Presented by

Christopher Slack
Computational Analysis Expert, Fuel Systems
Airbus Operations Ltd

Model Based Design for Fuel System Development

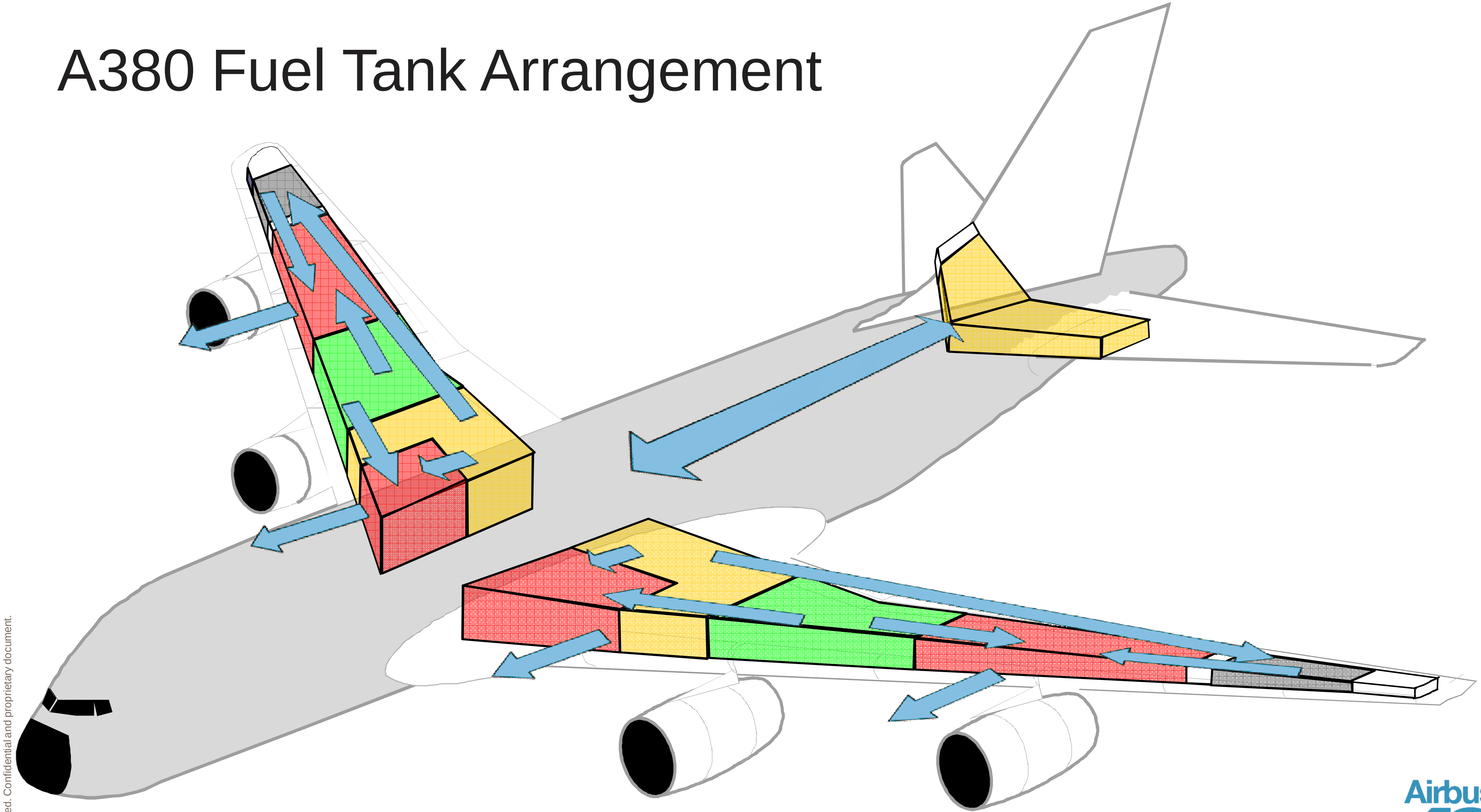
Use of Stateflow and Mathworks Toolsets

Airbus 40 years of innovation

AIRBUS

Why is it so complicated?

A380 Fuel Tank Arrangement



A/C Maximum Weight : 560 Tonnes.
Max Fuel Capacity : 250 Tonnes (320,000 Litres)
Payload : 550 Passengers + 35 Tonnes Cargo

Why is it so complicated?

Multiple engines & tanks

Numerous functions to manage (some of which are safety critical)

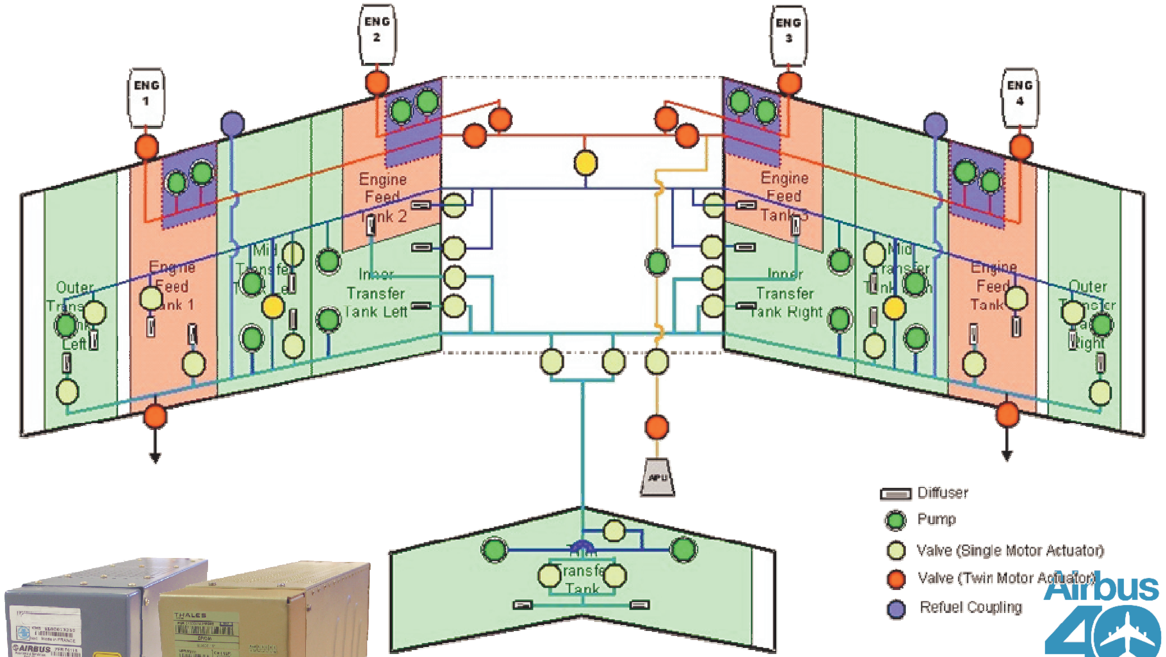
- Fuel Measurement
- Fuel Management
- CG control
- Refuel/Defuel
- Wing bending relief
- Communications to/from more than 20 other systems
- Hot/Cold Fuel Workarounds
- Self-test/Built in Test Equipment
- Failure Workarounds etc. etc.

Hundreds of individual pieces of equipment to manage

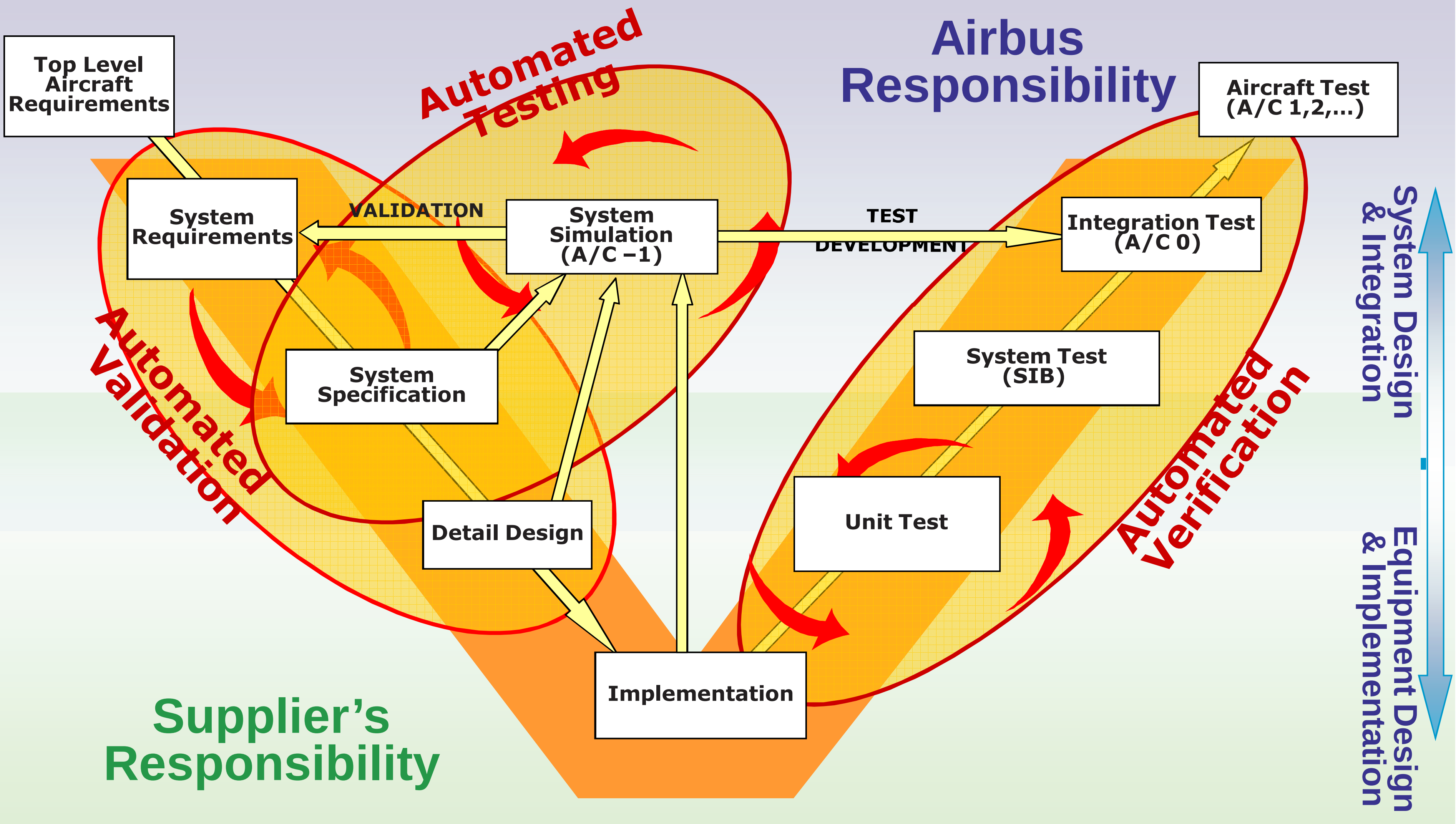
- A380; 13 tanks, 21 pumps and 43 valves

Safe Operation with Multiple Equipment Failures

- No turn-back/diversion
- No increased crew workload
- 2800 cases of "MMEL" + Single Failure

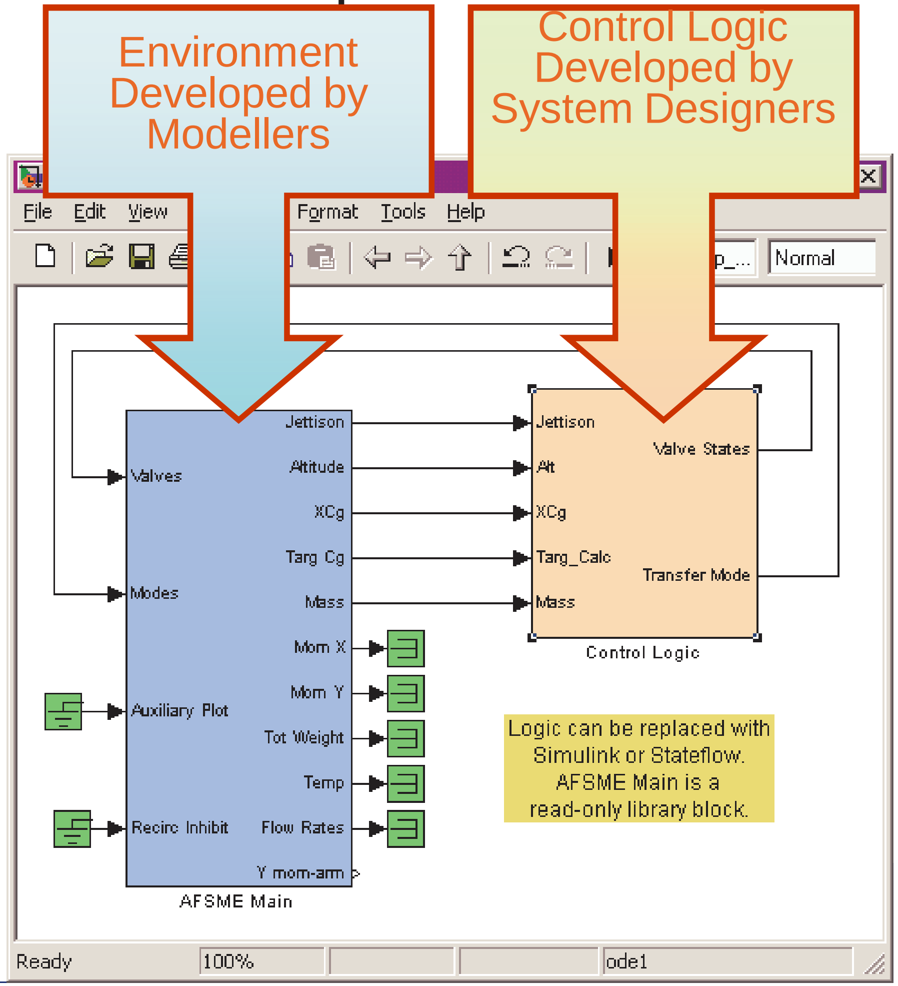


Systems Engineering V-Cycle



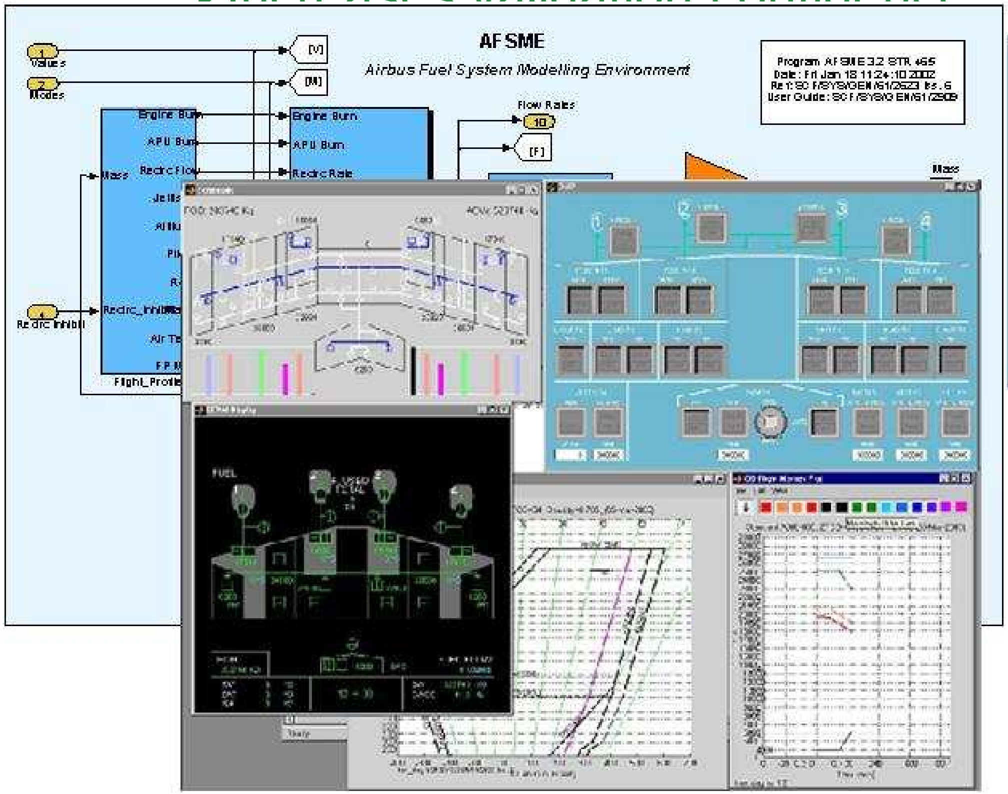
Model Based Development

- Development of Basic Operating Sequence.
  - Normal and Failure Operating Modes
- Rapid Prototyping of Transfer & Refuel Requirements.
- Simulink/Stateflow Application
  - Platform Independent
- Control Logic separated from Aircraft Environment
  - Engineers concentrate on System Design
  - Specialist Modellers concentrate on Environment Fidelity
- Statecharts control behaviour
  - Easier to use than “Enabled Subsystems”

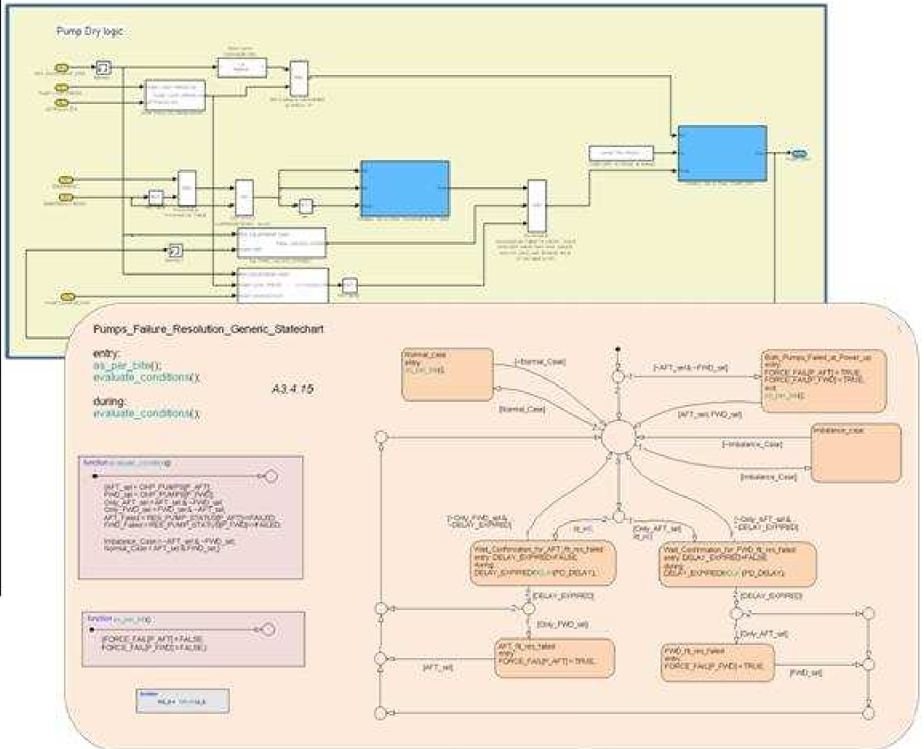


Model Based Design - In Practice

- Statecharts control behaviour
  - Easier than Enabled/Triggered Subsystems
- Enhanced Validation
  - Statechart representation can be clearer and less ambiguous
  - Increases validation confidence



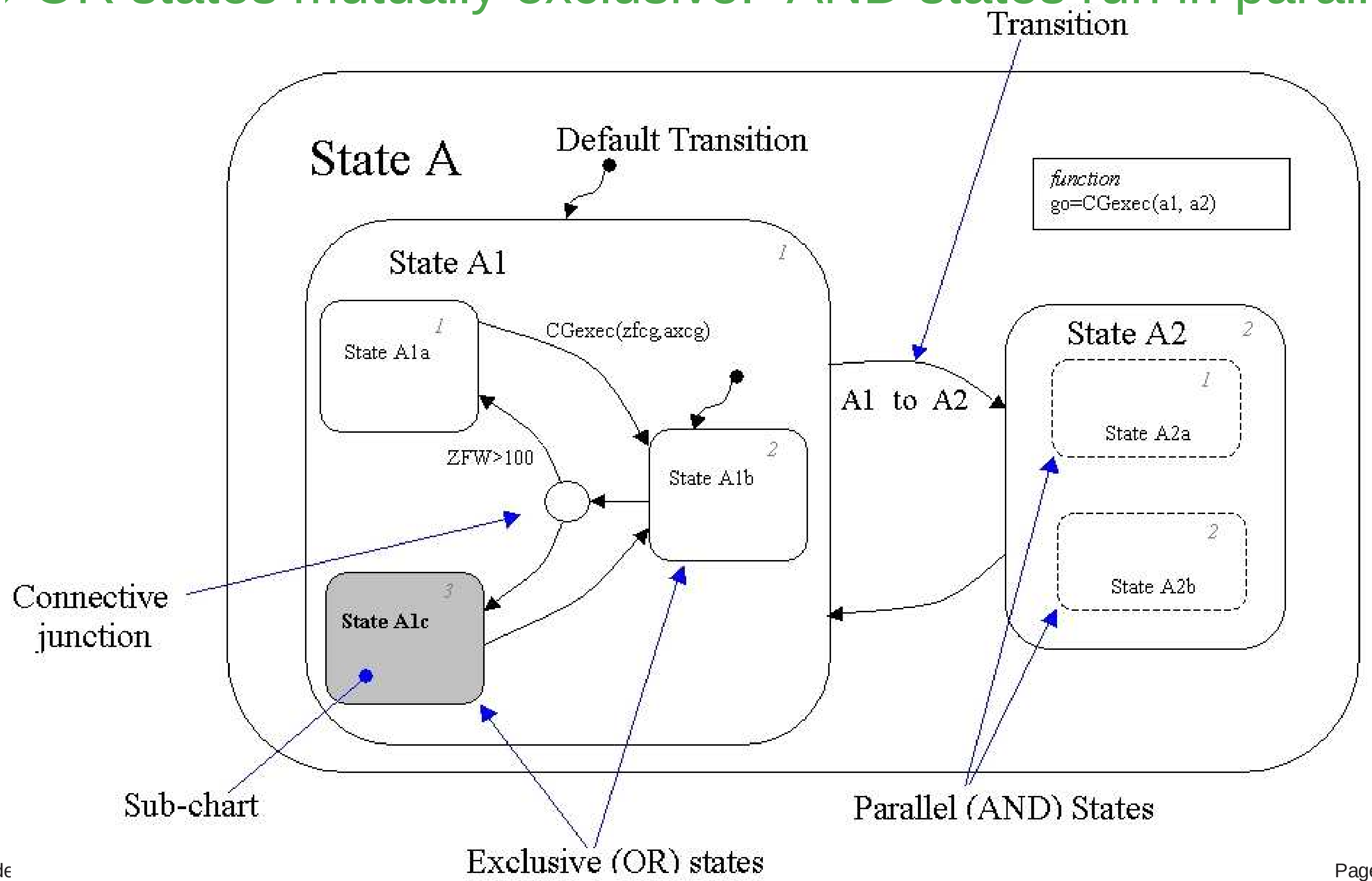
Fuel System Modelling Environment



Control Function Design

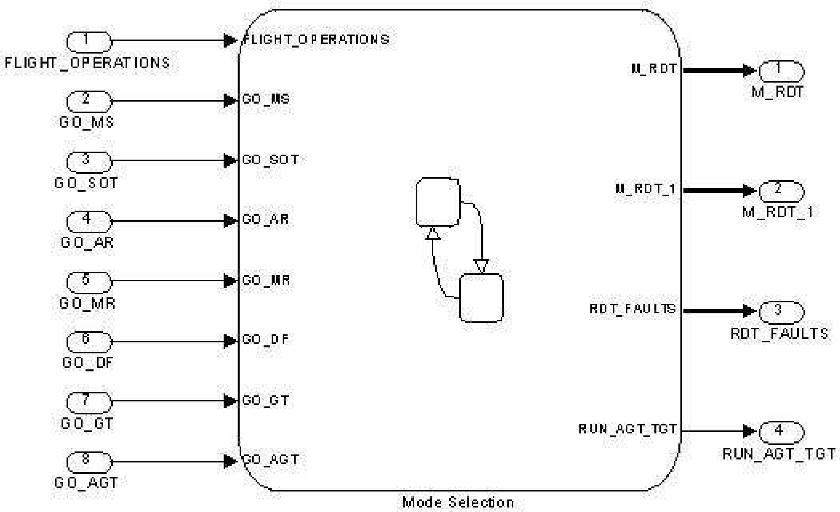
How Stateflow is Used

- Definition of Statechart
  - Describes the system states, rather than the functionality
  - Arrows show transitions between states, not data flow paths
  - OR states mutually exclusive. AND states run in parallel

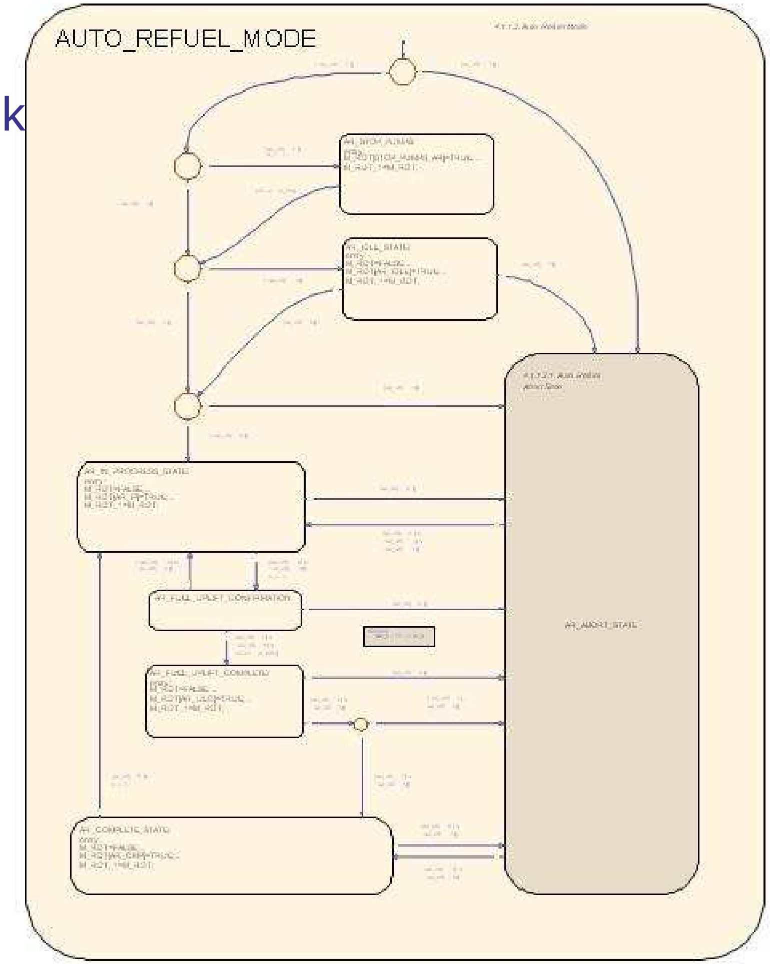


How Stateflow is Used

- Aircraft Fuel System Statecharts:
  - Linked to Requirements Database (DOORS)
  - Separate Chart for each Major A/C Function
  - Transition booleans calculated within Simulink
  - Input into Stateflow Chart
    - Driven behaviour of stateflow logic separated from driving conditions
    - Allows easier readability and testing



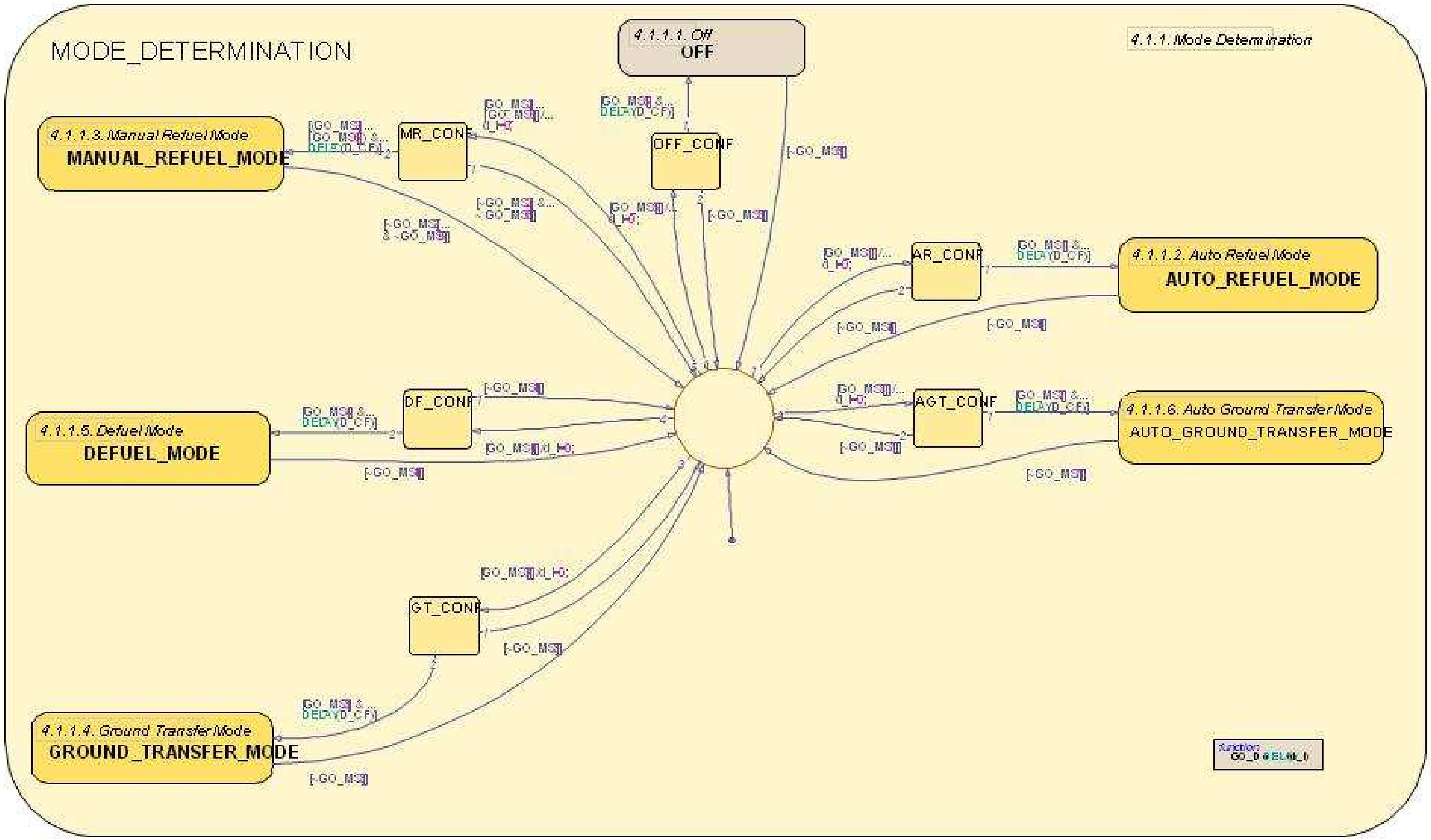
Top Level Chart



AIRBUS

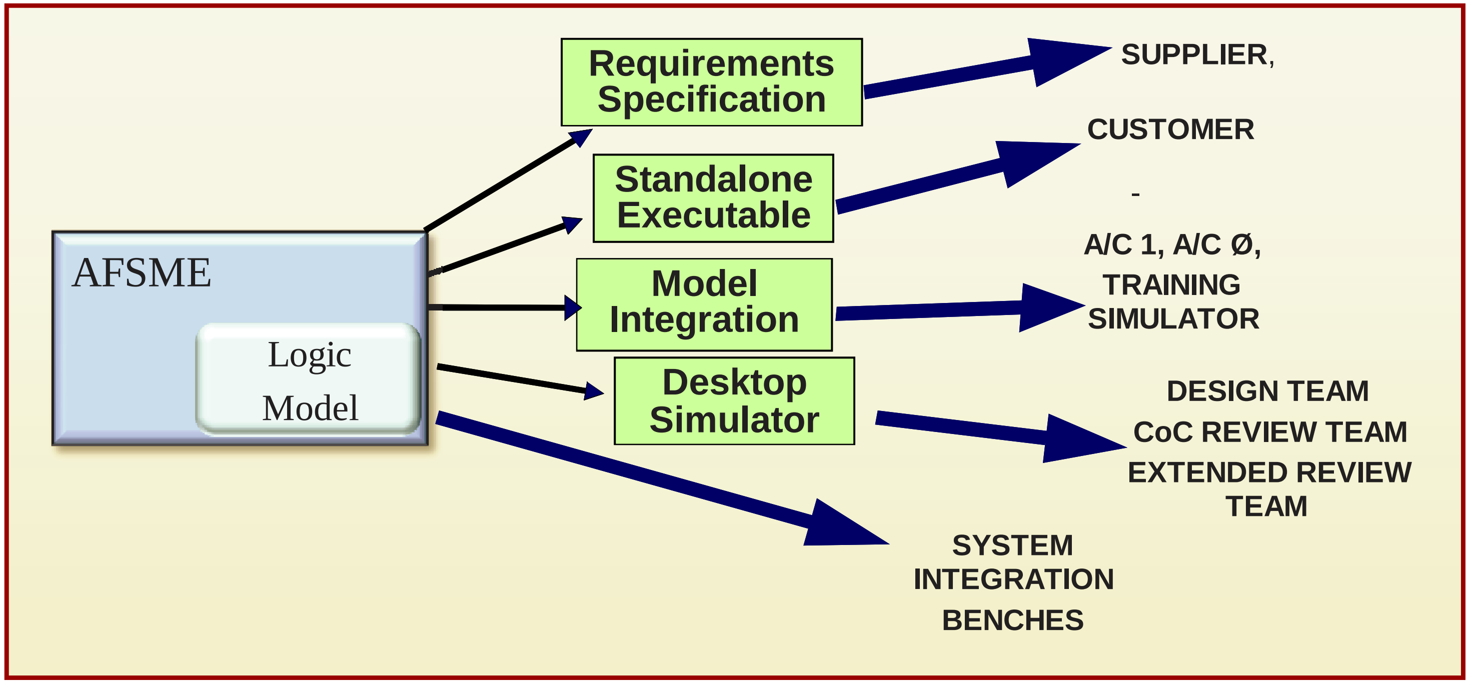
How Stateflow is Used

- Aircraft Fuel System Statecharts: On Ground Operations
  - Clean Layout – Sub-System dependencies unambiguous
  - System behaviour defined as mutually exclusive (OR) states.
  - System cannot be in (e.g.) “Refuel” and “Defuel” modes simultaneously



Where & When is Stateflow Used

- Model Re-Use
  - The model represents functional requirements
    - Can be used directly in a number of simulators:



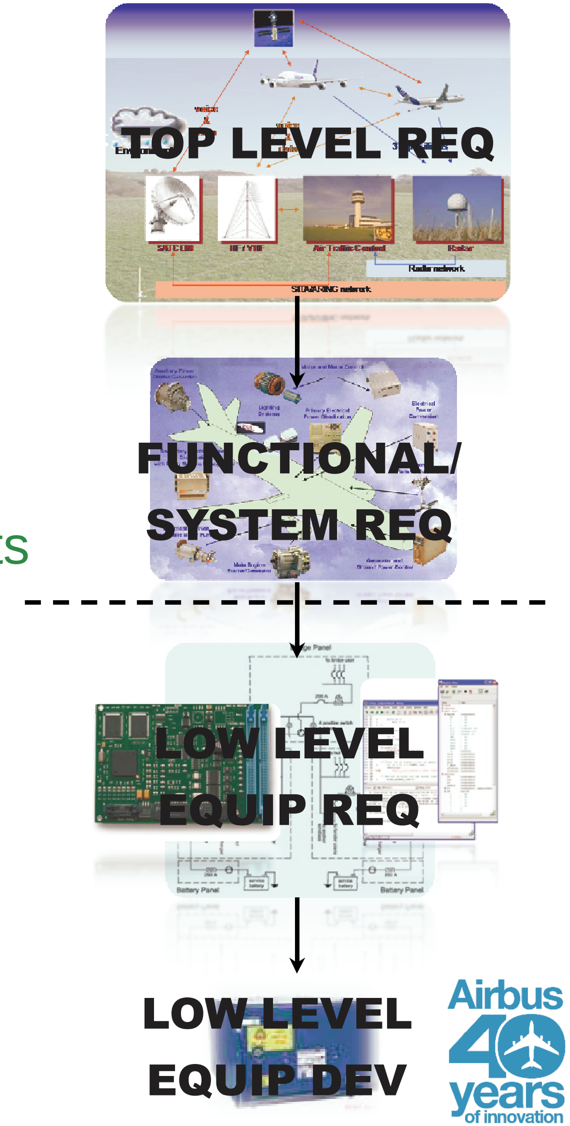
- Model is a "Write Once - Use Many" entity
- Changes to base model propagated down to each instance of use

Where & When is Stateflow Used

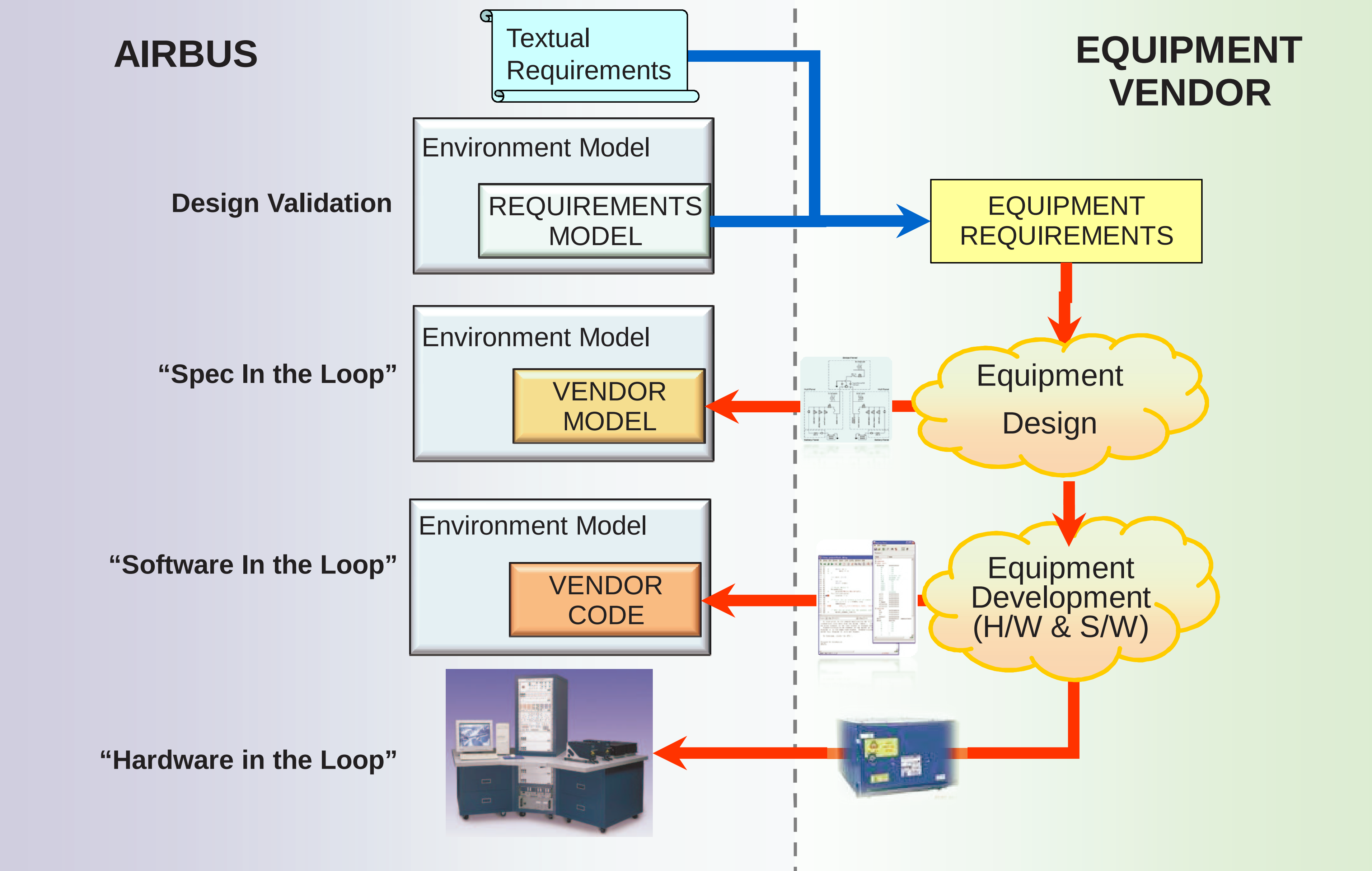
- Integrated Desktop Simulator
  - Requirements & Environment Model
  - Add Interfaces and other functionality
  - AutoCode using Real-Time Workshop
- Aircraft -1
  - Entire Software Simulation
  - Interfaces Identical to Aircraft
- Fuel System Test Benches
  - Verification of single equipment
- Aircraft-0 (Iron Bird)
  - Cockpit Avionics & Displays
  - All Systems Integrated (real & simulated)
- Full Flight Simulator
  - Single model for all platforms

Where & When is Stateflow Used

- Model Based Design Approach (Ideal)
  - Develop models to specify system functionality
    - Describes behavioural & functional aspects
  - Details become the System (and Sub-System) Requirements
    - Exercise the model to Validate Requirements
  - Delivered to Fuel System Supplier
    - Model contains Requirements *and* intent
    - Model execution provides system understanding
    - Minimal Work to turn into Code
    - Separate layer for independent validation

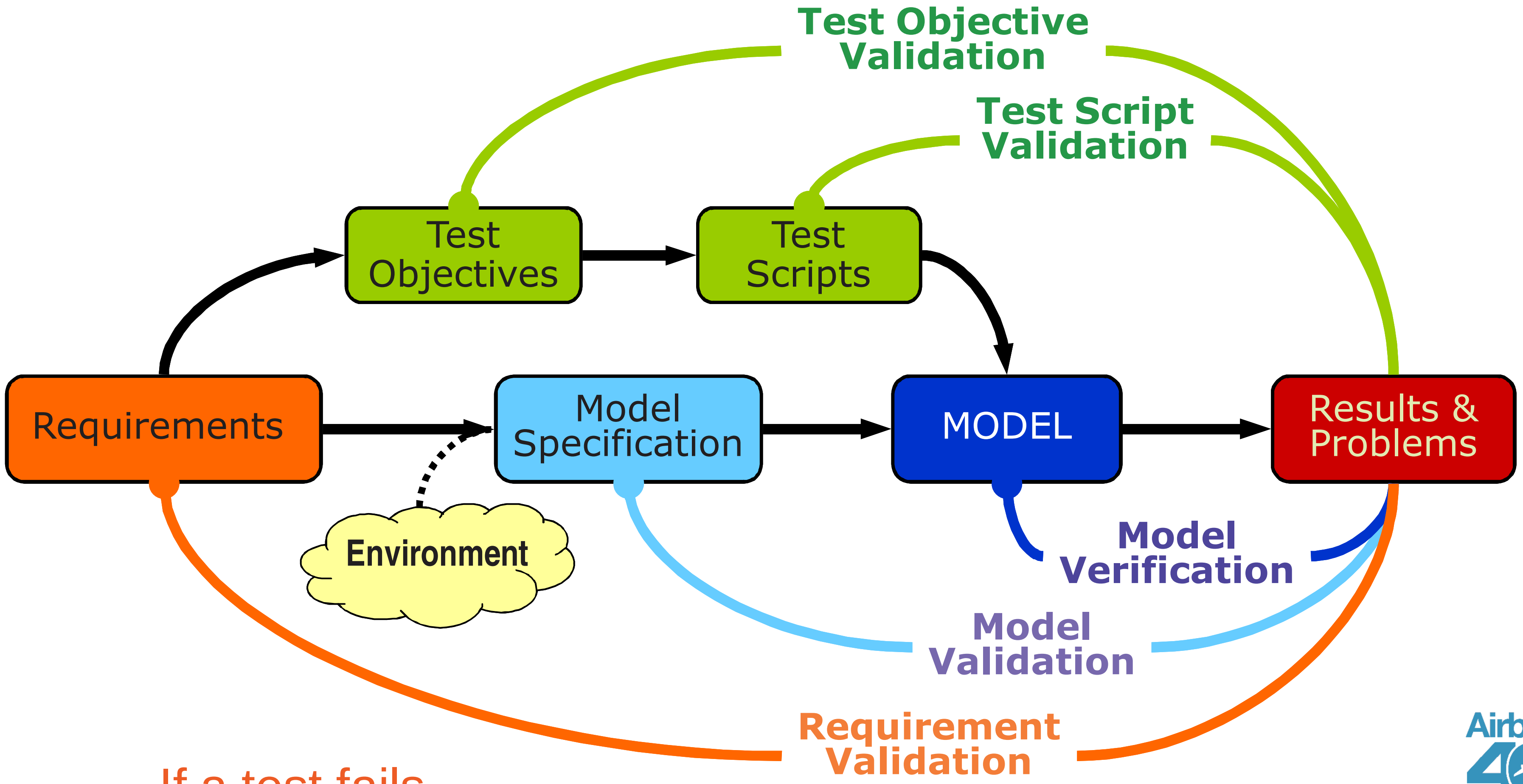


Early Supplier Involvement



Model Development Process

When the model is the requirements, the distinction between "Model Verification" and "Requirements Validation" is somewhat blurred



If a test fails –
is the requirement, the model or the test at fault?

Aviation Authorities View of MBD

- Certification Review Item : F17/ F22

"The complexity of specification written with formalised language raises the need for higher level specification description containing all the requirements implemented in the formalised specification"

- Effectively states that a model is only an implementation of unwritten requirements.
  - We need a model *and* textual requirements in order to sufficiently validate a system in terms of ARP4754/DO178B
    - E.g. Non-functional requirements difficult to model.
  - Affects our strategy for MBD

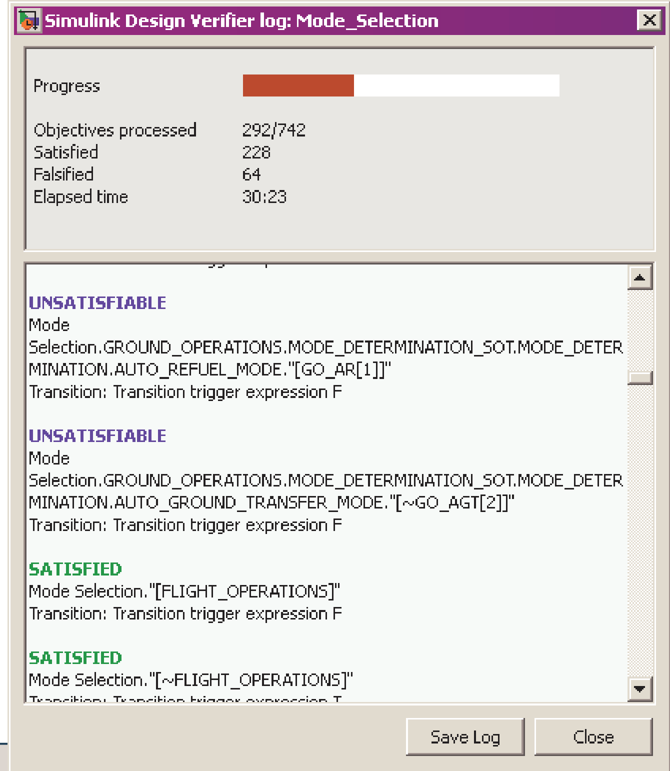
- This CRI specifically targets Software Specifications using SAO/SCADE/LDS
- But applied to SSRD developments using Stateflow.

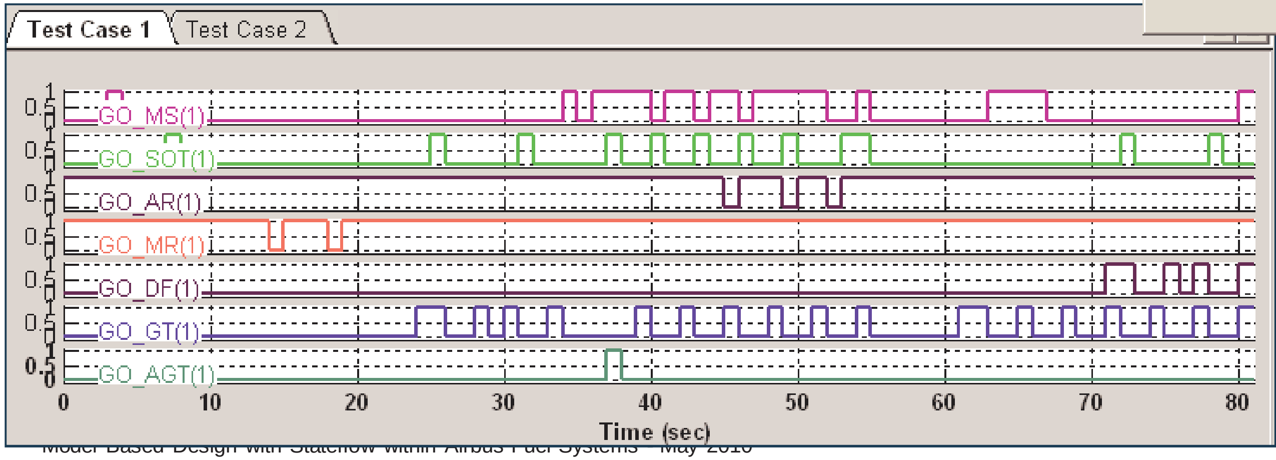
Advancements; Model Verification

- Recent use of “Simulink Design Verifier” (SLDV)
  - “Prover” Technology previously used with Esterel SCADE
  - Experimentations first with R2007b
    - Proof of concept, but unable to handle “large” models
  - Enhancements made in each release; R2008a, R2008b, R2009a, R2009b...
    - Now considered mature enough for industrial applications
- Two modes of Operation:
  - Test Generation
    - Tries to generate a minimal set of tests that provide maximal coverage.
      - Uses Modified Condition/Decision Coverage (MC/DC)
      - Conditional Transitions, Substate executed, Substate exited
  - Formal Proof
    - User specifies a property
    - SLDV tries to find a combination of inputs that falsifies that property

Model Verification – Test Generation

- Produces report showing:
  - "Objectives Satisfied"
    - – A test has been found that exercises a particular state or transition
  - "Objectives Proven Unsatisfiable"
    - Untestable/unreachable state or transition
  - "Objectives Undecided"
    - Could not determine an outcome in the time available
- Test harness Creation



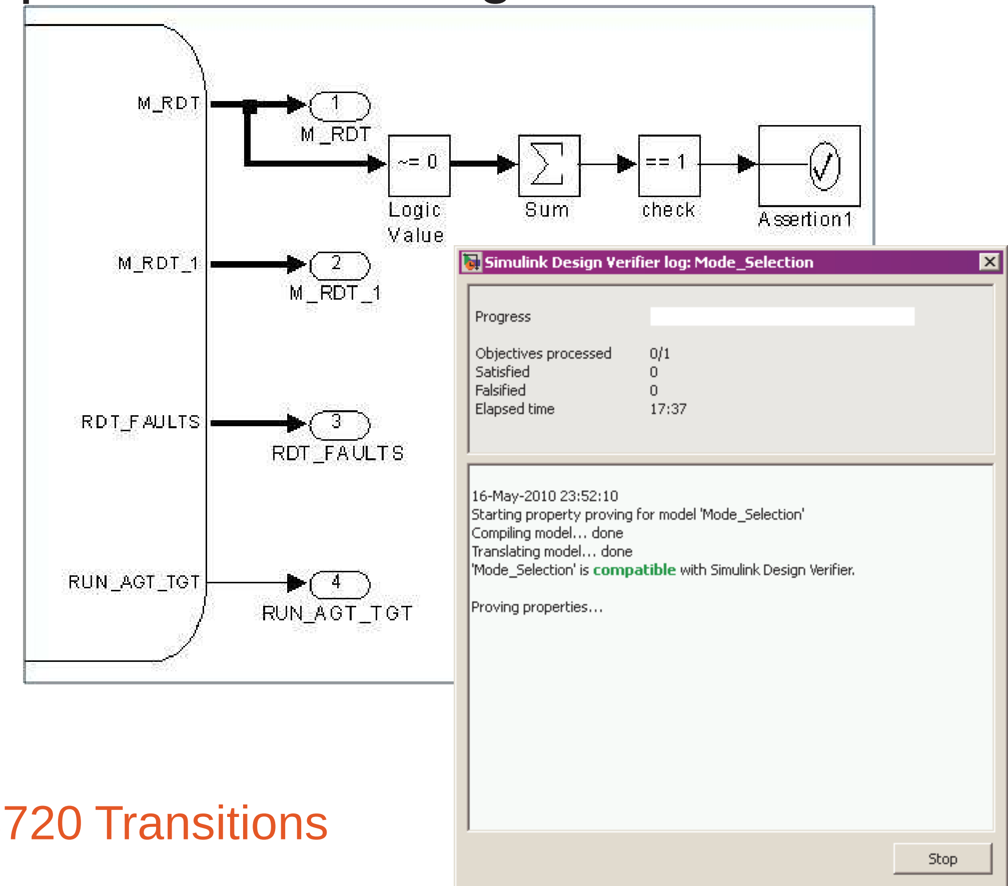


Subsystem comprising
1 Chart
102 States
186 Transitions

Model Verification - Model Proof

- Define Proof Objectives and Assertions
  - Using Simulink/Stateflow/Matlab
  - Based on higher level (inc. safety) requirements
- Proof Objective allows multiple values & ranges
- Example:
  - Output Array of booleans mutually exclusive
- If counterexample found, creates test harness
  - Can take a very long time...

"Simple" subchart of 102 States 186 Transitions – no counterexample found after 30 minutes.



Full A380 Fuel Model :

45 Charts, 5945 States and 8720 Transitions

Problems Encountered

- Process Problems
  - Model Style Guidelines need to be defined and rigidly enforced
    - Matlab code and “test” blocks find their way in to the model
  - Pure design requirements model unable to be exercised
    - “Extra” elements added to get it to operate. Need to clearly identify what are requirements and what are the “extras”.
  - Use of global (workspace) data
    - Obfuscates the system interfaces
  - Need to ensure that valves/pumps return to default values on exit of states
    - Multiple Exit Paths need to be considered
    - Implied Requirements
  - Keeping track of model updates with multiple designers
    - Potentially a configuration nightmare
    - Eased with the use of Model Reference

Problems Encountered

- Technical Problems
  - Fuel System Vendor uses SCADE for Qualified Code Gen.
    - No easy "auto" translator from Simulink/Stateflow into SCADE/SSM.
    - Hand conversion could introduce errors.
    - Vendors can develop "clever" tools for auto-conversion of charts
    - Aircraft Program "tied in" to a particular release of Matlab

      A380 Fuel still uses Matlab R12
  - Model Proof consumes lots of resources...
    - Memory
    - CPU Time
    - Large models need 64bit + lots of RAM
  - Extracting stateflow sub-charts quite a manual process
    - Improvements to toolset is making life easier

Lessons Learnt - Model Based Design

- Model build process can reveal anomalies/ambiguities
  - Validation for free
    - Identify Assumptions separately from requirements
    - Identify Executable Implementation from Requirements
- Model Architecture
  - Separate Requirements Model from Environment Model
  - Separate real interfaces from simulation/test interfaces
- Validation Testing
  - A test that is more complex than that being tested is probably wrong
  - Easy to be caught in the trap of “Test for Success”
    - Testing for intentional, but not unintentional behaviour
    - Project managers demand simple progress metrics

Lessons Learnt – System Design

- System Designers focus on Designing the System
  - The System Model is the System Requirements
    - – But extra functionality required to exercise model are not requirements
    - – Non-Requirements need clear labelling
- Discontinuity between Design and Implementation
  - Detailed Models required for Integration Simulators
    - – Required before availability of equipment
    - – Need to create models of *potential* implementation
- Easy for Designers can be Difficult for Simulators
  - Matlab Function Blocks
  - M-File S-Functions
  - Test Harnesses
    - – Can break the automatic code generators
- Model Size Increases Monotonically
    - – Can break toolsets e.g. SLDV

Summary – Model Based Design

- It's as bad to talk about "M&S" as it is to say "V&V"
  - Two distinct parts of an end-to-end process.
  - Two distinct methods of implementation, results and consequences
  - Modelling is a Means to an End – not an End in itself

- Difficult to distinguish between Verification and Validation
  - Each requirement has a validation statement
    - I.e. A "test"
  - If a test fails, have you performed:
    - Validation of the requirement?
    - Verification of the model?
    - Validation of the test?

Thankyou

AIRBUS

AN EADS JOINT COMPANY
WITH BAE SYSTEMS

Airbus 40 years of innovation