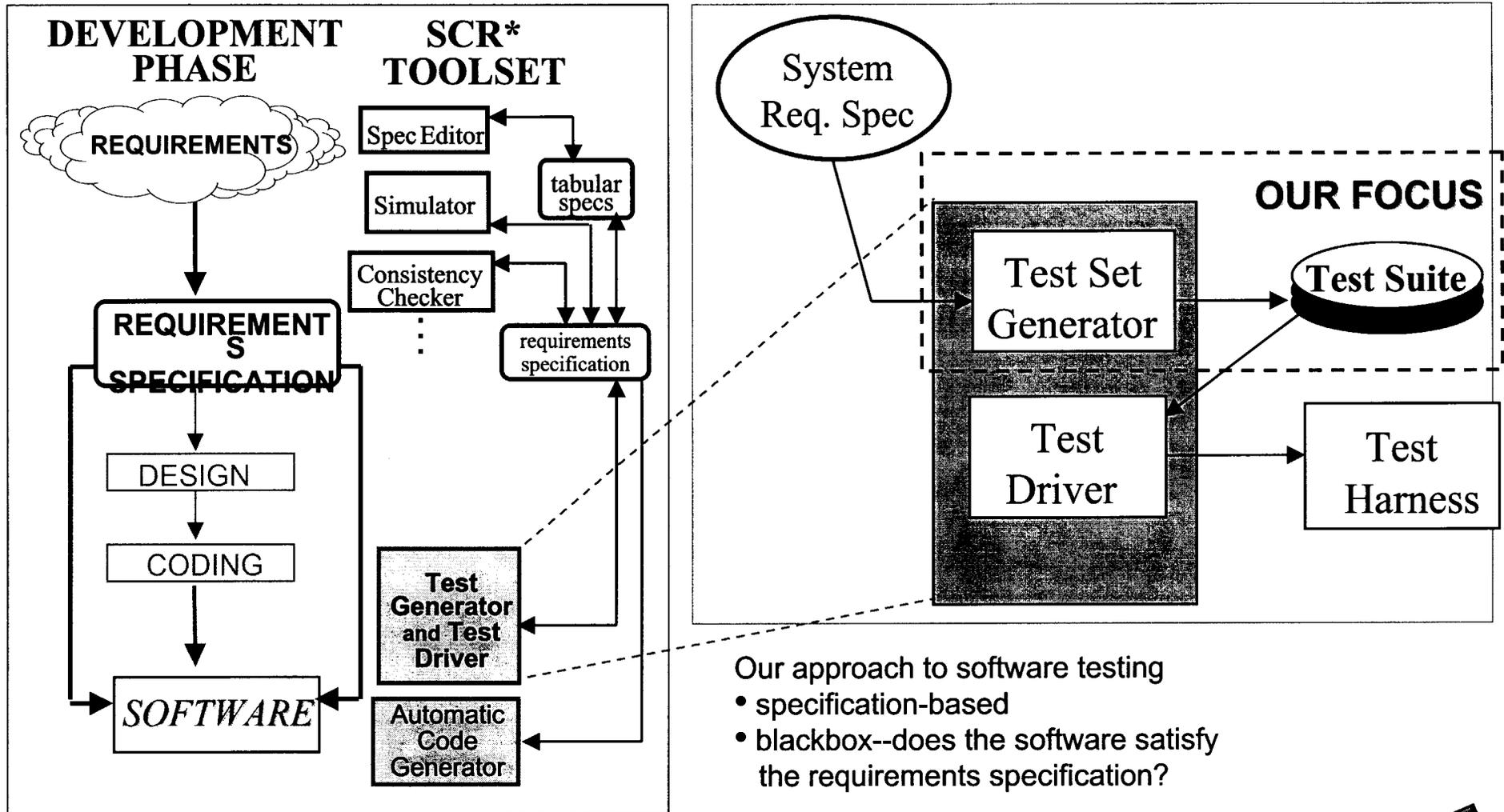


FPE Simulator GUI

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



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:

$$\text{FPMode} = \text{Idle} \wedge @C(\text{mResp_Request}) \ \& \ \dots$$
$$\Rightarrow \text{FPMode}' = \text{Run_Resp_NoWP}$$

Test Cases Overview (cont'd)

Source Mode	Events	Destination Mode	Test Case
Idle	@C(mResp_Request) AND (mResp_Request' > 0 AND ((tNR_ID' > 0 AND tNR_ID' <= MaxID) OR (tIR_ID' > 0 AND tIR_ID' <= MaxID) OR (tGR_ID' > 0 AND tGR_ID' <= MaxID))) ELSE	Run_Resp_NoWP Idle	C1 C1else
Run_Resp_NoWP	@C(mResp_Done) AND (mResp_Done' = cResp_Request AND tNoReqsQd) OR @T(mFlushAllResps) ELSE	Idle Run_Resp_NoWP	C2 C2else
Run_Resp_NoWP	@T(mWayPoint) WHEN (tCurrent_Req_Type = NR AND tIRq_len = 0 AND tGRq_len = 0) ELSE	Run_Resp_WP Run_Resp_NoWP	C3 C3else
Run_Resp_NoWP	@T(mWayPoint) WHEN (tCurrent_Req_Type = NR AND (tIRq_len > 0 OR tGRq_len > 0)) ELSE	Run_Int_Resp Run_Resp_NoWP	C4 C4else
Run_Int_Resp	@C(mResp_Done) AND (mResp_Done' = cResp_Request AND tIRq_len = 0 AND tGRq_len = 0 AND TimeOut=false) ELSE	Run_Resp_WP Run_Int_Resp	C5 C5else

Correspondence Between Test Cases and Mode Transitions

Test Cases Overview (cont'd)

Source Mode	Events	Destination Mode	Test Case
Run_Int_Resp	@C(mResp_Done) AND (mResp_Done'≠ cResp_Request AND tTimeOut=true) ELSE	Run_Resp_NoWP Run_Int_Resp	C6 C6else
Run_Int_Resp	@T(mFlushAllResps) ELSE	Idle Run_Int_Resp	C7 C7else
Run_Resp_WP	@C(mTimeOut) WHEN (tIRq_len = 0 AND tGRq_len = 0) ELSE	Run_Resp_NoWP Run_Resp_WP	C8 C8else
Run_Resp_WP	@C(mResp_Request) AND ((tGR_ID' ≠ tGR_ID AND tGR_ID' > 0) OR (tIR_ID' ≠ tIR_ID AND tIR_ID' > 0)) ELSE	Run_Int_Resp Run_Resp_WP	C9 C9else
Run_Resp_WP	@T(mFlushAllResps) ELSE	Idle Run_Resp_WP	C10 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 ◇
mResp_Request 1 cResp_Request 1
mWayPoint TRUE cResp_Request 0

-----C4----- -----C4-----
mResp_Request 1 cResp_Request 1
mResp_Request 10 ◇
mWayPoint TRUE cResp_Request 10

-----C5----- -----C5-----
mResp_Request 1 cResp_Request 1
mResp_Request 21 ◇
mWayPoint TRUE cResp_Request 20
mResp_Done 10 cResp_Request 0

-----C6----- -----C6-----
mResp_Request 2 cResp_Request 2
mResp_Request 4 ◇
mResp_Request 11 ◇
mWayPoint TRUE cResp_Request 10
mTimeOut TRUE ◇
mResp_Done 10 cResp_Request 2

-----C7----- -----C7-----
mResp_Request 9 cErrMsgBadID = ID_Out_of_Range
mResp_Request 3 cErrMsgBadID = null
 cResp_Request 3
 ◇
mResp_Request 13 cResp_Request 10
mWayPoint TRUE cResp_Request 0
mFlushAllResps TRUE

-----C8----- -----C8-----
mFlushAllResps TRUE ◇
mResp_Request 1 cResp_Request 1
mWayPoint TRUE cResp_Request 0
mTimeOut TRUE cResp_Request 1

-----C9----- -----C9-----
mFlushAllResps TRUE ◇
mResp_Done 1 ◇
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 ◇

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----- mFlushAllResps TRUE	-----C1else----- ◇	Eliminate -- OVERLAPPED BY C9	-----C6else----- mResp_Request 2 mResp_Request 4 mResp_Request 11 mWayPoint TRUE mTimeOut TRUE	-----C6else----- cResp_Request 2 ◇ ◇ cResp_Request 10 ◇	Eliminate -- OVERLAPPED BY C6	-----C10else----- mResp_Request 1 mWayPoint TRUE mResp_Request 2	-----C10else----- cResp_Request 1 cResp_Request 0 ◇
-----C2else----- mResp_Request 1 mResp_Request 2	-----C2else----- cResp_Request 1 ◇		-----C7else----- mResp_Request 3 mResp_Request 10 mWayPoint TRUE mResp_Request 2	-----C7else----- cResp_Request 3 ◇ cResp_Request 10 ◇			
-----C3else----- mResp_Request 1 mResp_Request 3	-----C3else----- cResp_Request 1 ◇		-----C8else----- mResp_Request 1 mWayPoint TRUE mResp_Request 3	-----C8else----- cResp_Request 1 cResp_Request 0 ◇			
-----C4else----- mResp_Request 1 mResp_Request 7	-----C4else----- cResp_Request 1 cErrMsgBadID = ID_Out_of_Range		-----C9else----- mFlushAllResps TRUE mResp_Done 1 mResp_Request 1 mWayPoint TRUE mFlushAllResp	-----C9else----- ◇ ◇ cResp_Request 1 cResp_Request 0 ◇			
-----C5else----- mResp_Request 4 mWayPoint TRUE mResp_Request 10 mResp_Request 500	-----C5else----- cResp_Request 4 cResp_Request 0 cResp_Request 10 ◇						

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
 - ...