Model Checking Investigations for Fault Protection System Validation

Kevin J. Barltrop

Paula J. Pingree

Background

- Complexity of software for flight missions is increasing
  - Entry, Descent & Landing on Mars
  - Rendezvous with celestial bodies
  - Spacecraft Formation Flying
  - Rover Surface Operations
- The criticality of flight software operation is increasing for flight missions

- Software verification and validation (V&V) methods and tools must also advance to keep pace with software development
- Traditional methods are being stretched
- Formal methods and model checking offers a powerful technique for software V&V

- We implement Fault Protection flight software for robustness and autonomy in the event of detected on-board failures or faults
- We are capturing software designs in state-charts and using automated code generation

We asked…

1. Can we apply "lightweight" formal methods and model checking to mission flight software verification at JPL?
2. Can we automate the process?
3. Can we quantify the benefits compared to traditional verification approaches?

The Approach

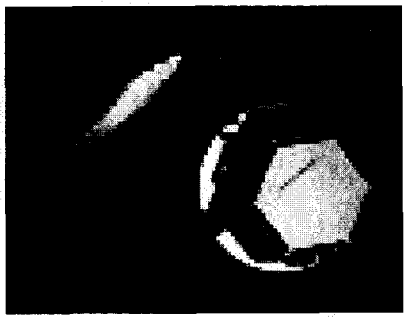
- Utilization of the Spin model checker with automatically translated state-charts provided as input by the HiVy tool set
- Validation examples selected and scoped to offer maximum demonstration benefit to flight projects within the capabilities of our R&TD team, tools and methods
- Translation of system design and environment models from Stateflow to Promela (the input language of Spin), integration of the closed-loop system including C-code interfaces, specification of Linear Temporal Logic (LTL) correctness properties to validate, and model checking results with Spin

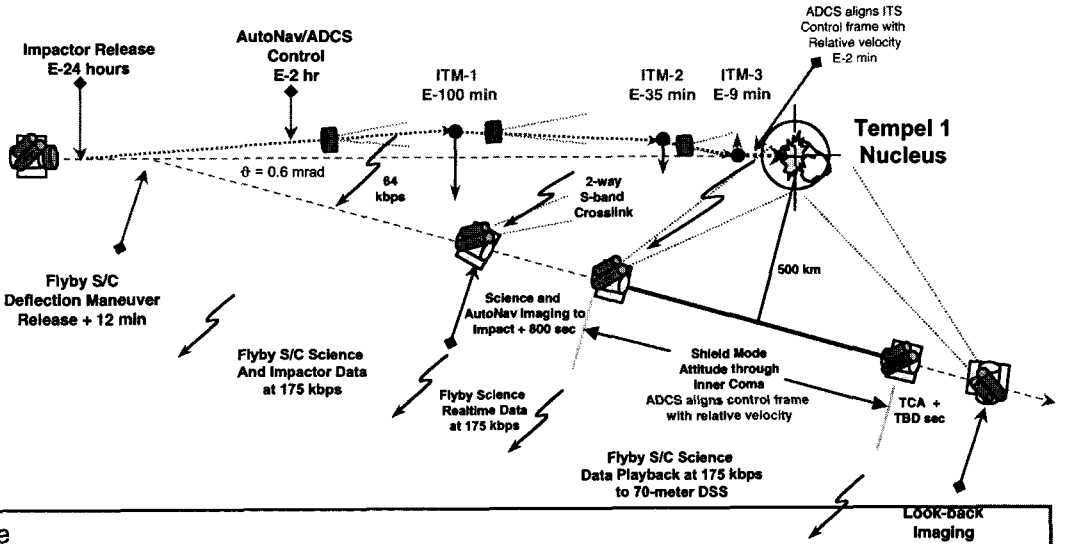
	Traditional	State-charts	Model Checking
Requirements	*Informal*	*Informal*	⇨ *Formal (LTL)*
Design	*Informal*	*Semi-formal* ⇩	⇨ *Formal (Promela)*
Code	*Formal*	*Formal*	

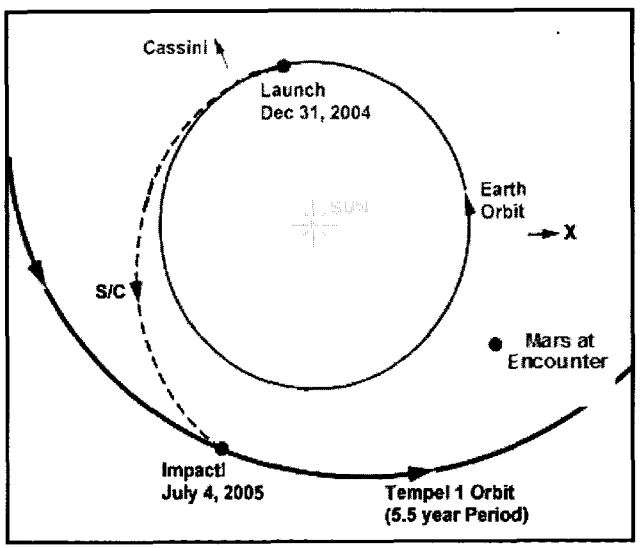
Deep Impact Mission Overview

Salient Features

- Deliver a 350 kg impactor at 10 km/s to open the interior of a comet nucleus. Target is Comet P/Tempel 1.
- Impactor produces crater dependent on comet porosity and strength.
- Flyby spacecraft observes impact, crater development, ejecta and final crater with visible and IR multi-spectral instruments.
- *On-board autonomous optical navigation* used for precise targeting and control of impactor and fly-by spacecraft.
- 7 month mission duration. Launch: December 31, 2004 / Encounter:July 4, 2005





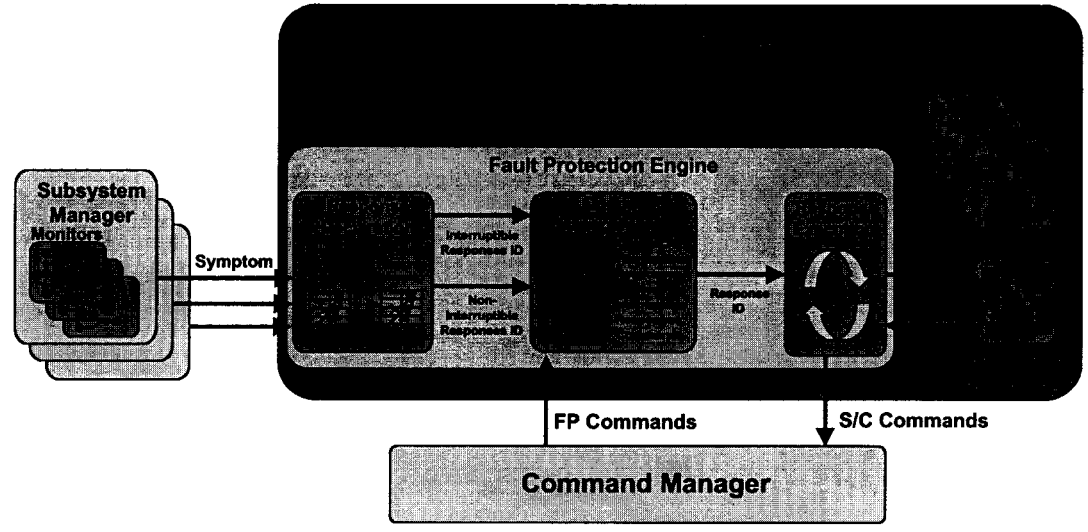


Science

- To determine the differences between the interior of a cometary nucleus and its surface.
- Determine basic cometary properties by observing how the crater forms after impact.
- To identify materials in the pristine comet interior by measuring the composition of the ejecta from the comet crater.
- Determine the changes in natural outgassing of the comet produced by the impact.
- To help discover whether comets lose their ice, or seal it in over time (evidence for dormancy vs. extinction).
- Address terrestrial hazard from cometary impacts

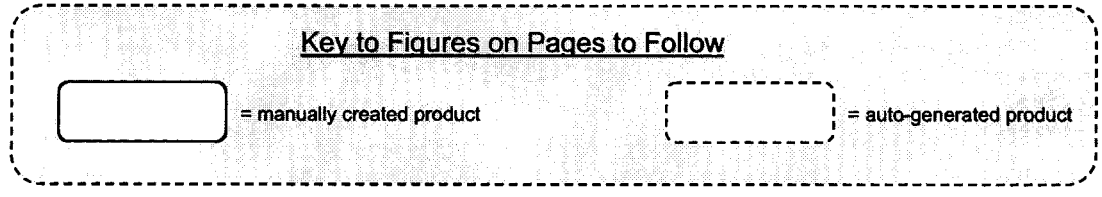
Fault Protection Architecture is Primarily Inherited from Previous Missions

- Deep Impact inherited much of its fault protection software architecture
  - ***Pathfinder*** provided a centralized fault management engine that coordinates system level responses.
  - ***Deep Space 1*** provided direct code generation from state chart diagrams.
  - ***Cassini*** provided the critical sequencing approach.

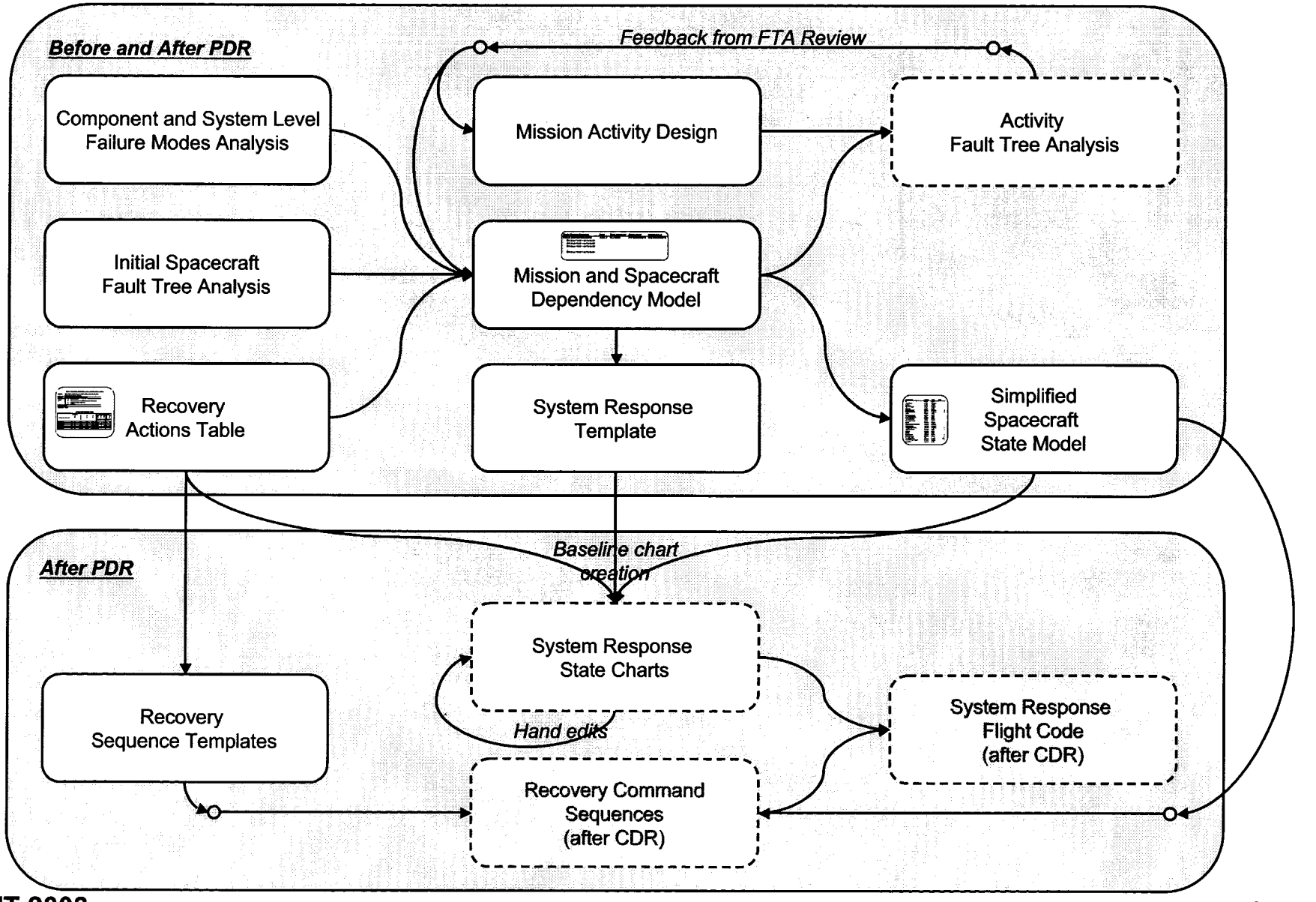


- The FP team discussed upgrading the engine to act as a model-based system
  - Existing project investment in explicit behavior design was too large to make switching techniques viable, but model-based algorithms have been implemented as ground tools.

- The project chose a compromise to exploit advantages of the two technologies
  - Use model-based ground tools for design analysis, and for downstream auto-coding of software and test scripts.
  - Use auto-coding to eliminated the need for a programmers to implement specific behaviors.



Design Process Relied on Auto-Generation of Products



Spacecraft Dependency Model Leads to State Chart Designs

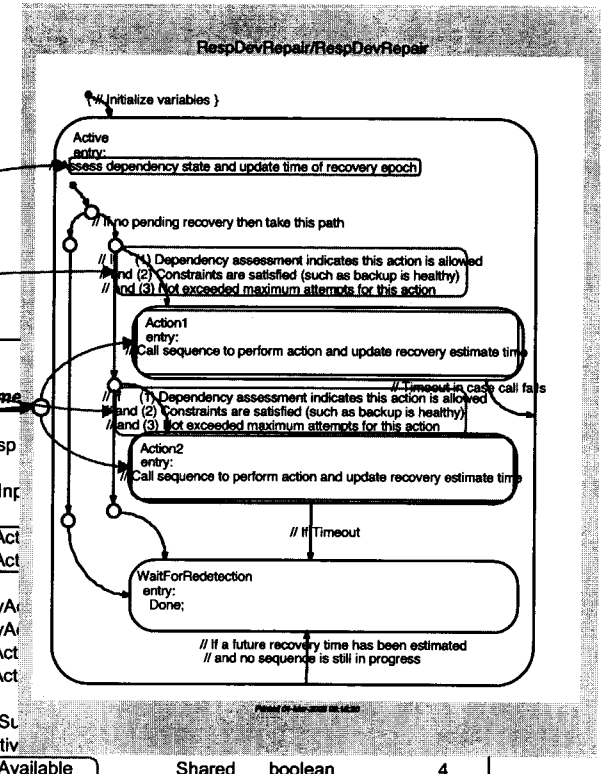
Flight System Elements	Req't	State Indicator	Prime Input	Health Input
Parent Element Mnemonic	needs X	FaultCode	PrimeStateName	HealthStateName
Child element mnemonic #1				
Child element mnemonic #2				
Child element mnemonic #3				
...				
Child element mnemonic #n				

Table 2. Illustration of Corrective Actions and Constraints Analysis

Purpose	The Device Repair Response shall recover device functionality after a fault has been detected.
Location	✱ Prime Computer ✱ Backup Computer
Tiers of action	1. Reset 1553 RT 2. If not at encounter cycle/reload device electronics 3. Swap to backup electronics 4. Exhaust
Interfering	No
Comments	none

Table 3. Tier Description Table

Response Chart ID	Table Size: Flyby	Element	Prime	Avail	FR Sequence Calls: Sync	Reset	Cycle	Escape	Isolate
					Require self available			Require Next	
Recovery#1	4	Bus	Bus	Bus	.	.	.	.	1
Recovery#2	2	Cbh	Cbh	Cbh	1	2	.	.	.
Recovery#3	2	RfCmd	RfCmd	RfCmd	1	.	.	.	2
...	2	RfRate			1	.	.	.	.
Recovery#n	3	BatCell	BatCell	BatCell	1	.	.	.	.



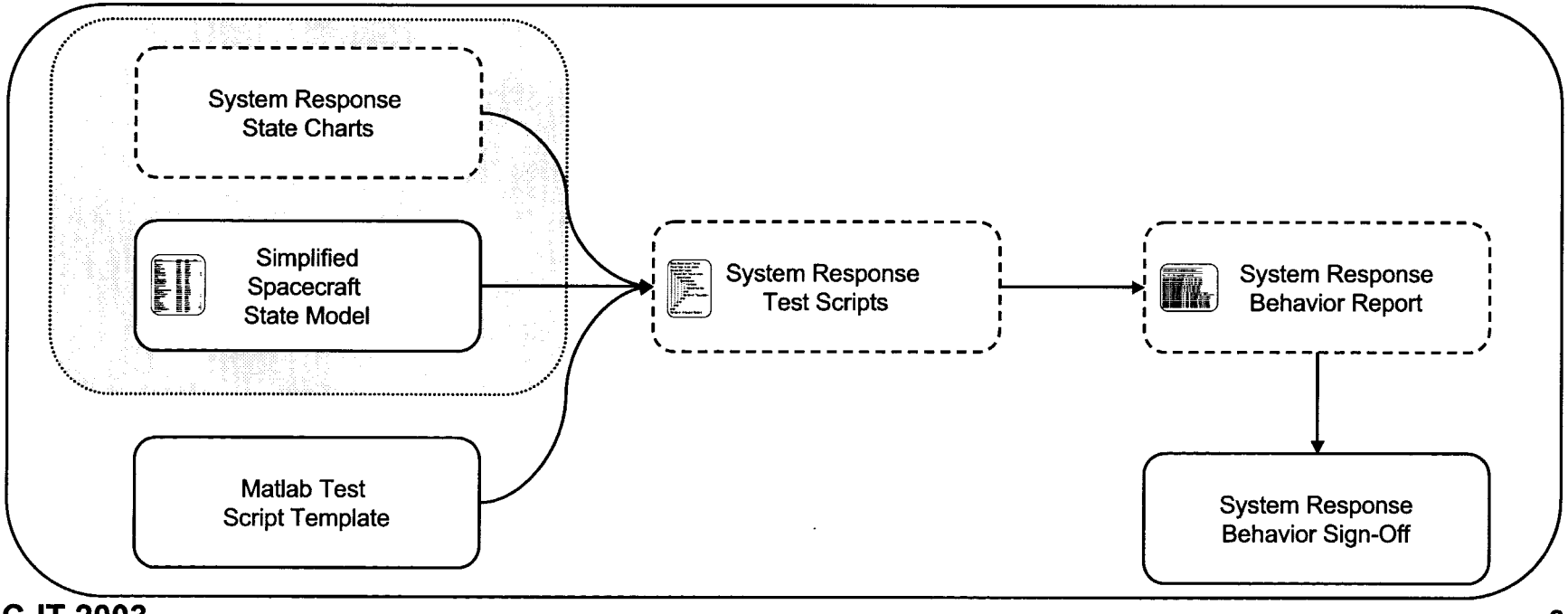
Data Name			
Id			
IsSubResp			
SysTime			
SysTimeInp			
Urgency			
EnabledAct			
EnabledAct			
IdDev			
MaxRetryA			
MaxRetryA			
TimeoutAct			
TimeoutAct			
Epoch			
CalledAsSu			
FpSeqActiv			
InputDevAvailable	Shared	boolean	4
InputDevPrime	Shared	uint8	4
MaxDev	Shared	uint8	
MinDev	Shared	uint8	
SeqIdAction1	Shared	uint16	4
SeqIdAction2	Shared	uint16	4
TimeOfLastRun	Shared	double	256
TimeOfRecoveryDev	Shared	double	4
ValueDev	Shared	uint8	

Manually created products based on system and subsystem interviews

Auto-generated products from the simplified spacecraft model and from template. This state information eventually feeds into the *Promela* environment model.

Testing Prior to Creating Source Code Relied on Auto-Generation of Products

- Initial design testing derives directly from the design tables and model files.
- To keep test time reasonable, the Matlab test scripts are restricted to traversing a "high value" subset of the spacecraft state space.
  - We wish to focus on aspects that are unique to each behavior.
  - For states that are instances of the same kind of state, the test covers just a subset of those instances. (e.g., a chart has 64 instance of thermal channels – we explore the first four and last two)
  - The explored state space is based on classification of state and input variables (selects, masks, events), with state vector variations limited to at most <u>one</u> member of <u>each</u> class.
- At this level we cannot check end-to-end flight system behavior, because the Matlab test harness lacks a representation for the rest of the flight system.
  - In a later venue we'll use our models to auto-generate behavior predicts for activities on various test beds.



Automated Test Script Generation Leads to Response Behavior Report

Data Name	Scope	Type	Size
Id	Local	uint8	
IsSubResp	Local	boolean	
SysTime	Local	double	
SysTimeInput	Local	double	
Urgency	Local	uint8	
EnabledAction1	Param	boolean	3
EnabledAction2	Param	boolean	3
IdDev	Param	uint8	
MaxRetryAction1	Param	uint8	
MaxRetryAction2	Param	uint8	
TimeoutAction1	Param	uint16	
TimeoutAction2	Param	uint16	
Epoch	Persist	double	
CalledAsSubResponse	Shared	boolean	
FpSeqActive	Shared	boolean	
InputDevAvailable	Shared	boolean	4
InputDevPrime	Shared	uint8	4
MaxDev	Shared	uint8	
MinDev	Shared	uint8	
SeqIdAction1	Shared	uint16	4
SeqIdAction2	Shared	uint16	4
TimeOfLastRun	Shared	double	256
TimeOfRecoveryDev	Shared	double	4
ValueDev	Shared	uint8	

Entries in data dictionary are classified for script generation according to naming conventions.

```
Data Description Tables
Parameter Initialization
Select Item Loop
  Select Item Value Loop
    Mask Loop
      Event Loop
        Time Loop
          Response Call
        end
        Behavior Tabulation
      end
    end
  end
end
Behavior Analysis Report
```

Auto-generated Matlab script from data dictionary and template.

Examples:
Select → "Select star tracker A"
Mask → "Don't power-cycle if is back-up"
Event → "Delay action if recovery is in progress"

```
=========================================
VERIFICATION FOR RespAttControlRepair
=========================================
 ----------------------------------------
Req (10) - Fix appropriate element
----------------------------------------
Applied RespAttEstRepair(1) to no test case
Applied RespAttEstRepair(2) to no test case
Applied RespAttEstRepair(3) to no test case
Applied RespAttEstRepair(4) to no test case
Applied RespGyroRepair(1) to no test case
Applied RespGyroRepair(2) to no test case
Applied RespGyroRepair(3) to no test case
Applied RespGyroRepair(4) to no test case
Applied SeqGyroEscape(1) with AllSelect=0 2 times
Applied SeqGyroEscape(1) with FpHazardFault(1)=1 2 times
Applied SeqGyroEscape(1) with ValueGyro(1)=1 2 times
Applied SeqGyroEscape(1) with ValueGyro(1)=2 2 times
Applied SeqGyroEscape(1) with RiuIdRiu(1)=1 2 times
Applied SeqGyroEscape(1) with ValueRiu(1)=1 2 times
Applied SeqGyroEscape(1) with ValueRiu(1)=2 2 times
Applied SeqGyroEscape(1) with StkIdStkr(1)=1 2 times
Applied SeqGyroEscape(1) with ValueStkr(1)=1 2 times
```

Auto-generated by post-processing tool, but sign-off on by test engineer. The requirements checked here feed into the correctness properties defined for *Promela*.

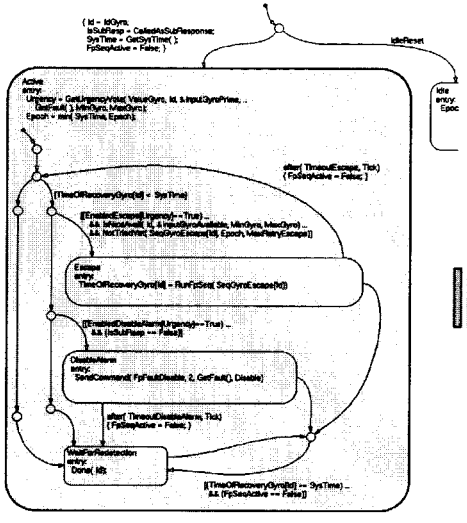
Summaries of Spin & HiVy

- Spin is a widely distributed software package that supports the formal verification of distributed systems
- The software was developed at Bell Labs by Gerard Holzmann
- Promela (Pro*cess* Me*ta* La*nguage*) is the Spin input language
- The Spin software is written in ANSI standard C, and is portable across all versions of the UNIX operating system. It can also be compiled to run on any standard PC running Linux, Windows95/98, or WindowsNT.
- http://netlib.bell-labs.com/netlib/spin/whatispin.html

- HiVy is based on the new Hierarchical Sequential Automata (HSA) format and provides automatically translated models for input to Spin
- HiVy was developed by JPL and Erich Mikk (independent consultant) beginning in FY02
- The HiVy toolset consists of the programs:
  - *SfParse* extracts pertinent data from the Stateflow® model file
  - *sf2hsa* translates parsed output into HSA (intermediate format)
  - *hsa2pr* translates HSA into Promela
  - and the HSA Merge Facility

Promela Model Creation

1. Translate the Fault Protection Response state-chart



SfParse → Sf2HSA → HSA2PR

```
/**
 * function_Active()
 **/

active proctype function_Active()
{
      /*
        current_state_Active
         =state_WaitForRedetection;
      */
endloop1:
atomic
      {
       WAIT_ACTIVATION(activation_Active);
       /* case 1 */
       if :: current_state_Active ==
       state_DisableAlarm ->label_DisableAlarm:
      {
            if :: (T) ->
```

2. Create Promela environment model to close-the-loop around the FP response

spacecraft model

```
bool FPR_TURN_ON__FSC_P7_H_SW1=F;
bool FPR_TURN_OFF__FSC_PP_HTR_SW1=F;
active proctype function_POWER_SPARE_4()
{
  int _not_base=0;
  loop:  atomic{
       WAIT_ACTIVATION(activation_xft)
       if
          ...
  }
```

+

```
Globally defined variables
local variables
C functions
C Macros
```

Promela Model Creation - continued

3. Add Non-Determinism

RespGyroRepair.pr

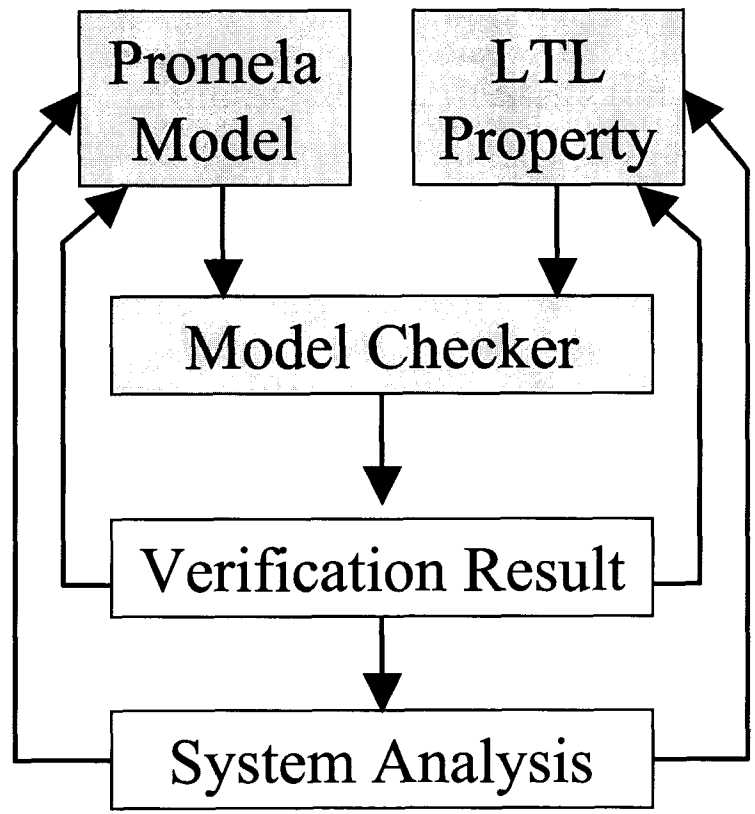
```
if
    :: c_code { IdGyro = 0; }
    :: c_code { IdGyro = 1; }
    :: c_code { IdGyro = 2; }
    :: c_code { IdGyro = 3; }
fi;

if
    :: c_code { MaxRetryEscape = 1; }
    :: c_code { MaxRetryEscape = 2; }
    :: c_code { MaxRetryEscape = 3; }
    :: c_code { MaxRetryEscape = 4; }
    :: c_code { MaxRetryEscape = 5; }
fi;
```

- Thus a system can be exercised in Spin with all possible ranges of values
- An integrated system will provide visibility into the real system

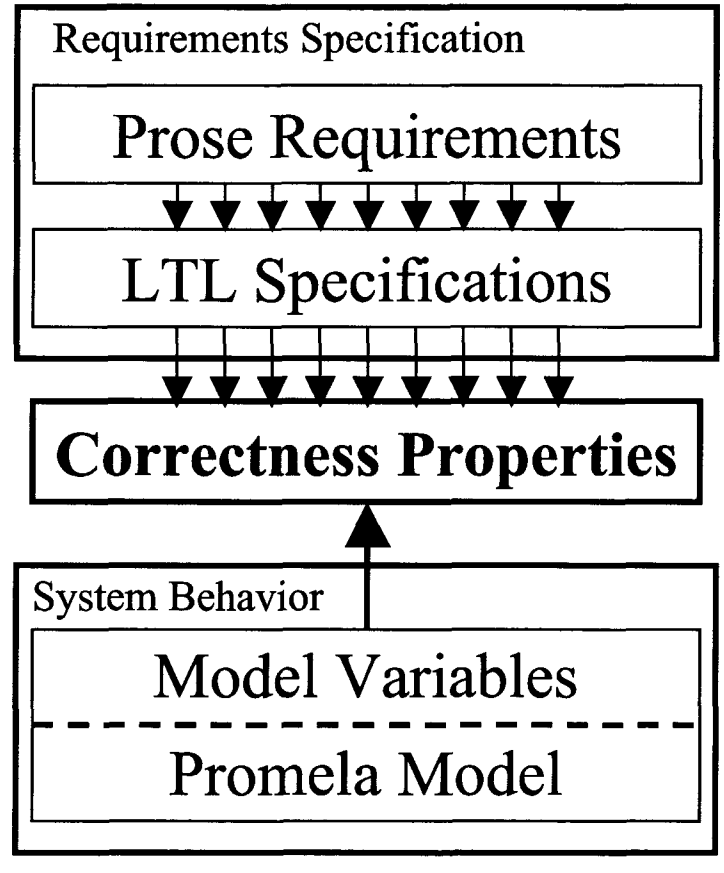
Model Checking of Linear Temporal Logic (LTL) Correctness Properties

- LTL Properties
  - Formally specify requirements
  - Automatically verifiable
  - Suitable for Model Checking
  - Verified over a Promela model
- Verification Results
  - Iterative model refinement
  - Iterative property refinement
  - Results applied to system
- System Analysis
  - New Properties of Interest
  - Model Additions



Correctness Property Generation

- **System Behavior**
  - Expressed in Promela
  - Observed in Model Variables
- **Requirements Specification**
  - Expressed in English Text
  - Formally Specified in LTL
- **Correctness Property (CP)**
  - LTL Specifications adapted to available model variables
    - Yields model specific LTL
    - Requirement equivalence preserved
  - Model Checker verifies property



Note: 'prop_list' and 'propositions' files auto-generated by HiVy provide state and event model variable definitions for use in CPs

LTL Operators

LTL operators express how system events and states relate temporally (in time).

[]	always	[]p	p remains invariantly true
<>	eventually	<>p	p will become true at least once
U	until	p U q	p will remain true until q becomes true
→	implies	p → q	(!p || q) *if p is true then q is true*

Also legal in LTL:

|| (logical OR), **&&** (logical AND) and **!** (negation)

Example Property

- Prose Requirement
  - No repair response shall attempt recovery actions for an element unless the corresponding urgency has been assessed as either need it or want it

- Formal LTL Specification
  - **❑((RecoveryAction) → UrgencyNeedIt || UrgencyWantIt)**

- Correctness Property
  - **❑(RunFPSeq==True → Urgency==NeedIt || Urgency==WantIt)**
    - "NeedIt" and "WantIt" are integer constants

- Verification Result
  - Spin reports that the property holds over the Promela model
  - Requirement verified with respect to System Model behavior

Conclusion/Future Work

- There is incentive to apply model checking techniques toward the verification and validation of mission-critical flight software such as DI FP
- The HiVy Tool set helps automate the generation of Promela models
- LTL Correctness Properties formalize the connection between design requirements and verification articles
- Approaching the initial design process with a model-based techniques will make it easier to use model checking

- We are continuing to verify DI FP response models against CPs
- We are building up small "systems" of responses for verification with Spin; responses are coordinated by a Fault Protection Engine, also included in the integrated Promela model
- We seek to quantify the benefits of model checking for DI FP at the conclusion of our effort

We acknowledge the contributions of E. Benowitz, G. Holzmann, A. Oyake, J. Powell, & M. Smith to this work.