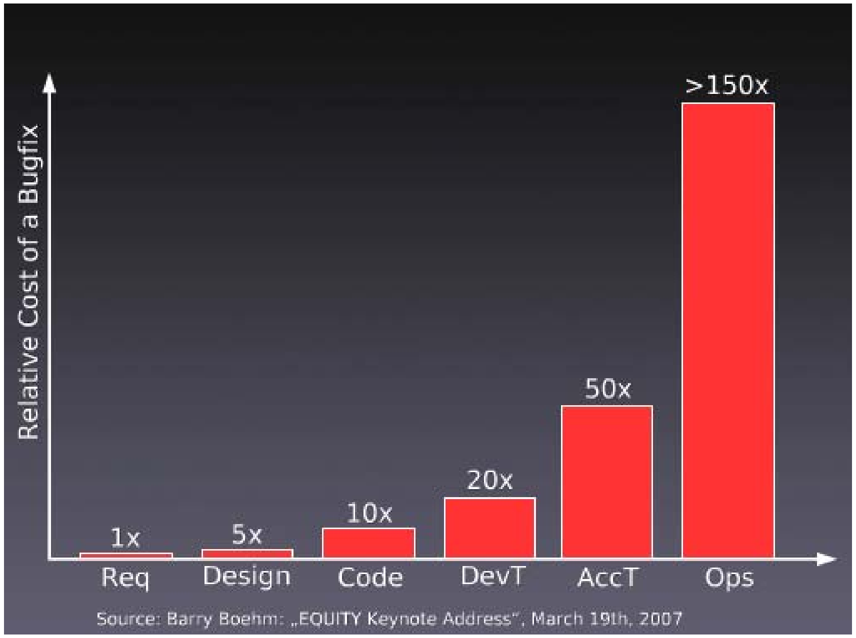
Defining and Verifying System Architectures using Model Based Executable Specifications

Pete Karousos
Lockheed Martin Missiles & Fire Control
peter.g.karousos@lmco.com

LOCKHEED MARTIN

Motivation

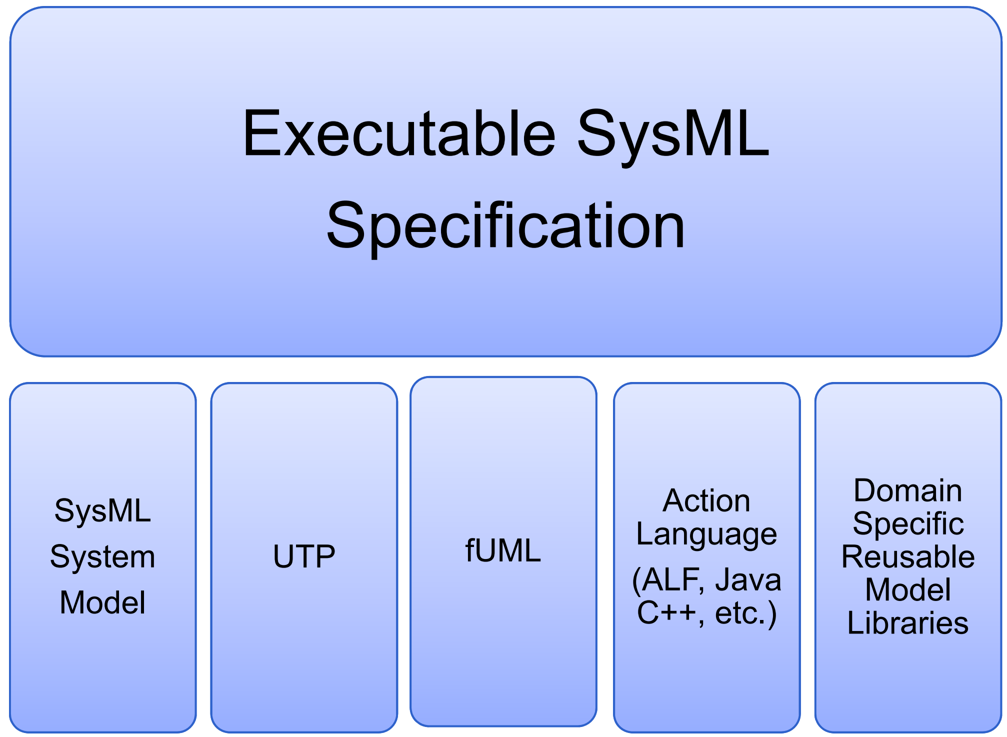
- Text Based Requirements:
  - Communication/interpretation problems
  - Incorrect or incomplete requirements
  - May be difficult to test
- Cost of fixing defects in requirements increases exponentially with time
- Static SysML & UML Models
  - Reduce ambiguities
  - Link requirements to architecture, behavior and parametric trades
  - Being static could have errors
- Executing these models provides confidence the behaviors in the model are doing the right thing



Fixing Specifications Defects Later in Development is Prohibitively Expensive

What is an Executable SysML Specification?

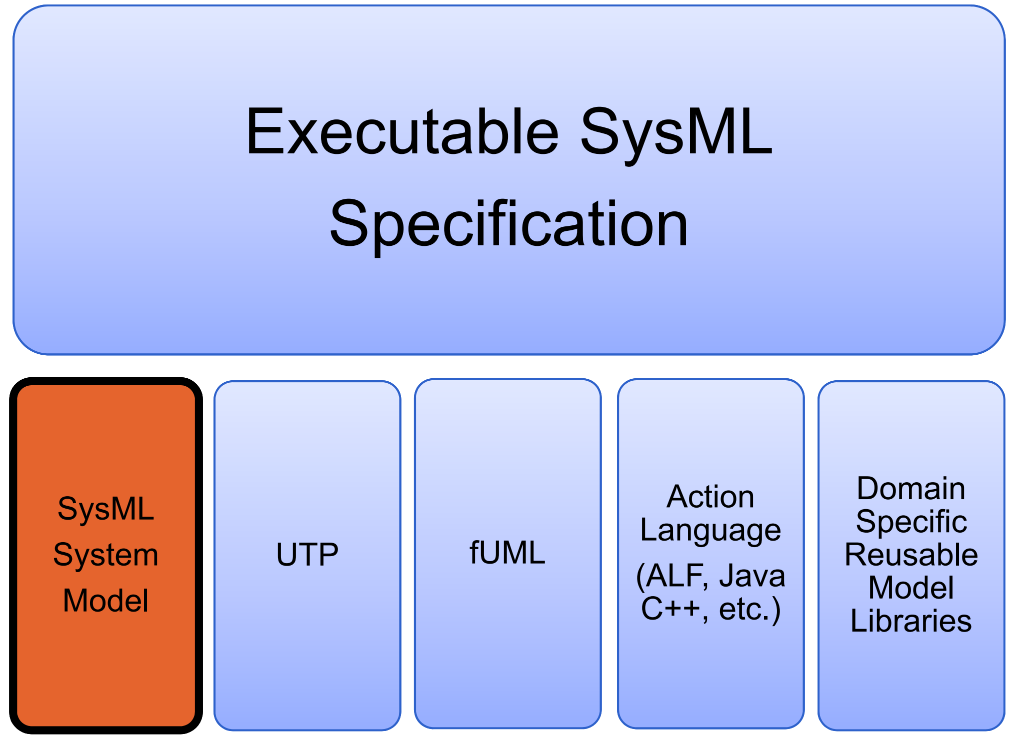
- Tests the Logic of Behavioral Requirements
- Combines Graphical and Textual Programming
- Standards based for tool interoperability
- Scalable – any combination of
  - System of Systems
  - System
  - Subsystem
  - Component



Platform Independent Standards Based Dynamic Model for Validating Requirements

Systems Modeling Language (SysML)

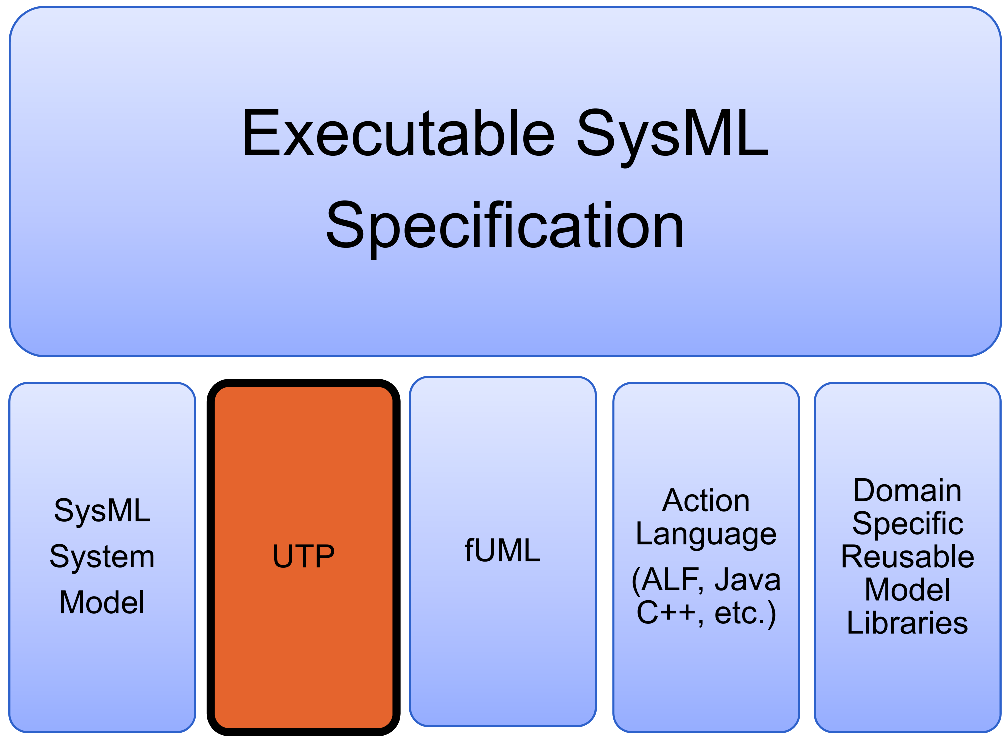
- OMG Profile of UML for Systems Engineering
  - Strategic INCOSE Decision to Customize UML
- Standard way to Model Hardware, Software, Analysis, Verification & Validation
- Provides Built-in Facilities to Model Requirements
- An Executable Specification Makes Heavy use of Activity Diagrams



SysML Provides a Standards Based Modeling Language

Unified Testing Profile (UTP)

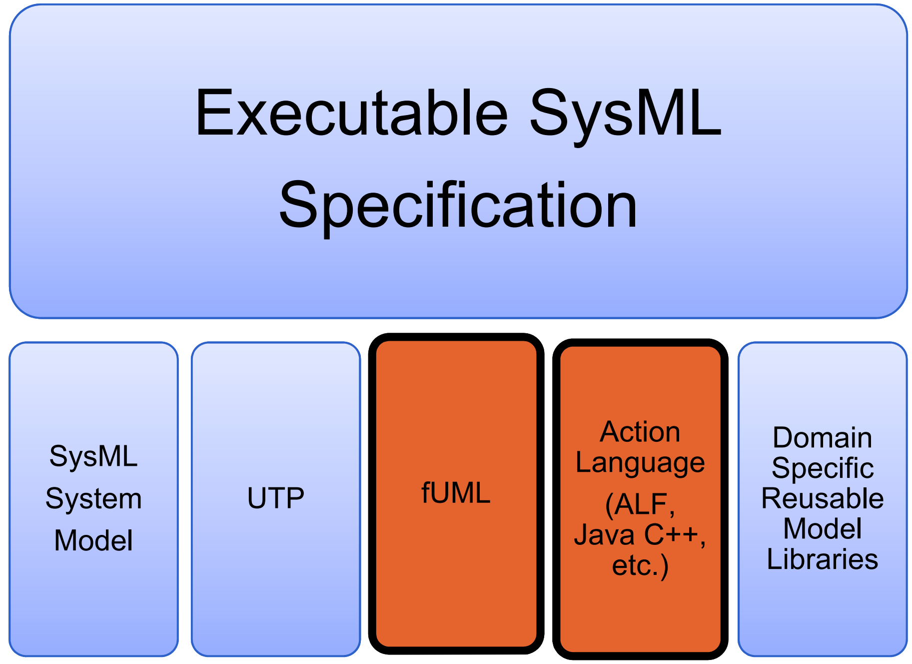
- OMG Profile of UML for Test Architectures
- Adds Specific Concepts for Testing Such as
  - Test Contexts
  - Test Components
  - Test Verdicts
- Drives Consistency Testing the Executable Specification
- Scalable Pattern
  - System of Systems
  - System
  - Subsystem
  - Component
  - Class/Function

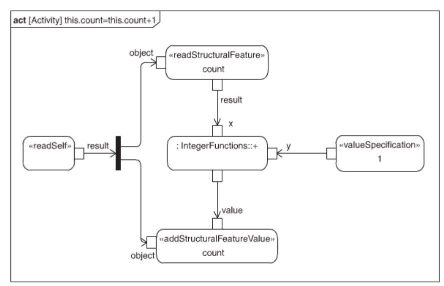


UTP Provides a Standards Based Test Architecture Framework

fUML and ALF

- OMG fUML Provides a Precise Definition of a Subset of UML
  - Allows for a fUML to Platform Translator for Execution
- OMG ALF is the Action Language for fUML
  - Using Pure fUML is Cumbersome
  - Closely Matches UML Semantics
  - Easier to Write Code for Low Level Constructs

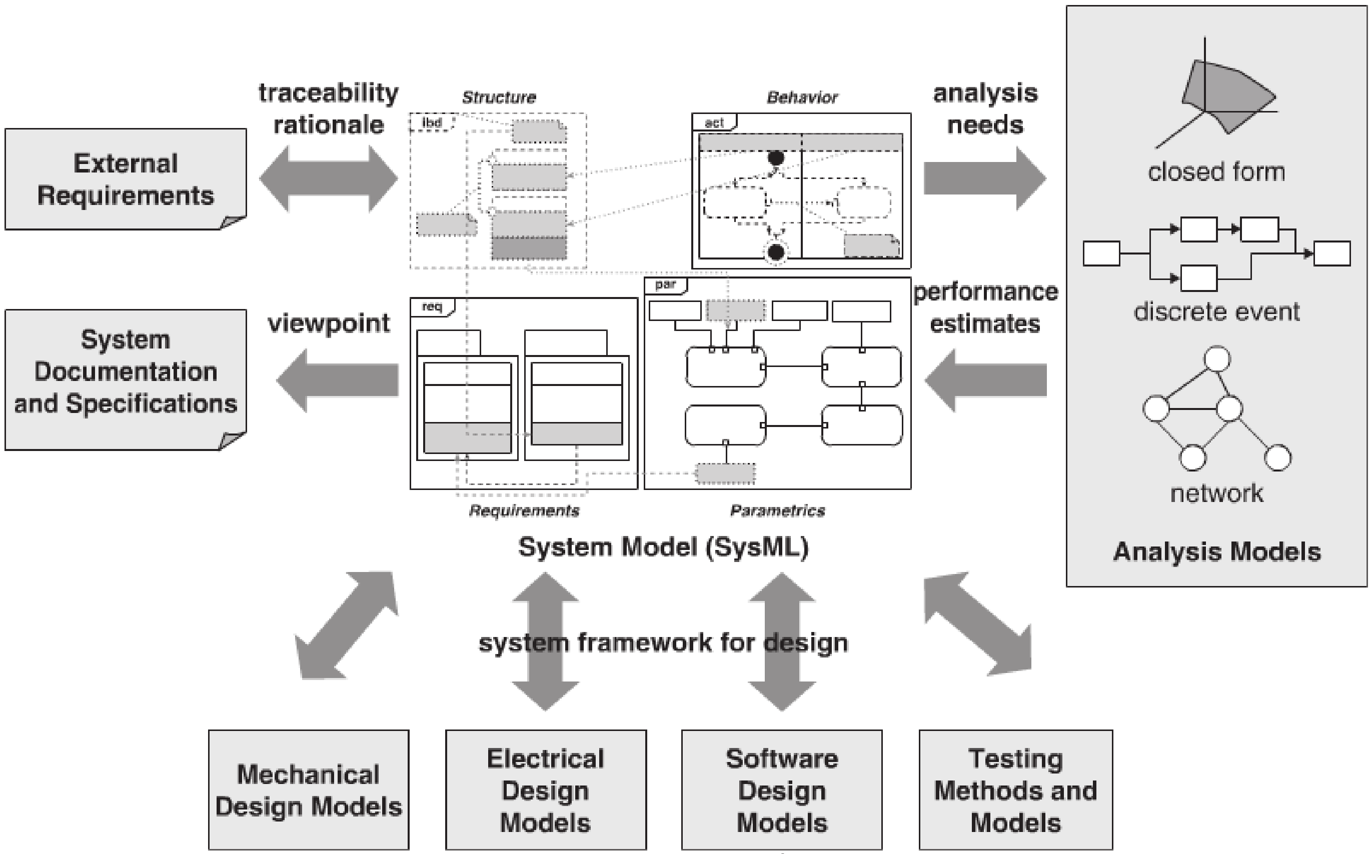





```
activity
incr_count()
{
  this.count++;
}
```

fUML/ALF Provide a Standards Based Execution Semantics

Executable Specification Context



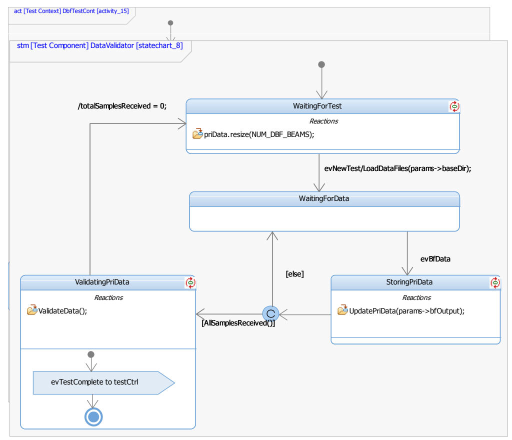
Friedenthal, Moore, and Steiner. *A Practical Guide to SysML, 2nd Edition* Morgan Kaufmann: 2011

Executable System Model Unifies Enterprise Models

Development Without Executable Specifications

System engineers provide the models

Systems engineers create the models



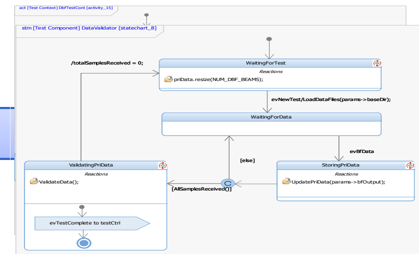
Developers create artifacts based on the models (maybe)

Designers provide feedback (maybe)

Non-executable Models may Still Have Errors in the Specifications

Modeling With Executable Specifications

Systems engineers create the specifications in SysML



Specifications validated through execution

Designers create artifacts based on the validated models

- Validated Requirements and Behavior
- Test Cases (inputs & outputs)
- Full Traces of the Execution
- Model Animation

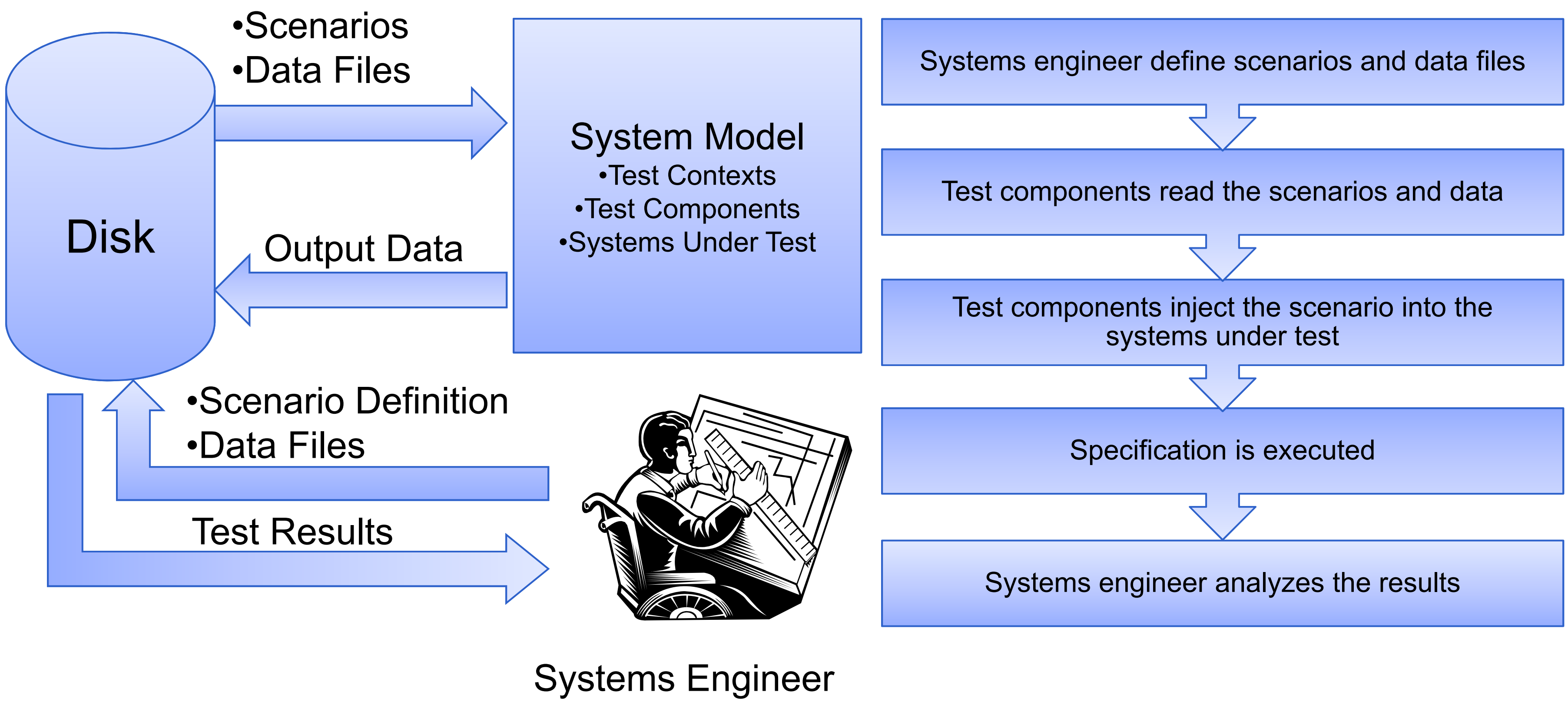
Executable Specifications Identify Specification Defects Early in Development

Steps for Creating an Executable Specification

Create System Model	• System Structure, Behavior, Requirements, Parametric Analyses
Decide what Parts of the System will be Executed	• Any combination of single activity all the way up to system of systems
Specialize the Blocks that will be Executed	• Only needs to be done if the block that will be executed needs support functions that are for testing purposes only.
Add Executable Statements	• Applied to the behavior that will be executed • Using the action language supported by the tool
Apply UML UTP	• Define test contexts • Define test components • Define scenarios
Execute the model	• Debug as necessary

Executable Models are Created by Following a Pattern

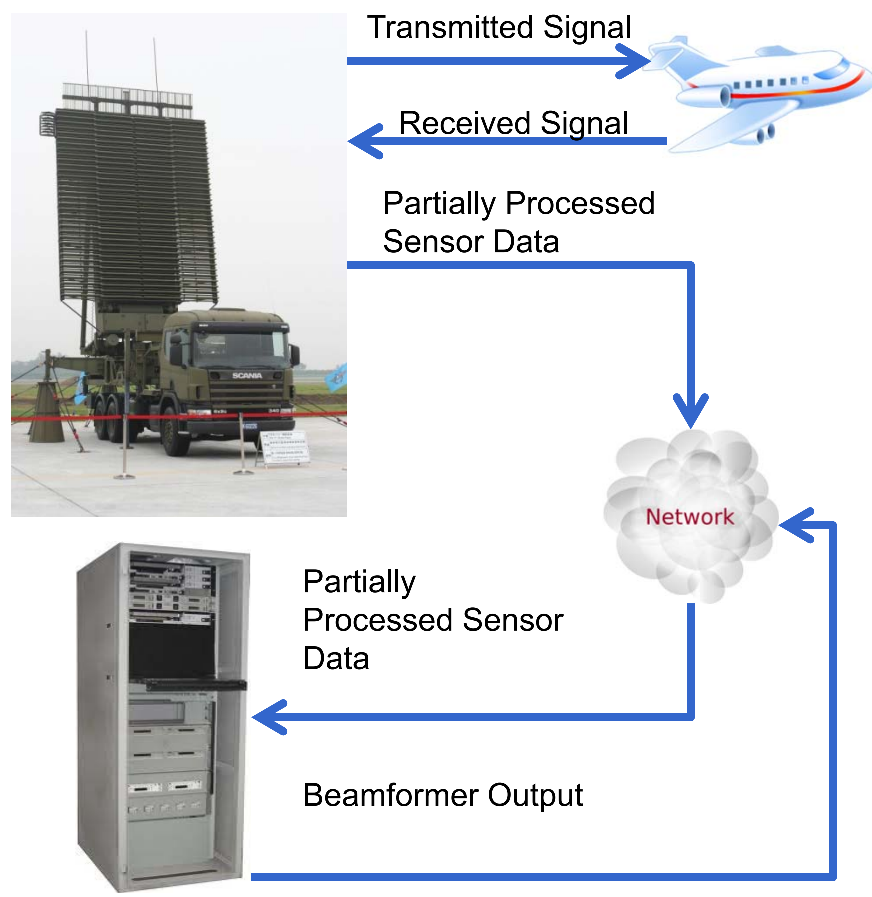
An Executable Specification Framework



File-based Testing Framework Promotes Reusability of the Test Components

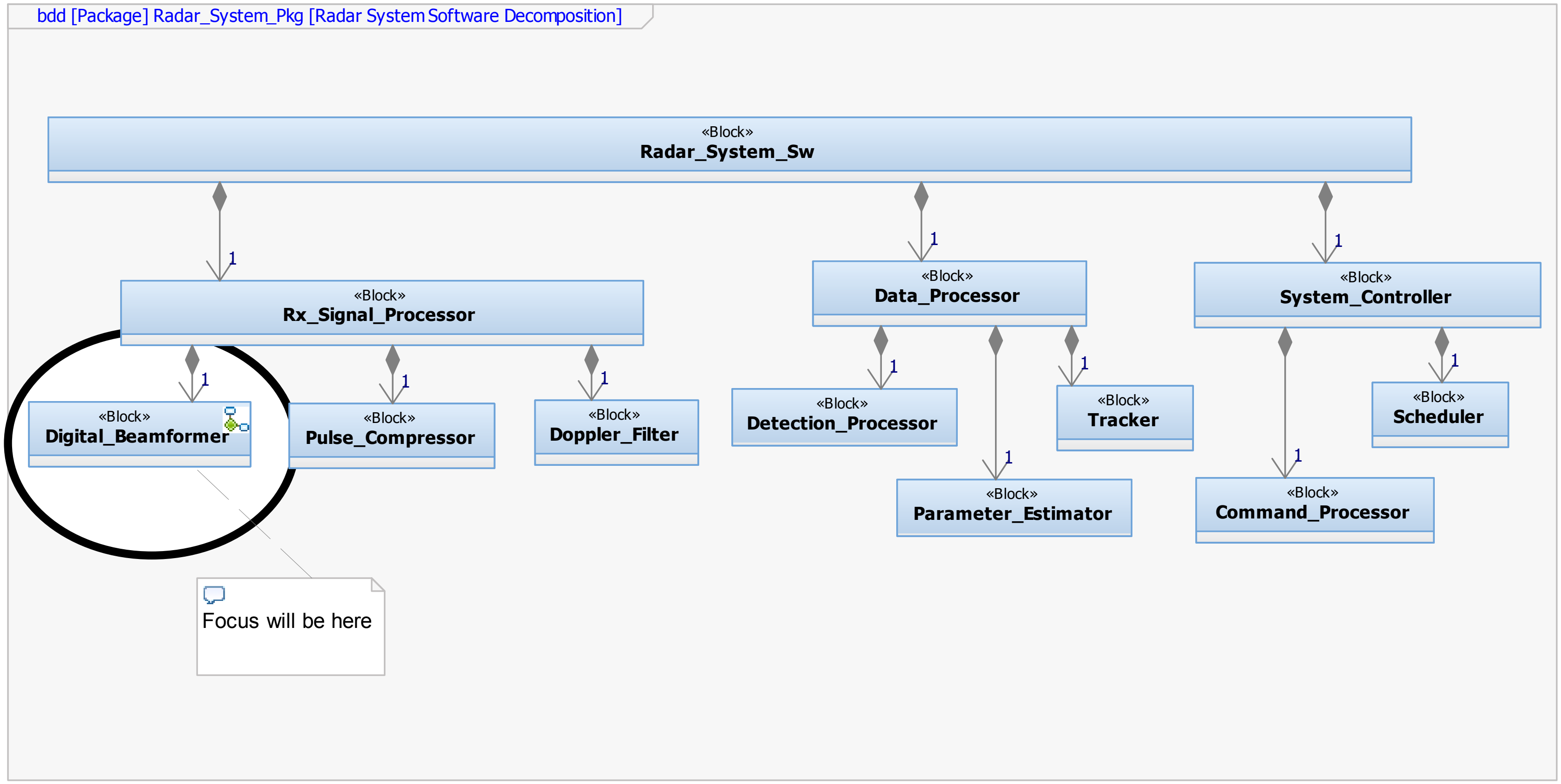
Digital Beamformer Example System

- Array sensor transmits and receives a signal
  - 1 coherent processing interval
- Signal is digitized, sampled, processed in the array
- Partially processed signal is sent over a network
- Digital signal is processed some more on an off sensor node
- Sensor data will not fit into a single packet
- Example will focus on the first part of the processing in the off sensor node working on the received signal
- This node is called the Digital Beamformer
  - It must reconstitute the data and then perform a spatial filtering operation on the data
- Verify it produces the right output based on the specification and known "truth" data



Need to Validate the Digital Beamformer SysML Model

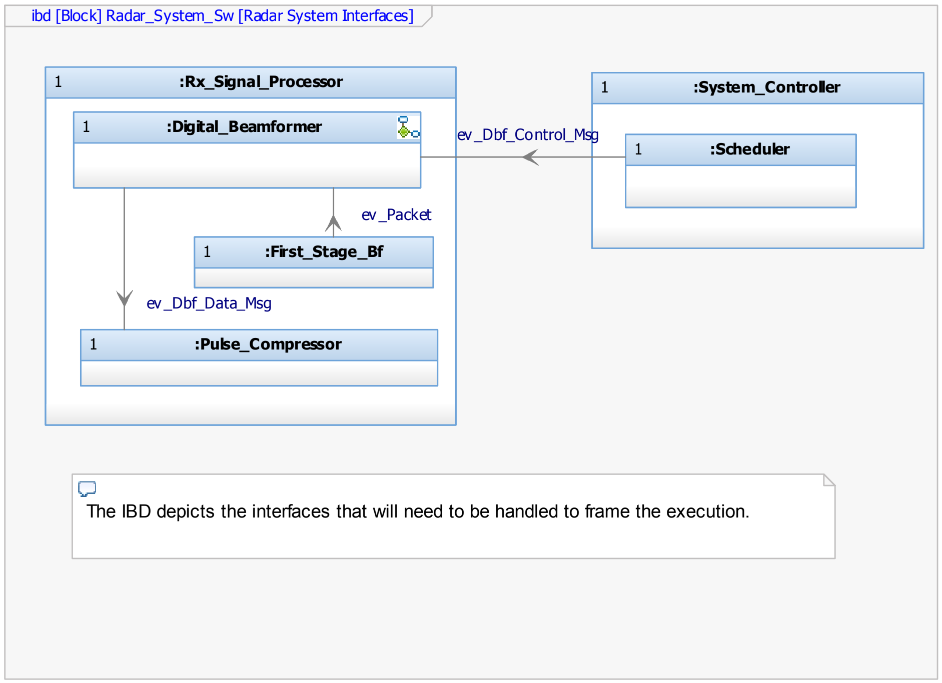
Software System Structure Definition



- Partial BDD of the system that shows the Digital Beamformer (DBF) component under test
- Any number of components across different levels of the hierarchy can be selected
- Not all components need to be executed

System Structure Defined in the System Model

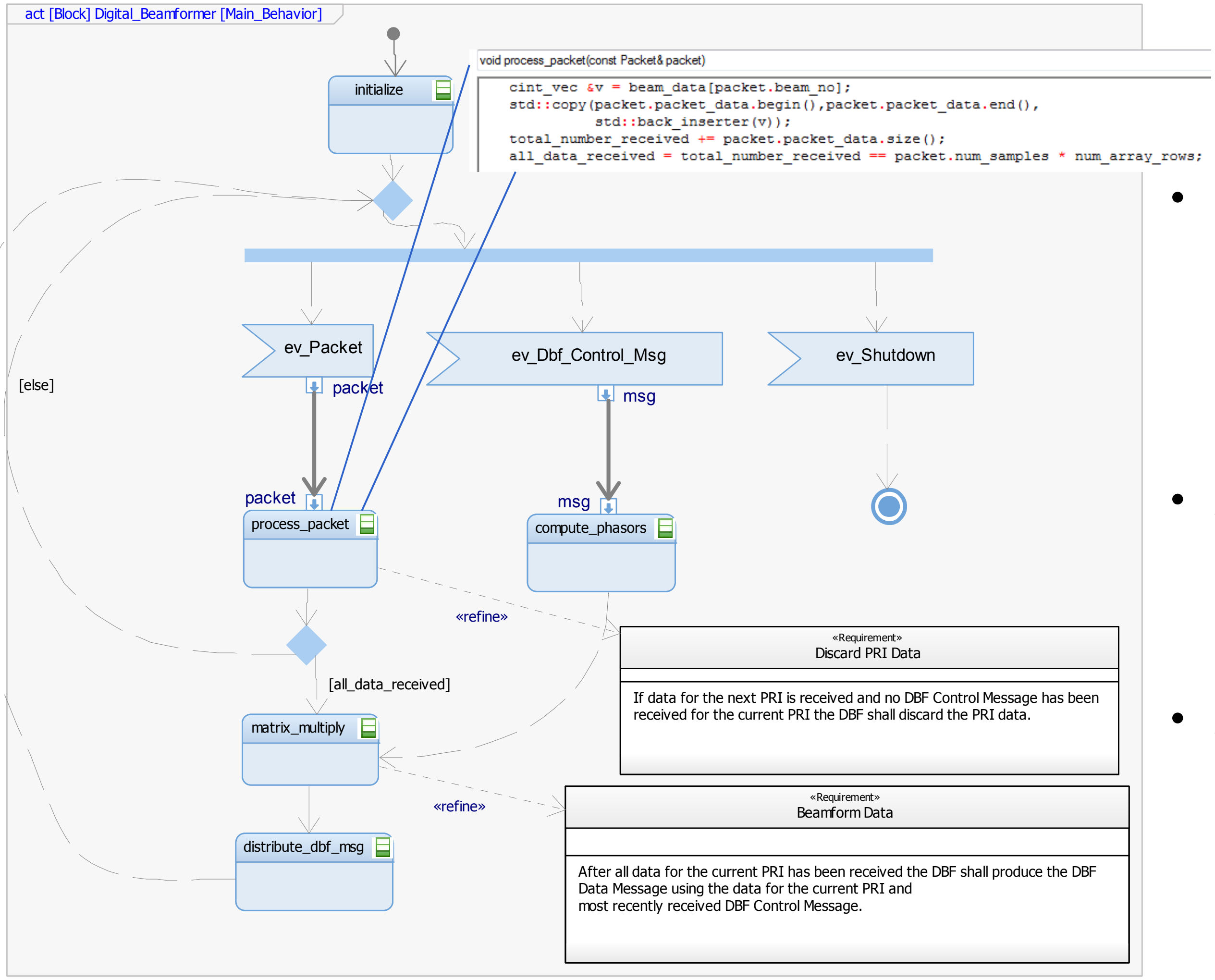
Digital Beamformer Interfaces



- IBD showing the interface messages that will be involved in the execution
- Tactical software messages
  - Defined in the Model
  - Tools auto-convert from language agnostic format to specific format
- Interface Documentation Auto-Generated from the Messages in the Model

Interfaces are Central to the Development of Executable Models

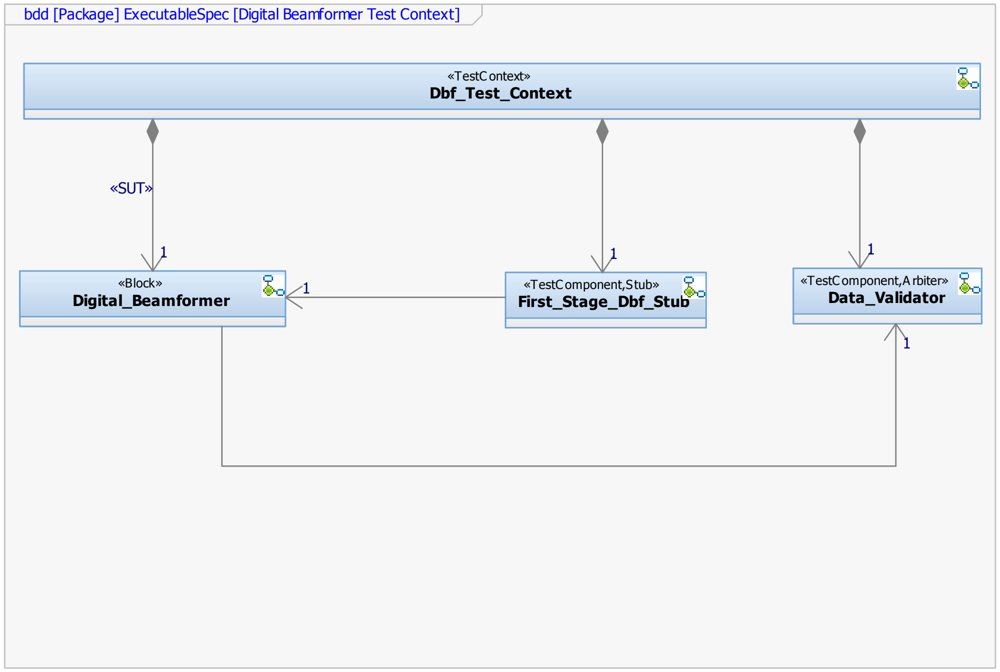
DBF Main Behavior



- Main Behavior Executes when the DBF is Instantiated
  - Not all Classifiers have a Main Behavior
  - Main Behavior may be a State Machine
- Actions may be opaque or have Activities as their Method
  - Opaque Action Written in the Action Language
- Actions Refine Requirements by Adding Executable Statements in the Action Language

Main Behavior Defines the Overall Control and Transformation of Inputs to Outputs

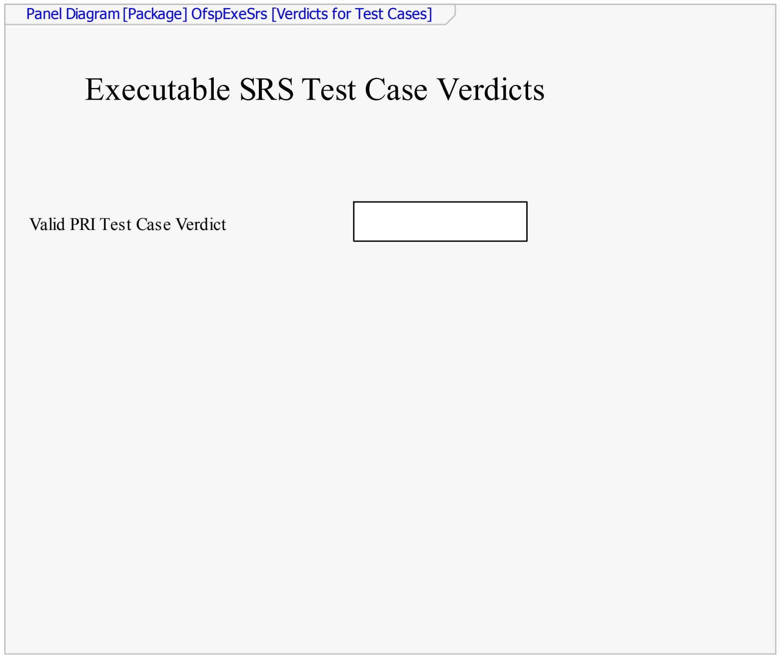
Apply UML UTP for Testing



- The DBF Test Context Controls the Tests
- DBF Defined as the System Under Test (SUT)
- Test Components Conform to the DBF Interfaces to:
  - Stimulate the DBF
  - Validate DBF Output & Report Test Success/Failure

Test Context Defines the Scope for What is Tested

Displaying Test Case Results

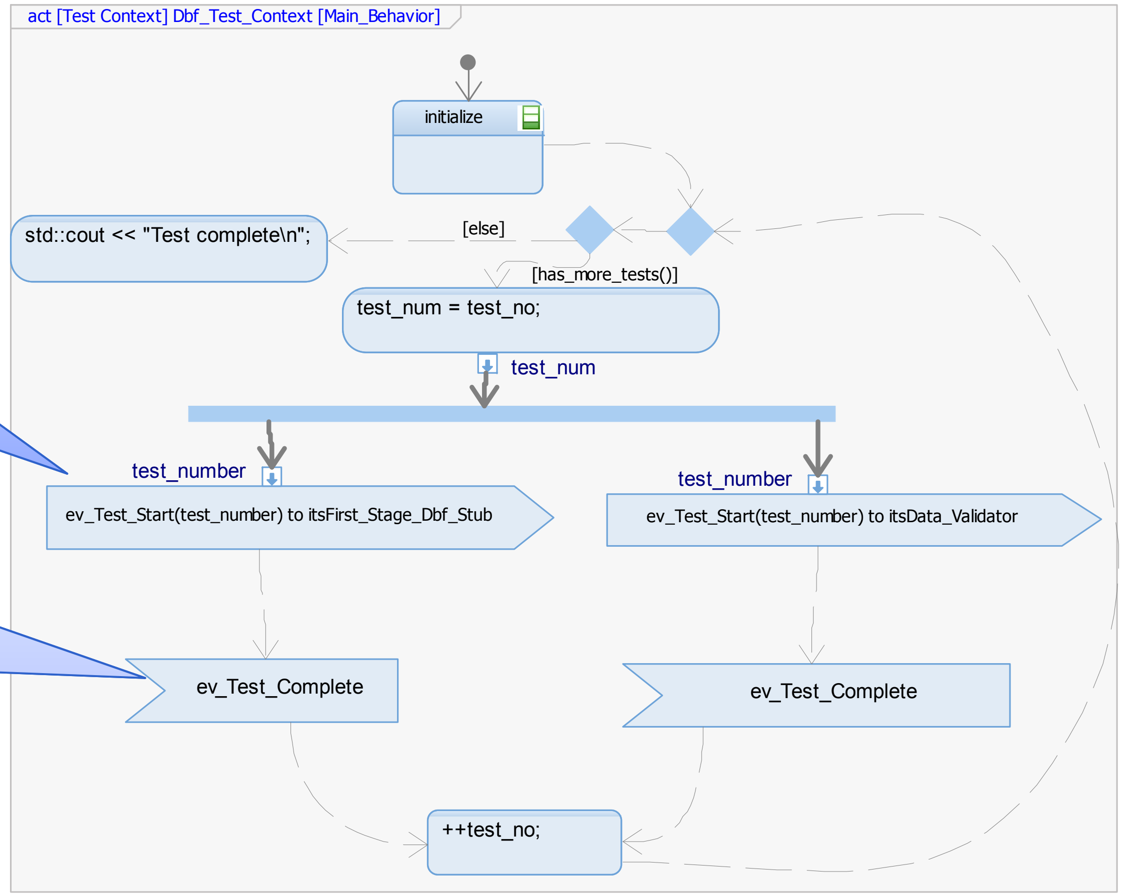


- Text Box is Linked to the Test Result Attribute of the Data Validator to display Pass/Fail Result
  - When the Test is Complete the Text Box Contains the Latest Value
- Panel Diagrams can Have Buttons, Gauges and Knobs
  - Allows the Modeler to Tune Values and Get Feedback

Panel Diagrams Provide Real-Time Feedback of Test Case Results

DBF Test Context Main Behavior

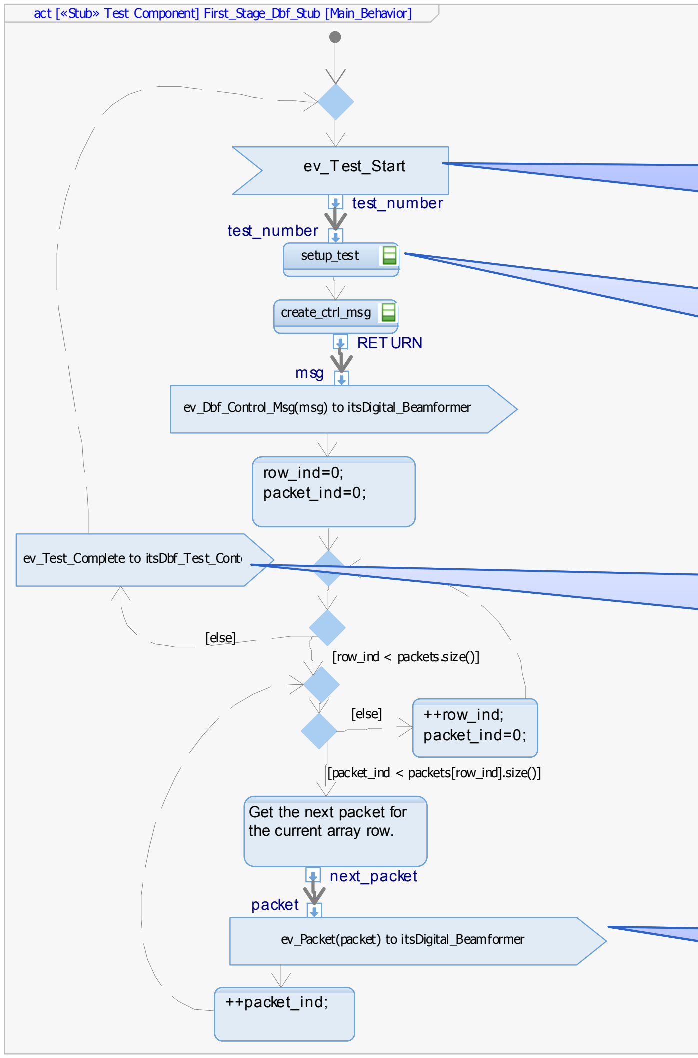
Test Components are Notified what Data To Load Based on Test Start Events.



Test Components Notify The Test Context when They Have Completed Their Test.

The Test Context Controls and Paces the Execution

First Stage DBF Stub Main Behavior



Main Behavior Waits for a Test Start Notification from the DBF Test Context.

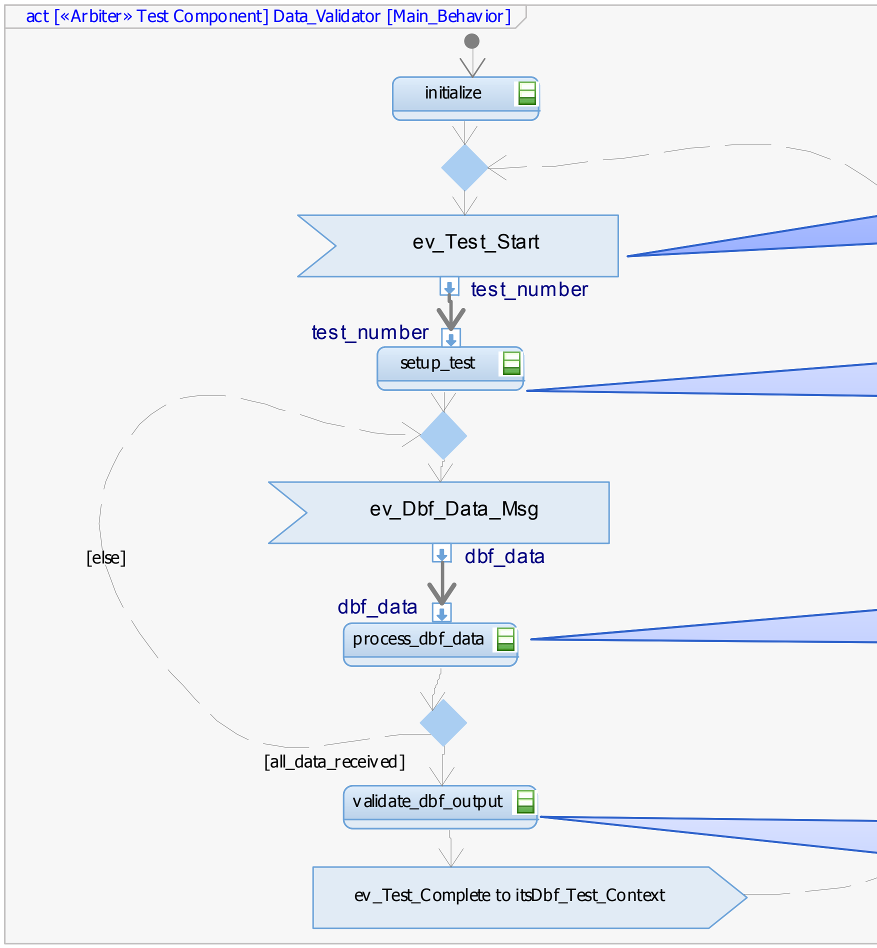
Stub Data is Generated Based on the Test Number that Arrives with the Test Start Notification.

When a Test is Complete a Signal is Sent back to the Test Context.

Signals are Sent to the DBF Based on the Stub Data.

Data Generator Simulates the Input Interface to the DBF

Data Validator Stub Main Behavior



Main Behavior Waits for a Test Start Notification from the DBF Test Context.

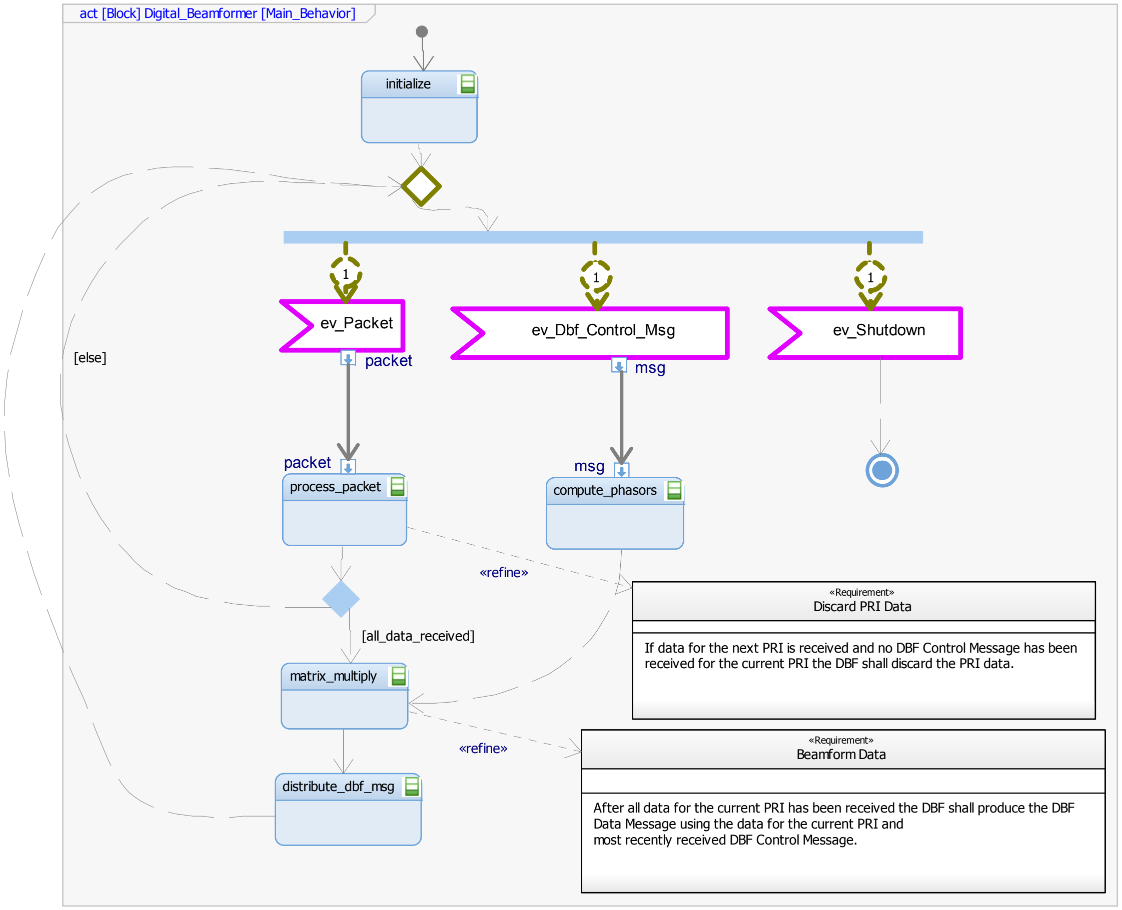
Load Appropriate Truth Data Based on Test Start Notification

Collect All DBF Data Messages (Based on Test Notification)

Validate DBF Output (Test Verdict Tied to Panel Diagram)

Data Validator Simulates the Output Interface of the DBF

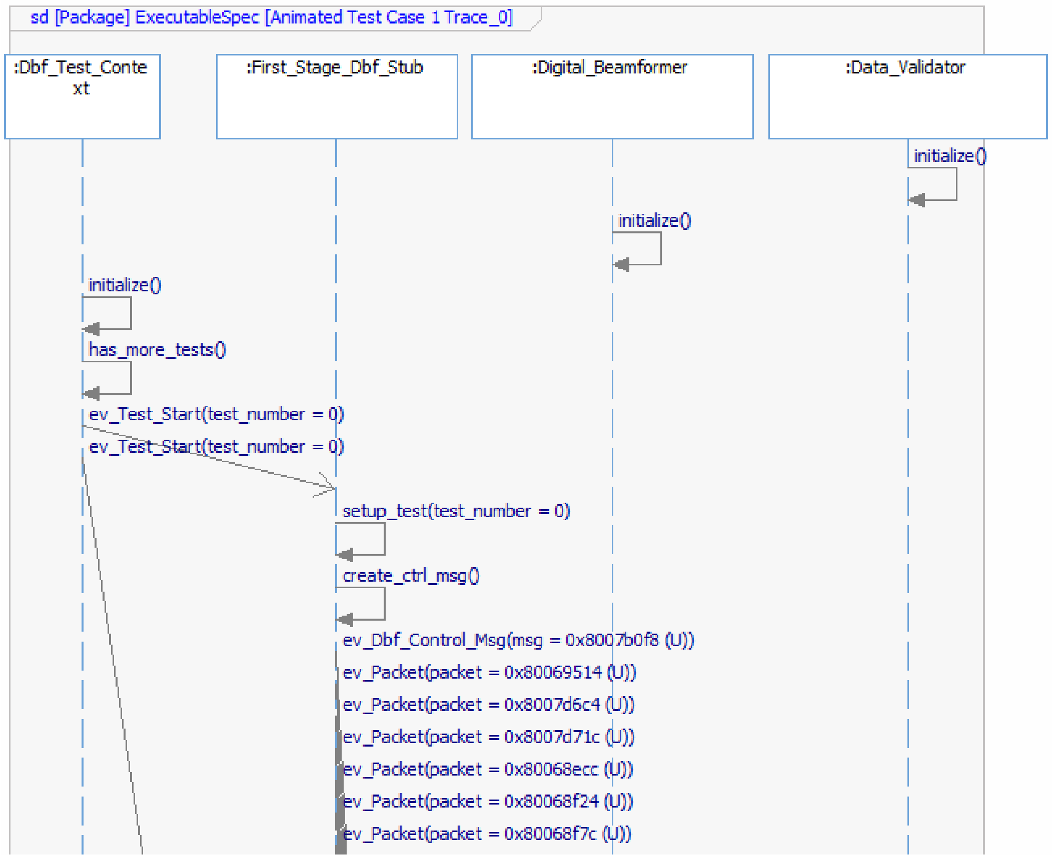
Animated Execution



- Animation Provides Visualization of the Execution
- Can be Shown to Stakeholders to Aid in Understanding the Requirements
- Useful for Debugging
- Supported by Multiple Tools
  - Rhapsody
  - MagicDraw
  - Enterprise Architect

Animation is an Enabler of Model Validation

Auto-Generation of Sequence Diagrams



- Sequence Diagrams are Auto-Generated by the Tool
  - Partial Trace Shown
- A Trace of What Actually Happened
- Can be Used as a Contract to Test Engineers
  - Identifies How the Specification was Tested at an Abstract Level
  - Assists in Test Case Creation

Auto-Generation of Sequence Diagrams Provide Full Execution Trace

Benefits of an Executable Specifications

- ✓Early Identification of Requirements/Architecture Defects
- ✓Early Identification of Error Cases
- ✓Animation can be Shown to the Customer
- ✓Greater Precision in Requirements
- ✓Utilization of Standards
- ✓Identification of “choke points”
- ✓Integrated Directly with the Model
- ✓Less Ambiguity
- ✓Independent of Design
- ✓Promote Model Reusability

Executable Specifications are an Affordability Enabler

Considerations

- Team must decide what part of the specification model should be executed
  - Logical architecture?
  - Physical architecture?
  - SoS, system, subsystem, component?
- Model Maintenance
- Execution language supported by tool may not be portable
  - ALF should help alleviate
- fUML standard defines a subset of UML/SysML for execution purposes
  - Tools may extend this subset
  - Tools may restrict this subset