Automated Test Generation for Flight Software (FSW)

Dharmalingam Ganesan
Mikael Lindvall
Christoph Schulze
Gunnar Cortes

Fraunhofer Center Maryland

David McComas
Nicholas Yanchik
Alan Cudmore
Steve Slegel

Goddard Space Flight Center
Code 582, Flight Software
Systems Branch

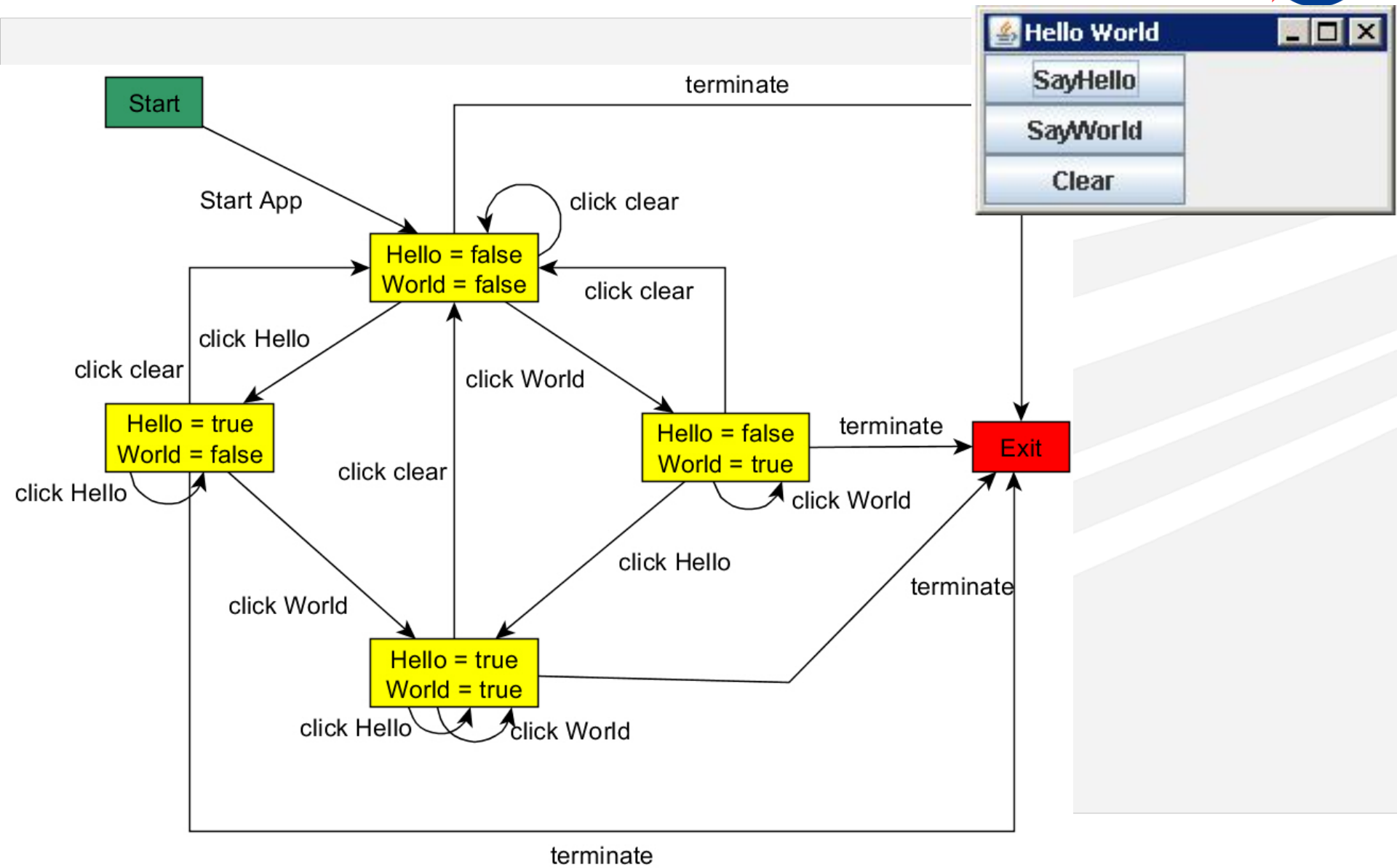
Problems with Current Testing

- Manual testing takes a lot of effort
- Manual testing could miss “corner-cases”
- Test cases difficult to maintain and reuse on multiple environments
- Tests are technical, typically not domain oriented
  - and thus difficult to understand for non-technical stakeholders

Model based testing (MBT)

- Solution: Generate Test Cases using MBT!
  - We incrementally model the behavior of the software to be tested
    - Good and “Evil” usages
    - Expected results are embedded in the model
  - Based on the model, the test generator automatically generates test cases
  - Any type of test cases can be generated
    - Junit, Xunit, soapUI, Selenium, CuTest, etc.
- Extra benefits
  - We have a very visual representation of the system
  - Test cases are domain oriented and easier to understand for non-technical stakeholders
  - Documentation of behavior

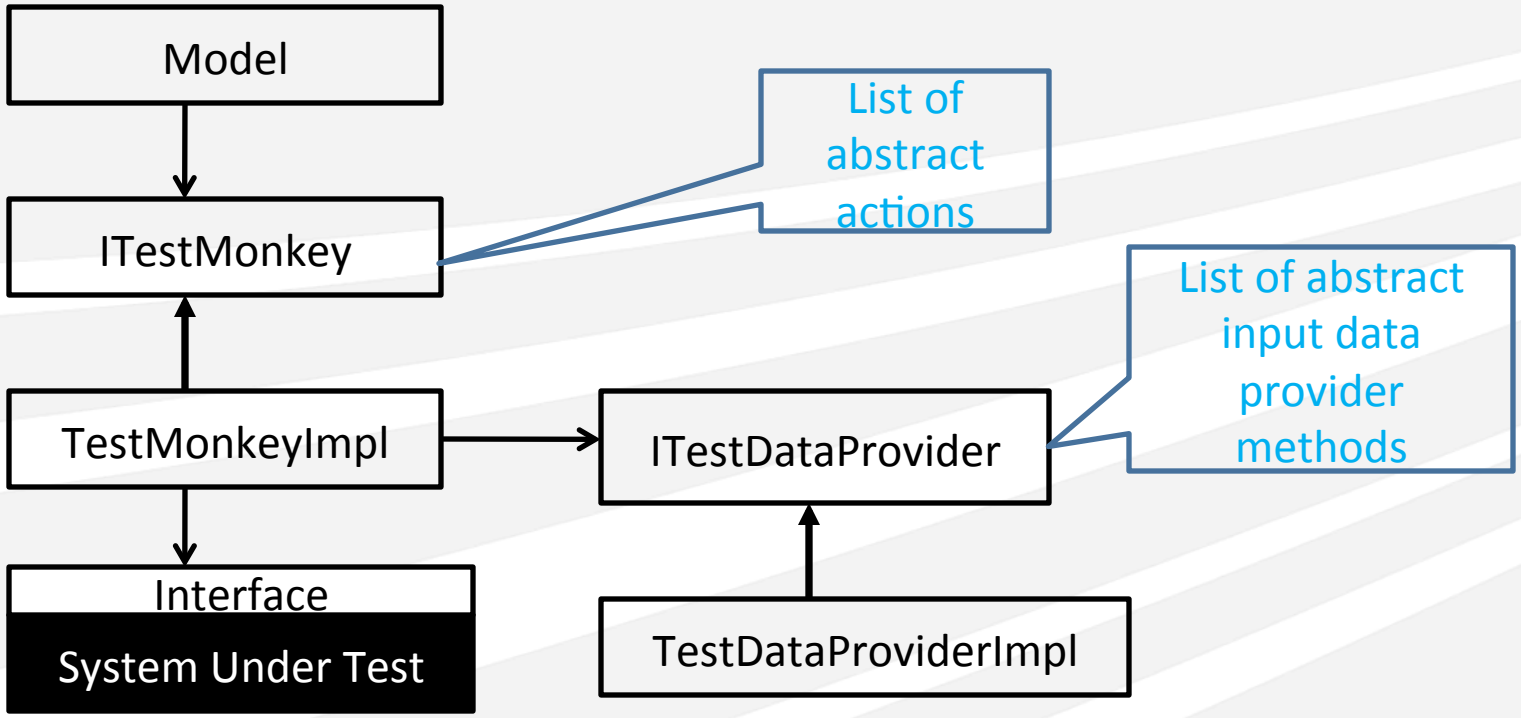
Hello World to MBT



Terminology

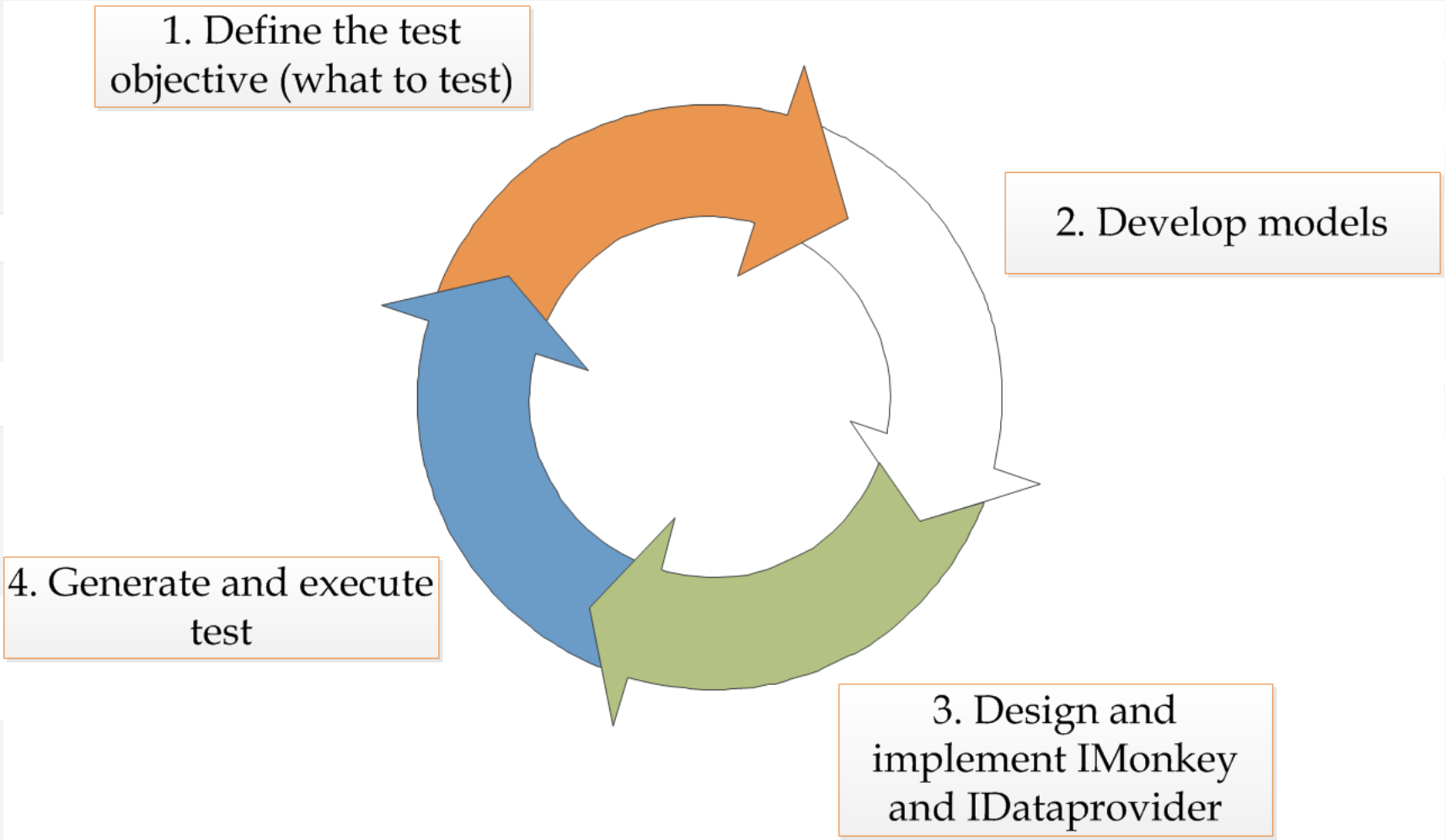
- Nodes of the model are called states
- Edges of the model are called transitions
- Each test case is a path from the start state to the exit state
- A random test is generated using a random walk from the start state to the exit state
- A test suite is a collection of test cases

Architecture of MBT



- Model is agnostic to the test execution technology
- ITestMonkey interface hides the test execution framework
- TestMonkeyImpl uses interfaces of the test execution framework

Workflow



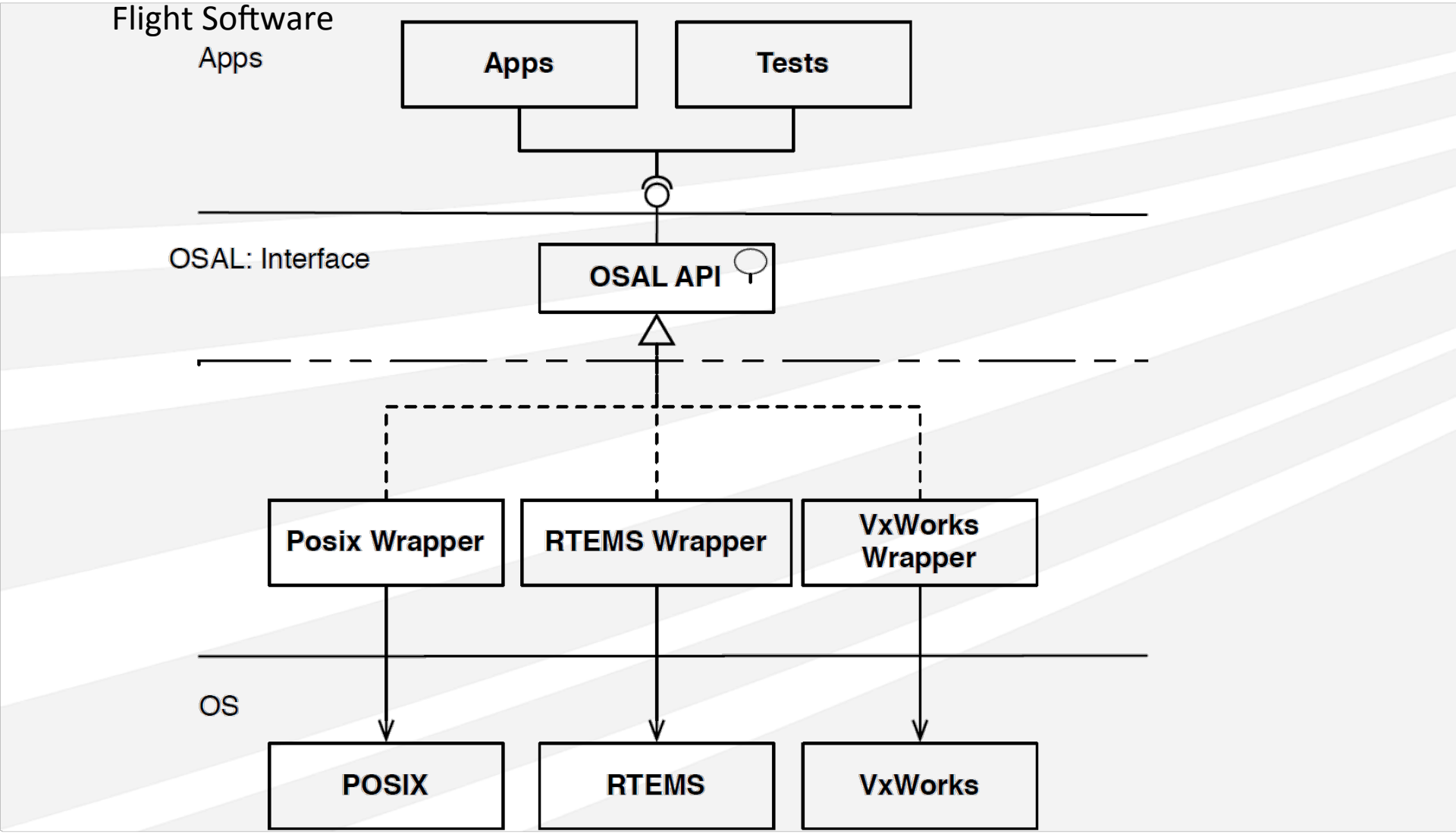
Flight Software Under Test

- **O**perating **S**ystem **A**bstraction **L**ayer (OSAL)
- Isolates flight software from operating systems and hardware
- Implementations for the real time systems RTEMS and VxWorks and posix compliant non-real time systems
- Used for mission critical embedded systems
- Provides support for file-system, tasks, queues, semaphores, interrupts, hardware abstraction, I/O ports and exception handling

NASA OSAL

- Why is it important that OSAL be bug free?
  - Flight software is mission critical and needs to be of very high quality
  - OSAL is the foundation of the core Flight Executive (cFE)
  - OSAL is used in many NASA missions, e.g. the Lunar Renaissance Orbit
  - If OSAL has issues, it might result in catastrophic failure

NASA OSAL – Architecture



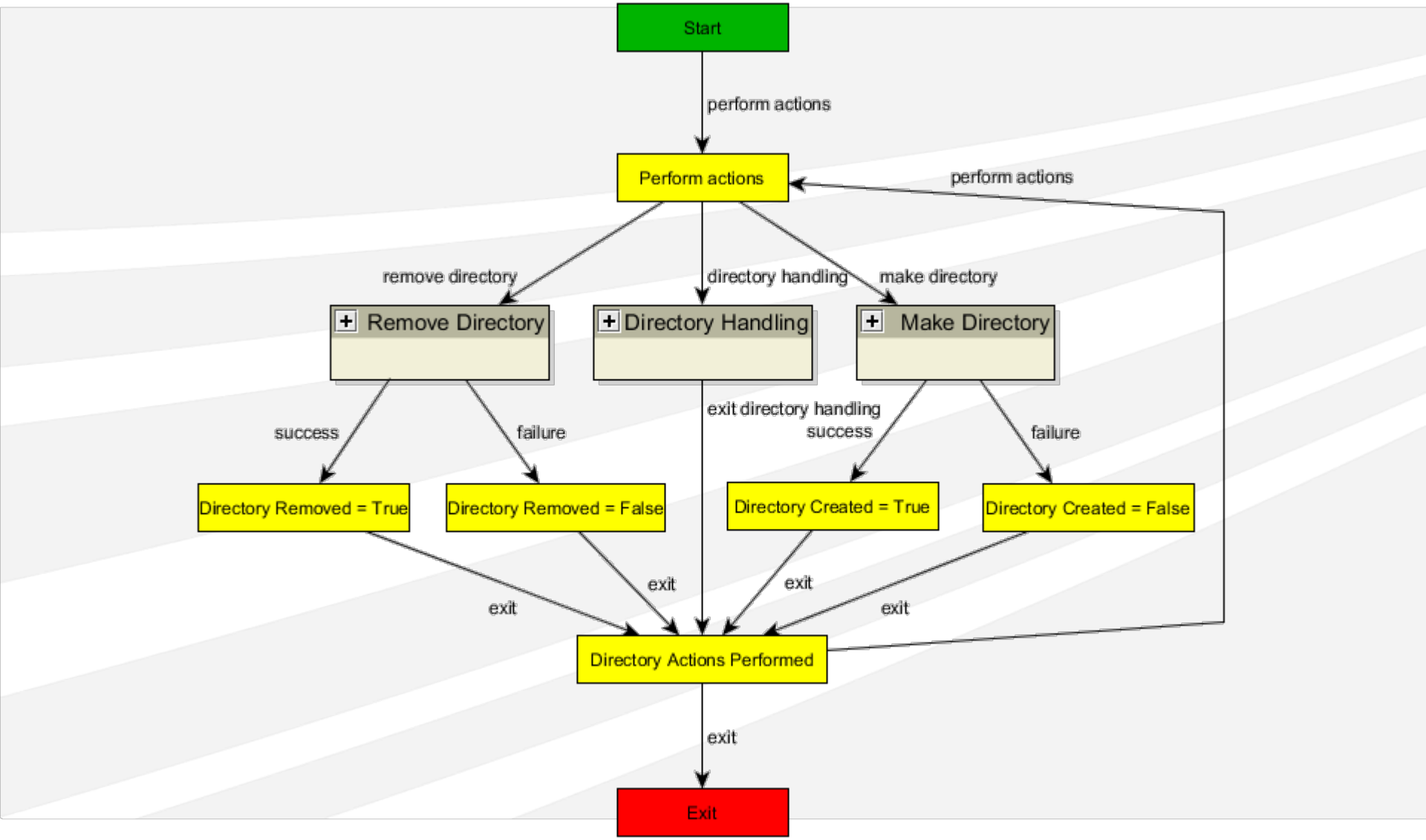
Applying MBT on OSAL

- Model captures OSAL's <u>expected behavior</u>
- Test monkey has abstract instructions e.g. "CreateFileValid", "CreateFileInvalid"
- We created and used a dynamic data provider
  - Generates file and folder names
  - Generates content for files

Applying MBT on OSAL ...

- OSAL‘s file system API‘s were modeled
- Several test cases in C were automatically generated
  - We used the open source CuTest test execution framework

Example of a OSAL model

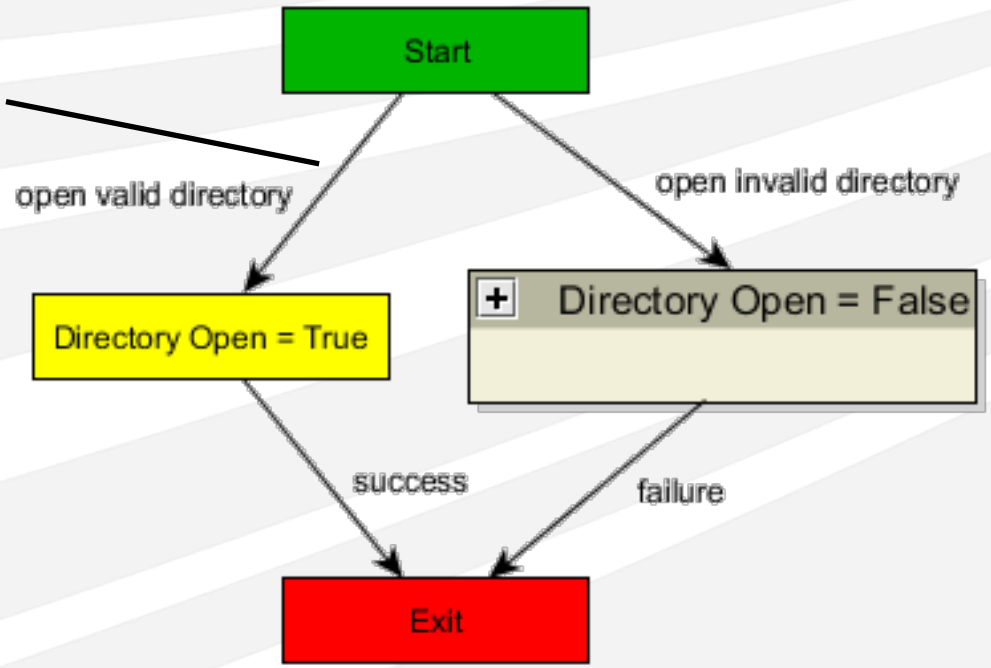


API spec of make directory

```
/*-----------------------------------------------------------------------------
   Name: OS_mkdir

   Purpose: makes a directory specified by path.

   Returns: OS_FS_ERR_INVALID_POINTER if path is NULL
            OS_FS_ERR_PATH_TOO_LONG if the path is too long to be stored locally
            OS_FS_ERR_PATH_INVALID if path cannot be parsed
            OS_FS_ERROR if the OS call fails
            OS_FS_SUCCESS if success

   Note: The access parameter is currently unused.
-----------------------------------------------------------------------------*/

int32 OS_mkdir (const char *path, uint32 access);
```

Sub-model of make directory

```
pointer = openDirectoryValid();
CuAssertTrue(tc,pointer != NULL);
```



Sample generated Test in CuTest

```
void Testosal_Filesystem_min_2(CuTest* tc)
 {
    status = makeFilesystemValid();
    CuAssertIntEquals_Msg(tc,"Filesystem could not be created",
                                    OS_FS_SUCCESS, status);

    status = mountFilesystemValid();
    CuAssertIntEquals_Msg(tc,"Filesystem could not be mounted",
                                    OS_FS_SUCCESS, status);

   pointer = openDirectoryValid();
   CuAssertTrue(tc, pointer != NULL);
   …
   status = removeFilesystemValid();
   CuAssertIntEquals_Msg(tc,"Filesystem could not be removed”, status);
}
```

Issues found using MBT

- OSAL is a high-quality product (only a few issues detected!)
- Issues found when running model based tests on the Posix implementation of OSAL:
- File-descriptors after removing file-system:
  - After somewhat long tests we would run out of file-descriptors
  - This would even happen with a newly created file-system
  - Cause: OSAL does not remove file-descriptors when the file-system is removed
- Wrong error codes returned and unimplemented features:

Test scenario	Error message	Expected	Actual
checkFileSystemNullName()	Expected 'invalid pointer' error	OS_FS_ERR_INVALID_POINTER	OS_FS_UNIMPLEMENTED
checkFileSystemOsCallFails()	Expected 'filesystem' error	OS_FS_ERROR	OS_FS_UNIMPLEMENTED
checkFilesystemValid()	Filesystem Not Checked	OS_FS_SUCCESS	OS_FS_UNIMPLEMENTED
copyFileLongSourceFilename()	Filesystem error code expected	OS_FS_ERROR	OS_FS_ERR_NAME_TOO_LONG
copyFileNonExistingSourceFile()	Filesystem error code expected	OS_FS_ERROR	OS_FS_SUCCESS
renameFileLongSourceFilename()	Filesystem error code expected	OS_FS_ERROR	OS_FS_ERR_NAME_TOO_LONG

MBT – some limitations

- Modeling requires specification of SUT
- Developers are typically not used to modeling
- Difficult to document individual test cases
  - Note: Models summarize all test cases
  - Some customers require document of each test case

Summary and Future Work

- MBT works for API-based testing!
- Some detected issues are serious (e.g., running out of file descriptors)
- MBT requires manual work for
  - constructing models
  - monkey and data provider implementations
  - maintenance of above artifacts
- However, test cases are automatically generated
- Future:
  - Evaluating it on requirements of cFE/CFS
  - ROI estimation

Acknowledgement

- Lisa Montgomery - NASA SARP for supporting our applied research
- Sally Godfrey – NASA GSFC

- Thank you!

Acronyms

- API: Application Program Interface
- cFE: Core Flight Executive
- CFR: Code of Federal Regulations
- CFS: Core Flight Software
- FSW: Flight Software
- MBT: Model-based Testing
- OSAL: Operating System Abstraction Layer
- ROI: Return on Investment
- SUT: System Under Test