Requirement-driven Model-based Testing of the cFS' Software Bus Service

Dharma Ganesan, Mikael Lindvall, Asa Valdimarsdottir, Stefan Hafsteinsson

In collaboration with the cFS team:
Susanne Strege, Walter Moleski, and Dave McComas, and Alan Cudmore

Agenda

- Context
- Motivation
  - System Under Test (SUT)
  - Challenges with traditional testing
- Model-based testing (MBT)
- Model-based testing applied on Software Bus
- Sample Issues detected
- Conclusion
- Possible future work

Context

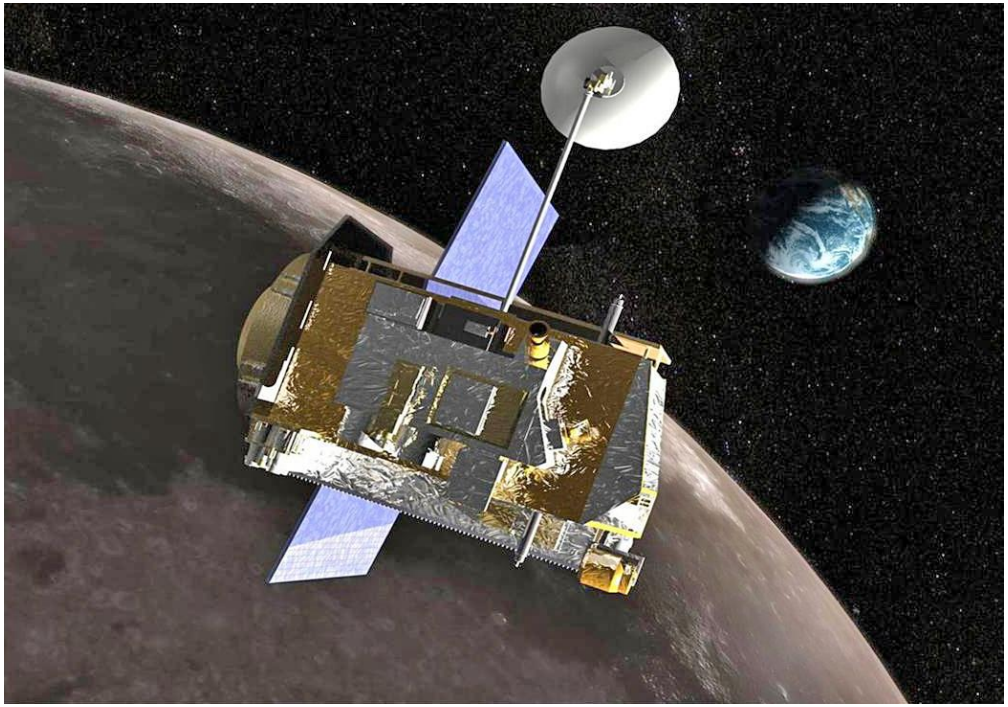
- The SARP project
  - Software Assurance Research Program

"An Assurance Approach to Identify, Test and Cover Off-nominal Behavior"

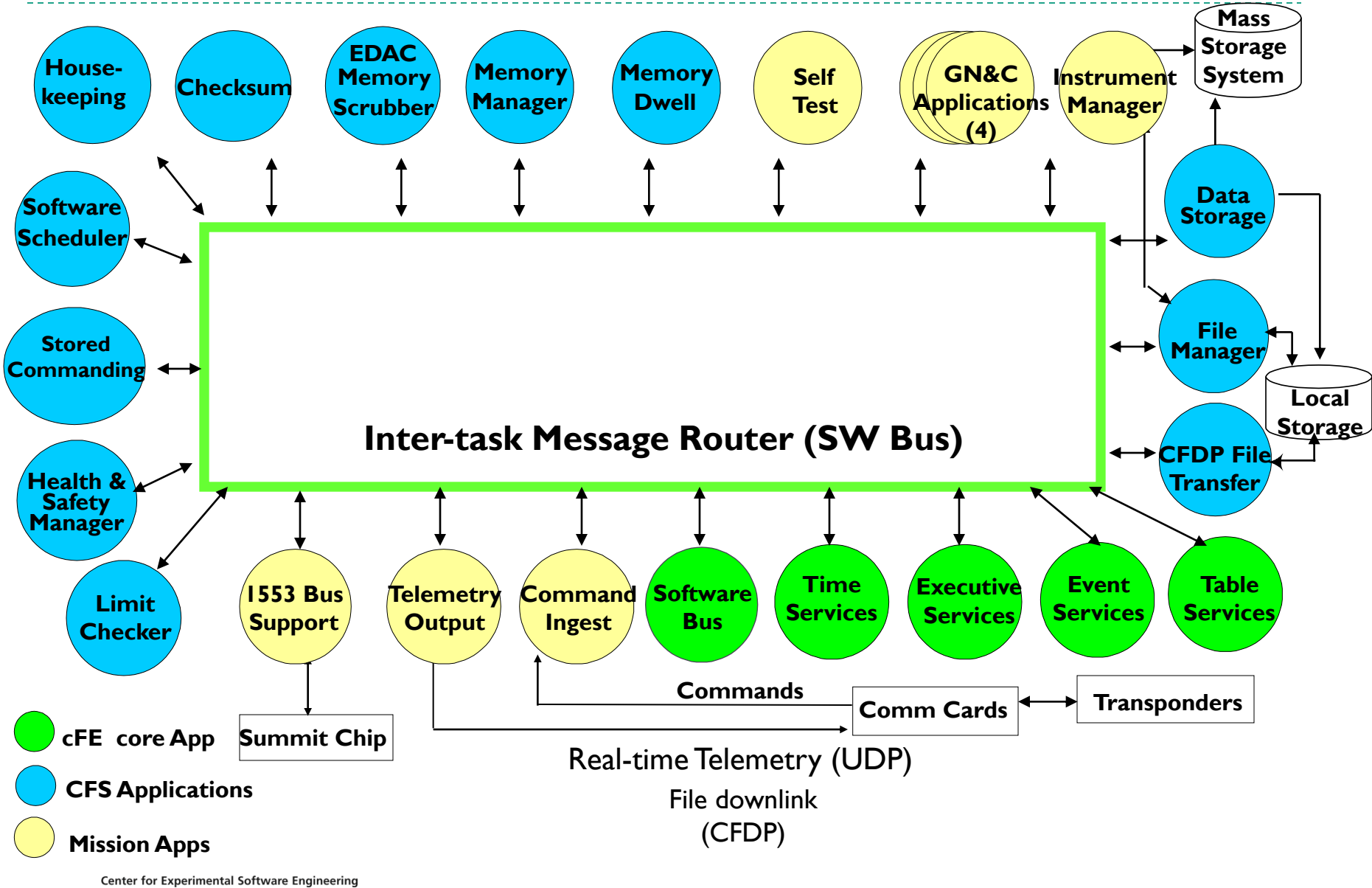
- Off-nominal behavior
  - Unexpected sequences of events
  - Out-of-range data values
  - Environment issues

Motivation

- Testing is always important
- Even more important when
  - Safety is critical
  - Failures are expensive



cFE/cFS Context Diagram



Challenges with traditional testing

- Why do we need more testing?

- Previous testing
  - Unit tests
    - Good coverage

  - Only test execution is automated
  - Test cases are manually constructed
  - Tests are too low-level detailed
  - Not testing multi-tasking nor communication between tasks

Solution

- Model-based Testing (MBT)
  - Relatively new technology
  - Black box approach
  - A model that describes desired behavior of the SUT
  - Automatic generation of innumerable test cases
    - Triggers off-nominal behaviors

Model-based Testing (MBT)

- The tester develops a model
  - Instead of writing test cases
  - Based on functional requirements and API documentation
  - In our case: A model program
    - Rules describe the SUT´s desired behaviors
  - Becomes the *"test oracle"*

MBT applied on the cFS‘ Software Bus

- Software Bus Service
  - Handles communication between tasks/applications
  - Publish-Subscribe architectural style
  - Main features
    - Pipes
      - Create, Delete, Subscribe through, Receive from
    - Messages
      - Send, Receive
    - Subscriptions
      - Subscribe, Unsubscribe

Challenges of testing a SB

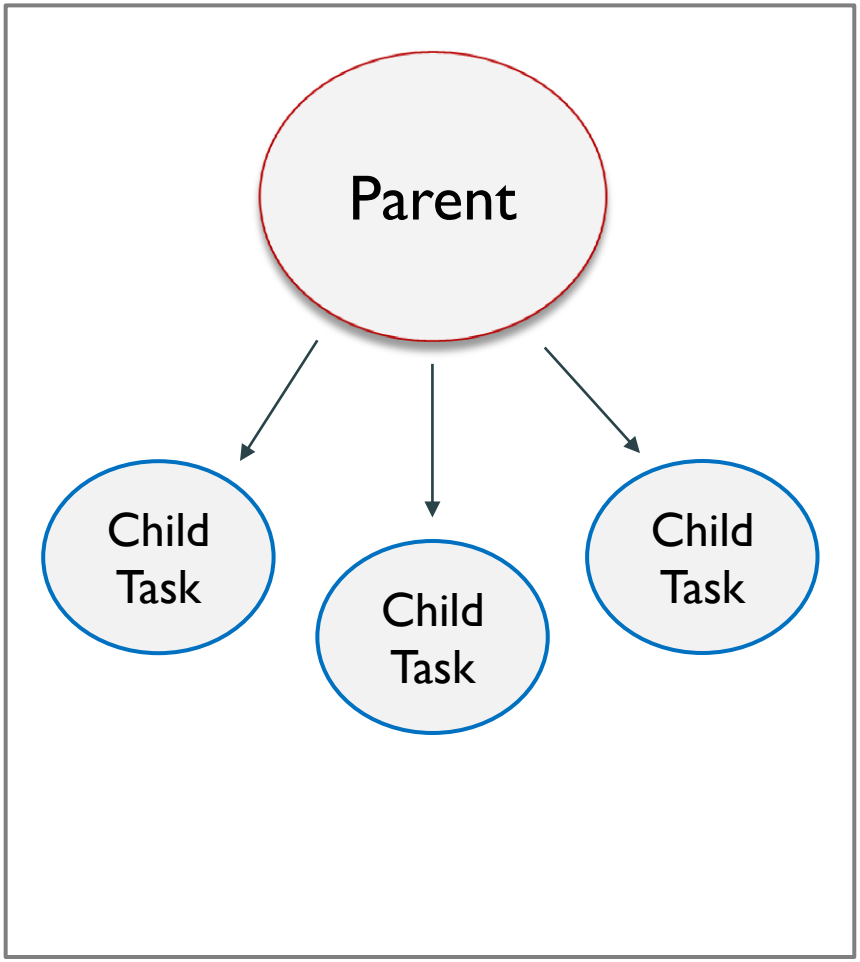
- Multi-tasking
- Communication between tasks/applications

- Need more than one task/application
- An application can not decide on the correctness of it‘s own execution

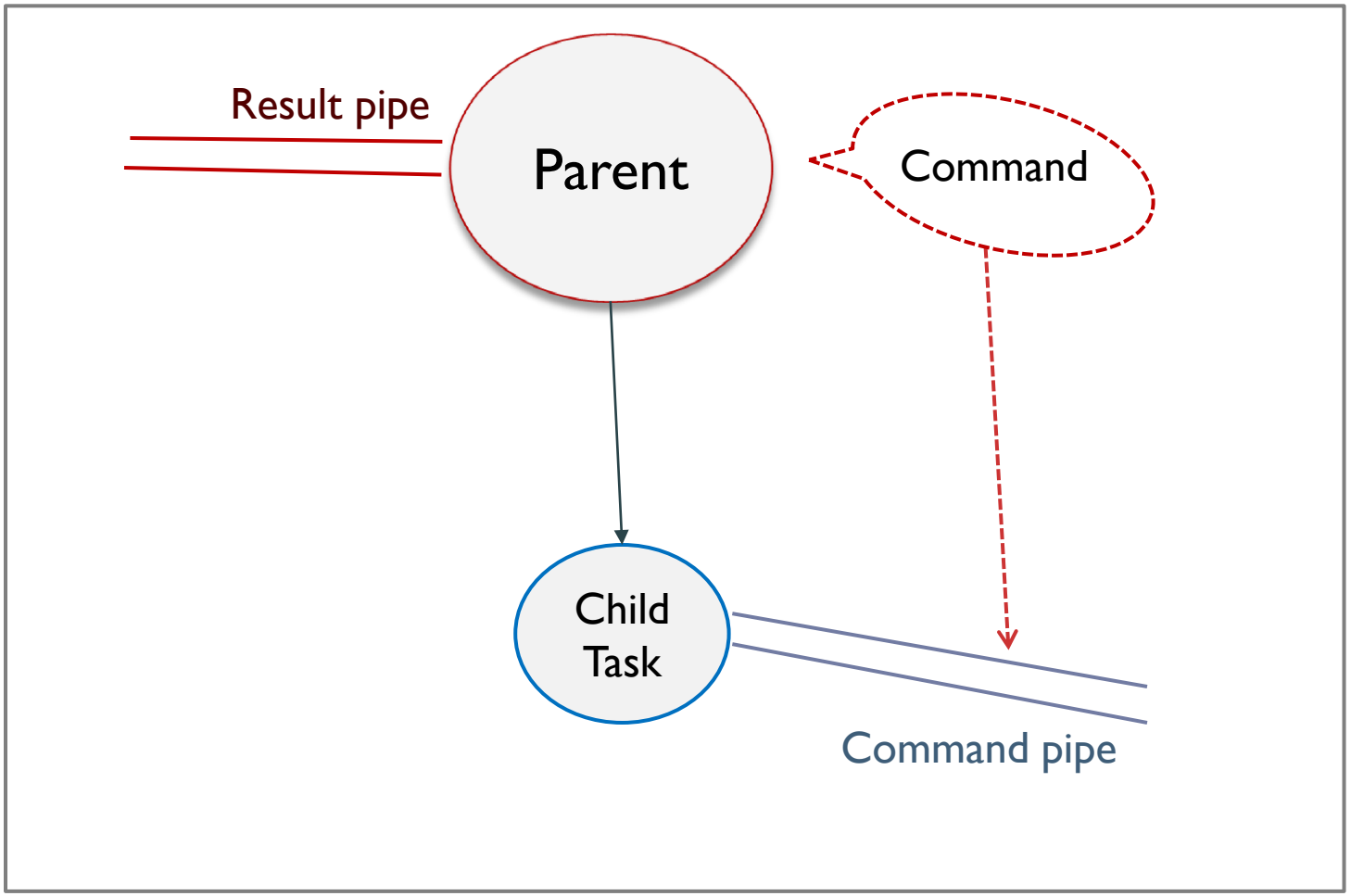
- Need an architecture for coordination
  - Overview
  - Control

Solution – The Parent Child Architecture

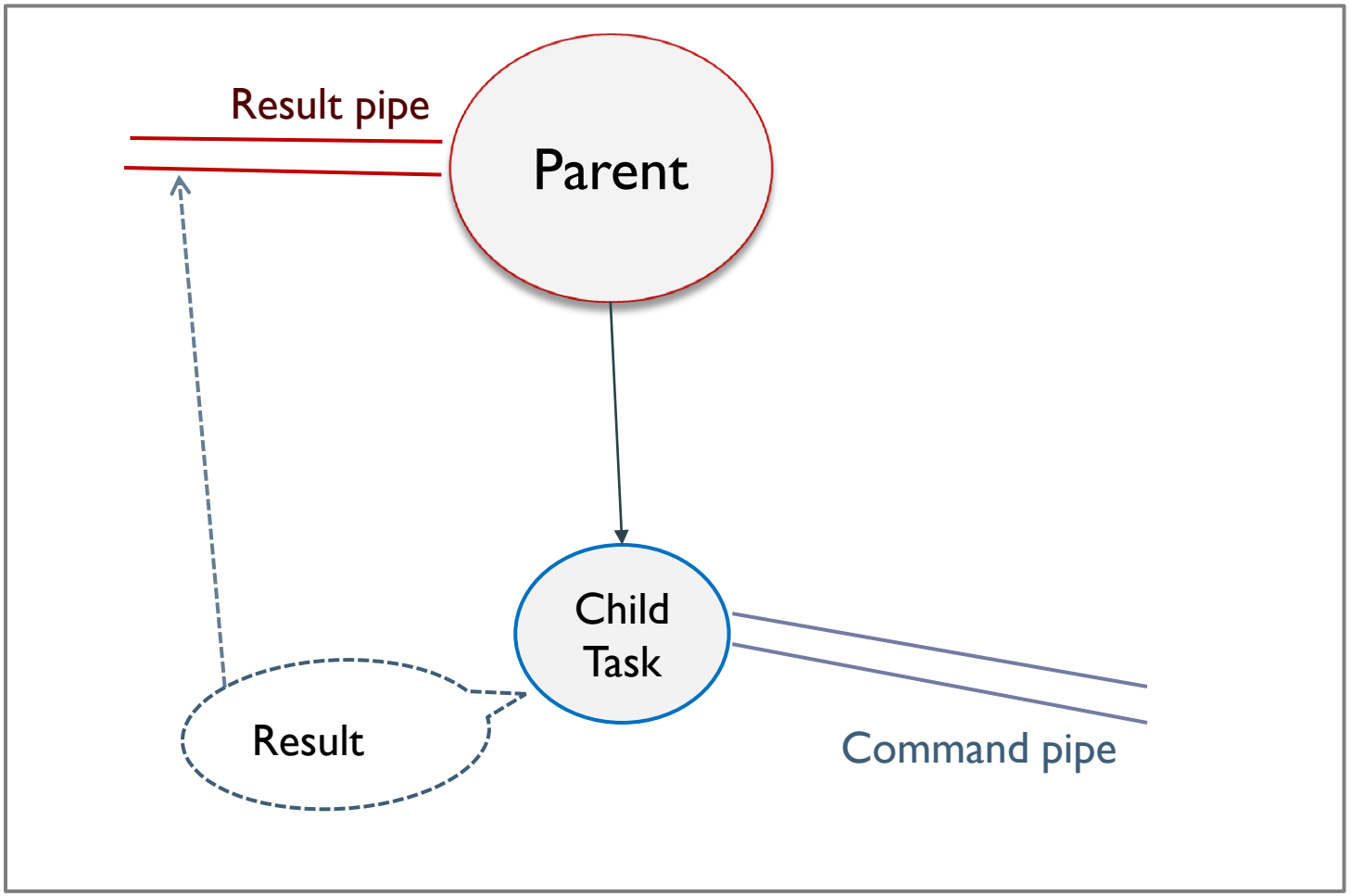
- Each test case acts as a ***Parent Application***
  - Generated by our model
- The Parent spans multiple ***Child Tasks***
  - The Parent sends commands to it's Child Tasks
  - The Parent "*knows*" which behaviors should succeed and which should fail



Solution – The Parent Child Architecture



Solution – The Parent Child Architecture



Example: Command sequence

- How to test whether publish works between two tasks?
- Our test case generates a sequence as below
- Parent send a sequence of commands:
  - Parent: "Child no. 1 create a pipe"
    - Child1creates a pipe and sends the return code to the parent
  - Parent: "Child no. 1subscribe to message nr 5"
    - Child 1 subscribes to message nr 5 and sends the return code
  - Parent:"Child 2 publish message nr 5"
    - Child 2 publishes message nr 5 and sends the return code
  - Parent:"Child 1receive message nr 5"
    - Child 1 receives message nr 5 and sends the return code

Our model

- Spec Explorer
  - A plug-in to Visual Studio
  - Tester develops a *model program*
  - Written in C#
  - **Simplified and abstract** version of the SUT
  - **Simple** structures and data types
    - Messages modeled as integers
    - Pipes modeled as a set of integers
  - No complex structures, threads, pointers or semaphores

Model Program (Sample)

- Model

```
[Rule]
public static void CreatePipe(int taskName, [Domain("PipeNameDomain")] int pipeName,
    [Domain("PipeDepthDomain")]int pipeDepth, bool result)
{
    bool guardStatus = CanCreatePipe(taskName, pipeName, pipeDepth);
    Condition.IsTrue(guardStatus == result);
    if (guardStatus)
    {
        AddPipe(taskName, pipeName, pipeDepth);
        Requirement.Capture("cSB4301");
    }
}
```

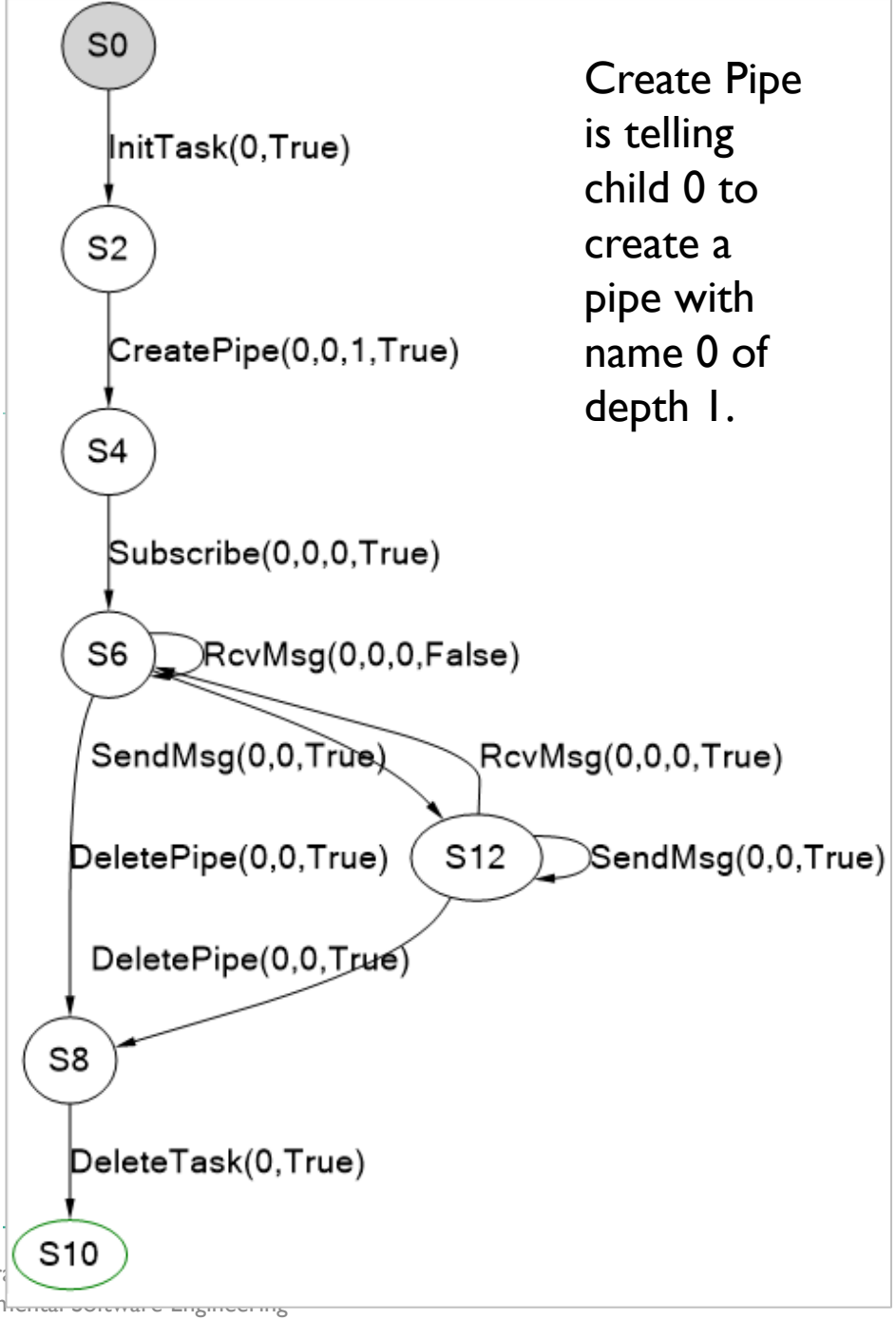
- We capture requirement numbers in our model
  - Helps in detecting missing requirements
  - Will be used for establishing traceability links

Our model

```
#region Rule Methods
[Rule]
public static void InitTask([Domain("TaskNameDomain")] int taskName, bool result)...

[Rule]
public static void DeleteTask([Domain("TaskNameDomain")] int taskName, bool result)...

[Rule]
public static void CreatePipe(int taskName, [Domain("PipeNameDomain")] int pipeName,
    [Domain("PipeDepthDomain")]int pipeDepth, bool result)...

[Rule]
public static void DeletePipe(int taskName, [Domain("PipeNameDomain")]  int pipeName, bool result)...

[Rule]
public static void Subscribe([Domain("TaskNameDomain")]int taskName, [Domain("MsgIdDomain")]int msgId,
    [Domain("PipeNameDomain")]int pipeName, bool result)...

[Rule]
public static void UnSubscribe([Domain("TaskNameDomain")]int taskName, [Domain("MsgIdDomain")]int msgId,
    [Domain("PipeNameDomain")]int pipeName, bool result)...

[Rule]
public static void SendMsg(int taskName, [Domain("MsgIdDomain")] int msgId, bool result)...

[Rule]
public static void RcvMsg(int taskName, [Domain("PipeNameDomain")] int pipeName,
    [Domain("MsgIdDomain")]int msgResult, bool result)...

#endregion
```

Generated from our model program

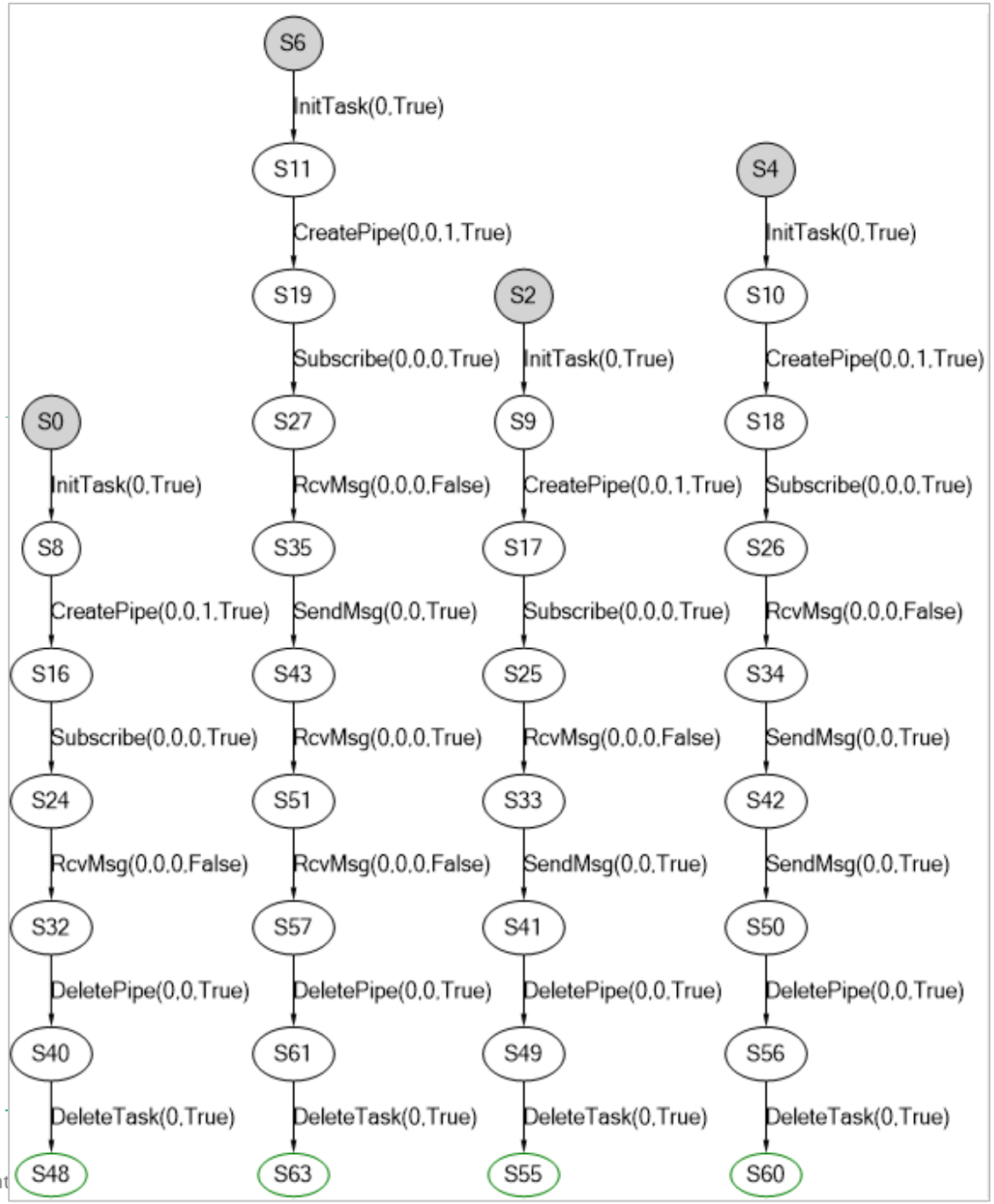
- Spec Explorer generates state machines from our model
- In regular MBT models are either manually created and/or derived from test cases (Fraunhofers tool FAST)



Create Pipe is telling child 0 to create a pipe with name 0 of depth 1.

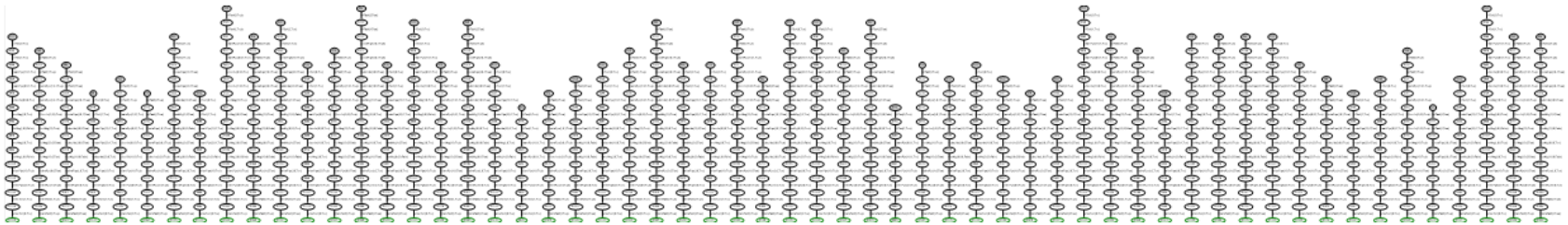
Generated test sequences - sample

- Each chain is a test case
  - A Parent Application
  - Parent creates a child (id is 0 in InitTask)
- Also get test code
  - Adapter converts the test code to SUT syntax
    - From C# to C

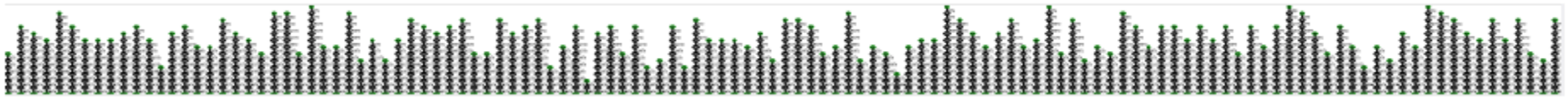


Generated test sequences - sample

- Two Child Tasks

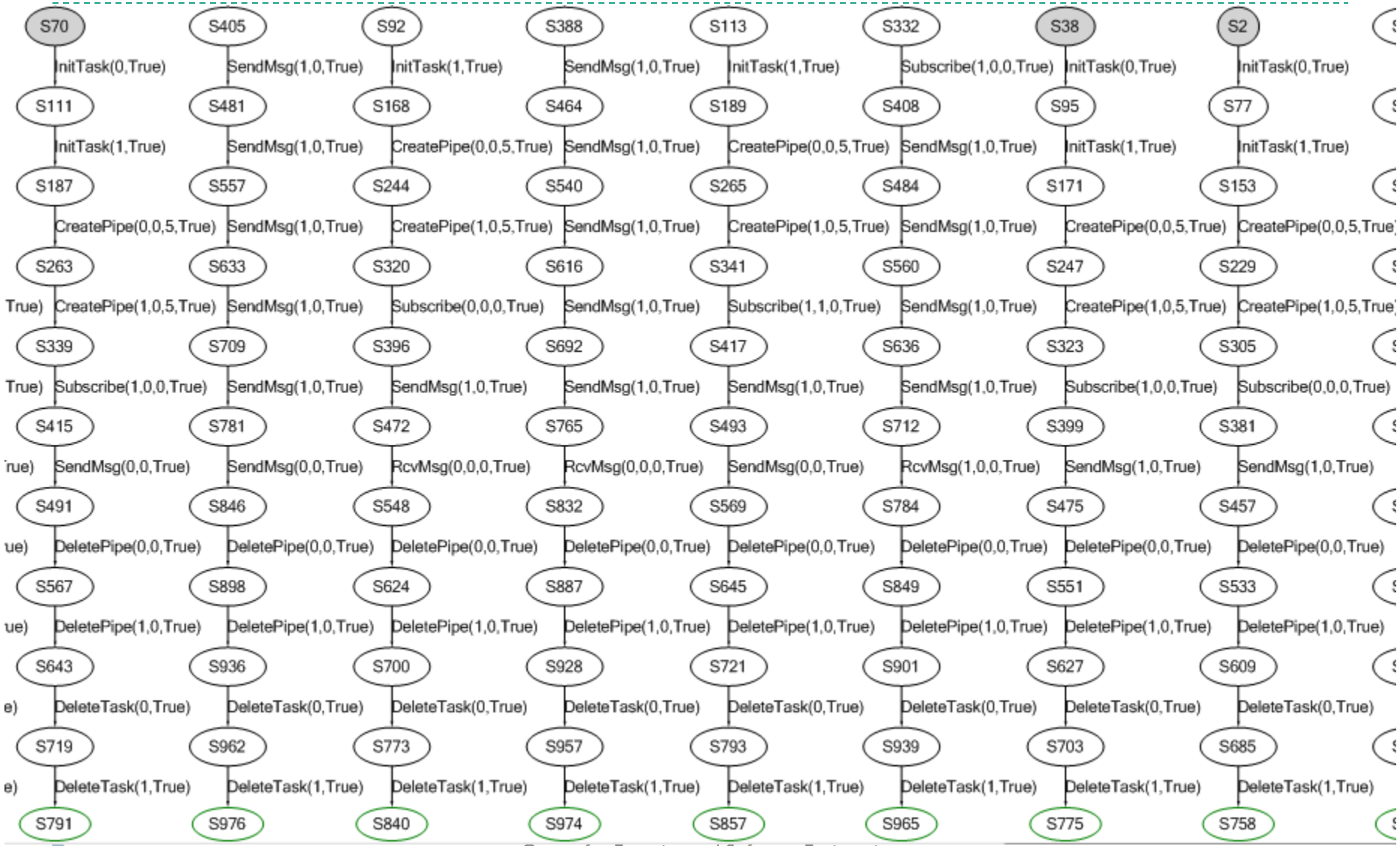


- Two Child Tasks and looser restrictions



Each test case (i.e. a parent) creates two child task and send different commands such as Publish, subscribe, unsubscribe, ReadMsg, etc.

Zoom-in


S70 S405 S92 S388 S113 S332 S38 S2
InitTask(0,True) SendMsg(1,0,True) InitTask(1,True) SendMsg(1,0,True) InitTask(1,True) Subscribe(1,0,0,True) InitTask(0,True) InitTask(0,True)
S111 S481 S168 S464 S189 S408 S95 S77
InitTask(1,True) SendMsg(1,0,True) CreatePipe(0,0,5,True) SendMsg(1,0,True) CreatePipe(0,0,5,True) SendMsg(1,0,True) InitTask(1,True) InitTask(1,True)
S187 S557 S244 S540 S265 S484 S171 S153
CreatePipe(0,0,5,True) SendMsg(1,0,True) CreatePipe(1,0,5,True) SendMsg(1,0,True) CreatePipe(1,0,5,True) SendMsg(1,0,True) CreatePipe(0,0,5,True) CreatePipe(0,0,5,True)
S263 S633 S320 S616 S341 S560 S247 S229
True) CreatePipe(1,0,5,True) SendMsg(1,0,True) Subscribe(0,0,0,True) SendMsg(1,0,True) Subscribe(1,1,0,True) SendMsg(1,0,True) CreatePipe(1,0,5,True) CreatePipe(1,0,5,True)
S339 S709 S396 S692 S417 S636 S323 S305
True) Subscribe(1,0,0,True) SendMsg(1,0,True) SendMsg(1,0,True) SendMsg(1,0,True) SendMsg(1,0,True) SendMsg(1,0,True) Subscribe(1,0,0,True) Subscribe(0,0,0,True)
S415 S781 S472 S765 S493 S712 S399 S381
rue) SendMsg(0,0,True) SendMsg(0,0,True) RcvMsg(0,0,0,True) RcvMsg(0,0,0,True) SendMsg(0,0,True) RcvMsg(1,0,0,True) SendMsg(1,0,True) SendMsg(1,0,True)
S491 S846 S548 S832 S569 S784 S475 S457
ue) DeletePipe(0,0,True) DeletePipe(0,0,True) DeletePipe(0,0,True) DeletePipe(0,0,True) DeletePipe(0,0,True) DeletePipe(0,0,True) DeletePipe(0,0,True) DeletePipe(0,0,True)
S567 S898 S624 S887 S645 S849 S551 S533
ue) DeletePipe(1,0,True) DeletePipe(1,0,True) DeletePipe(1,0,True) DeletePipe(1,0,True) DeletePipe(1,0,True) DeletePipe(1,0,True) DeletePipe(1,0,True) DeletePipe(1,0,True)
S643 S936 S700 S928 S721 S901 S627 S609
e) DeleteTask(0,True) DeleteTask(0,True) DeleteTask(0,True) DeleteTask(0,True) DeleteTask(0,True) DeleteTask(0,True) DeleteTask(0,True) DeleteTask(0,True)
S719 S962 S773 S957 S793 S939 S703 S685
e) DeleteTask(1,True) DeleteTask(1,True) DeleteTask(1,True) DeleteTask(1,True) DeleteTask(1,True) DeleteTask(1,True) DeleteTask(1,True) DeleteTask(1,True)
S791 S976 S840 S974 S857 S965 S775 S758

Requirement tracability

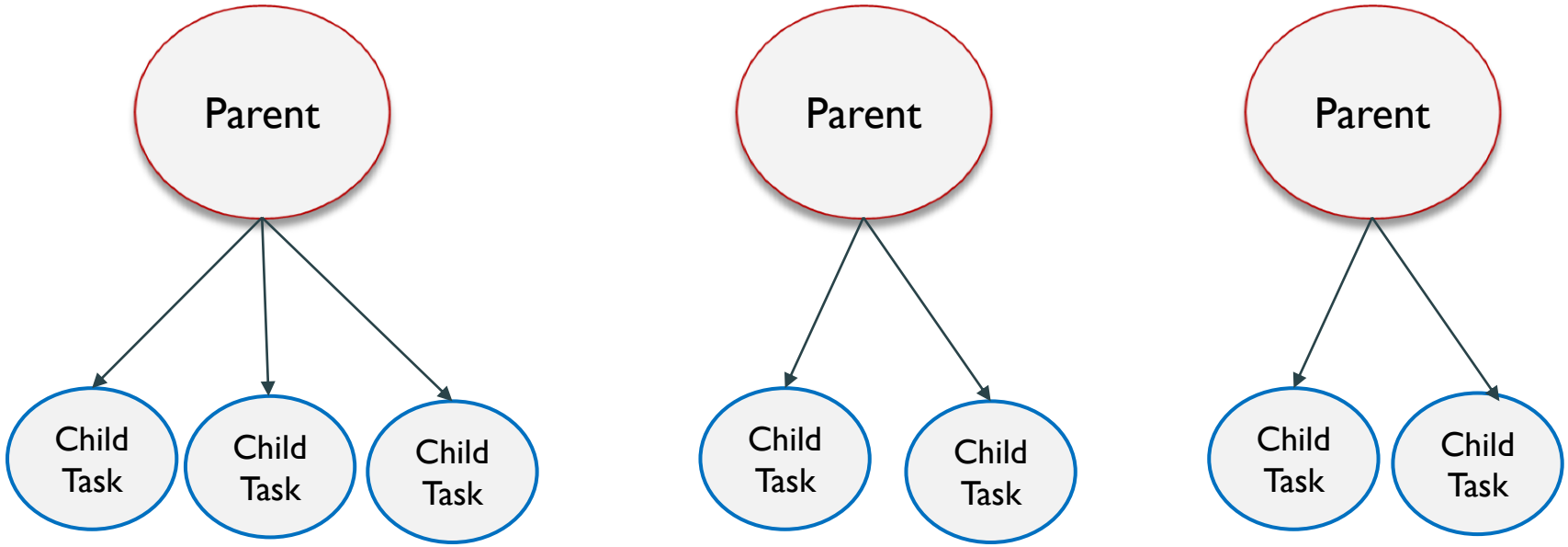
```
-Entering InitTask_w, TaskName:0
init task...
-Exiting InitTask_w, TaskId:8, Status is:0
Requirement checked: cES1311

-Entering CreatePipe_w, TaskName:0, PipeName:0, PipeDepth:0
create pipe...
EVS Port1 66/1/CFE_SB 2: CreatePipeErr:Bad Input Arg:app=Test_CFE_SB.0,ptr=0xb741c267,depth=0,maxdepth=256
-Exiting CreatePipe_w, Status:ca000003

-Entering CreatePipe_w, TaskName:0, PipeName:0, PipeDepth:-1
create pipe...
EVS Port1 66/1/CFE_SB 2: CreateP
-Exiting CreatePipe_w, Status:ca

-Entering DeletePipe_w, TaskName
delete pipe...
EVS Port1 66/1/CFE_SB 46: Pipe D
-Exiting DeletePipe_w, Status:ca

-Entering CreatePipe_w, TaskName
create pipe...
-Exiting CreatePipe_w, Status:0
Requirement checked: cSB4301
```

- Future work: Create visualization

File Edit Format View Help

```
TestCase Requirement
PubSubTestSuiteS0 cES1311
PubSubTestSuiteS0 cSB4305
PubSubTestSuiteS0 cSB4301
CreateDeleteTaskSuiteS0 cES1311.1
CreateDeleteTaskSuiteS0 cES1312
```

Cross running



Allows for testing of more complex concurrency situations

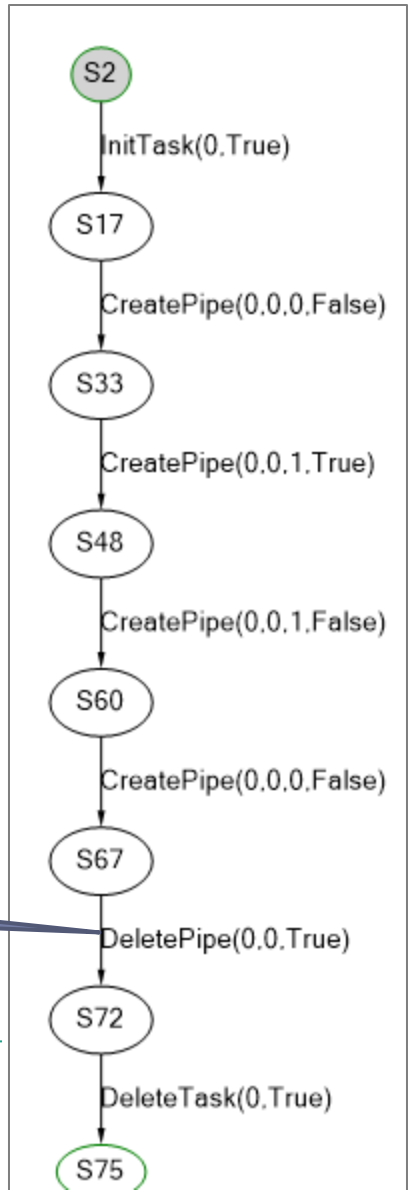
Issues detected

- These issues were unknown to the CFS team
- Off-nominal behaviors
  - Trying to create an already existing pipe caused data corruption
- Cross running multiple Parent Applications
  - Method called to delete Child Tasks does not clean up the task data properly
  - Caused memory issue
- Requirement issues
  - Missing or/and unclear requirements
  - Off-nominal scenarios often not discussed
  - Duplicate requirements

A sample issue detected

Another issue (sample)

- Off-nominal pipe creation
  - Found early
  - Trying to create an already existing pipe caused data corruption
  - Not accounted for in the SUT's code
  - Caused failures in later steps



Delete Pipe failed

Conclusion and future work

- MBT is promising
- Detecting off-nominal issues is the core of MBT
- MBT Helps in detecting
  - Missing requirements (we can improve requirements)
  - Functional errors in the SUT

Acknowledgements

- NASA SARP
  - Martha Wetherholt
  - Kenneth D. Rehm
  - Steven Hard
  - Markland Benson

- cFS team
  - Jonathan Wilmot

Contact:

- dganesan@fc-md.umd.edu
- mlindvall@fc-md.umd.edu