UDI Driver Test Suite Design

by

Gian-Carlo Bava (SCO)
Barry Feild (SCO)
Andrew Knutsen (SCO)

Change History

Revision	Date	Description
0.1	10/06/1999	Draft.
0.2	6/8/00	Use GIO mapper

Approvers:

TBD
TBD
TBD

1 Introduction

This document contains the design for the UDI driver test framework. This framework supports a variety of types of UDI drivers. This document is supplemented by test specifications for each driver metalanguage (NIC, SCSI, etc).

Presented first are the design goals. The design proposal overview follows. The design proposal overview is a high level view of how the test suite is constructed. Following the design proposal is the detailed design of the interfaces and components.

Finally, Portability, Extensibility, Packaging, and Documentation design detail is presented.

Each design section begins with a proposal section. The proposal section describes the design decisions that need to be made.

1.1 References

UDI Network Driver Test Suite Specification
http://hamilton.pdev.sco.COM/Projects/uditspec.htm

This is the UDI NIC driver test specification document. The UDI NIC driver test purposes are described with pass/fail points given for each test case.

1.2 Terminology

- Test Purpose: A low-level operation that can be performed on a driver.
- Test Case: A series of test purpose operations designed to test some aspect of a driver.
- Test Suite: A collection of test cases intended to test a specific type of driver.
- UDI Channel: A communications "pipe" used to link two UDI modules.
- UDI Metalanguage: The set of operations used over a specific type of UDI channel.
- UDI Mapper: A software module which converts one UDI metalanguage either to another metalanguage or to an OS-specific interface.
- GIO Metalanguage: The metalanguage used on Generic IO channels.

2 Design Goals

- Portable across UNIX platforms.
- Enable developers to write quality UDI drivers with a shortened and efficient test cycle.
- The architecture is extensible. Test suites for more metalanguages can be easily added. New test case operations can be easily added.
- The test is automated. Once configured and started the test cases run to completion unattended.

3 Design Proposal Overview

The proposed design consists of three functional blocks:

- A test controller module which interfaces with the driver under test, using its required metalanguage.
- The test cases. These are generally scripts, implementing particular test scenarios.
- The test purpose utility. This utility is called by the test cases, and passes purpose operations down to the test controller.

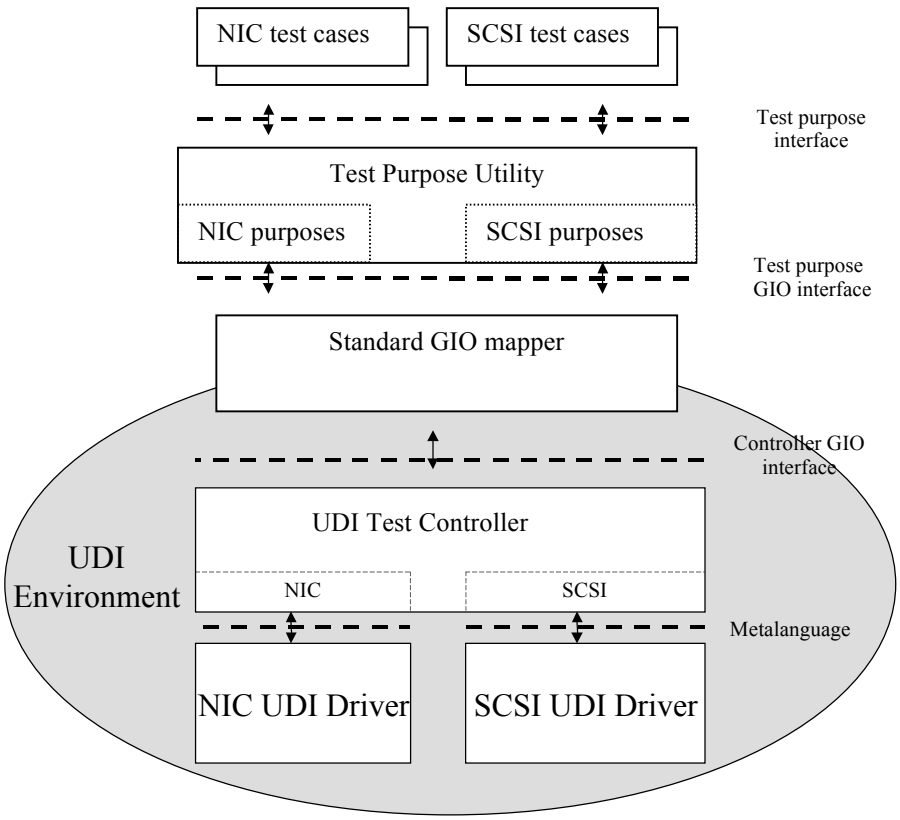
The test purposes and the controller communicate via the GIO (Generic I/O) mapper and metalanguage. Thus there are three interfaces:

- The defined sets of test purposes (one set for each driver metalanguage).
- The interface between the test purposes and the GIO mapper.
- The GIO interface to the test controller.

The UDI Test Controller is an internal mapper that is a substitute for the metalanguage interface provider. For example, the UDI Test Mapper would take the place (for purposes of testing) of the operating system Network Service Requester (NSR). In this case, the Test Controller uses NIC metalanguage channel operations to put a UDI NIC driver under a controlled test.

A diagram of the architecture is shown below.

UDI Driver Test Suite Architectural Overview



4 UDI Driver Test Cases

4.1 UDI Driver Test Cases Overview

The UDI drivers test suite includes a collection of test cases for NIC and SCSI UDI drivers. A test case is a logical set of operations that indicate a pass or fail result. Each operation that produces a pass or fail result is a test purpose.

The test specification documents describe the test cases and the test purposes for each test case. For each test purpose, the operation, relevant parameters, and the expected result from the operation is specified. The expected result is the determination point of whether the test purpose passes or fails.

For example, the expected result might be the status value of UDI_OK returned from a control block operation. When the test is run, if UDI_OK is returned as the status value of the control block, then the test purpose passes. If not, the test purpose fails. If any test purpose within a test case fails, the entire test case fails; otherwise it passes.

The test specification is used to implement three things:

- A set of test case scripts
- A program, called by the scripts, which passes the purposes down to the controller via the GIO mapper
- Code in the controller implementing the purposes

4.2 Test Purpose GIO Interface

The test purpose programs interface to the GIO external mapper using the read/write interface. Data structures are passed giving the operations (purposes) and arguments. Endianness should not

be an issue when communicating with the Test controller, since both are compiled for and running on the same architecture. The driver should handle hardware dependent endianness.

The GIO mapper will convert the write requests into GIO metalanguage operations, and conversely convert GIO operations into responses to read requests. Certain behaviors of the GIO mapper that have not yet been specified may have to be assumed. If this is the case, such behavior will be submitted for inclusion in the core UDI specification.

5 Test Controller

5.1 UDI Test Controller Overview

The Test Controller is a UDI internal mapper. Its basic function is to translate between the GIO channel communicating with the test cases (via the GIO mapper and the purpose program) and the metalanguage of the driver under test.

Each purpose operation is translated into one or more driver operations. For instance, a single purpose may cause a series of packets to be transmitted by a NIC. The controller also passes device status and data that is read or received back up to the cases via the GIO channel.

The controller is written in a modular fashion, so that additional metalanguages can be added without breaking existing code. (Separate UDI modules?)

5.2 Controller GIO Interface

The controller accepts two basic types of requests:

- Purpose operations (as defined by the appropriate test specification)
- Operations affecting internal state

The following internal operations are defined:

- NIC receive buffer flush

5.3 Configuration and Initialization

Configuration considers what must be in place before any of the test software begins to run. Configuration sets up the driver under test, the Test Controller, and the test cases with the proper environment and configuration parameters to run the test. The Test Controller needs to be configured as a substitute UDI metalanguage provider for the driver under test, and linked under the GIO mapper. Test cases may have configurable parameters.

There are two framework-related configuration and initialization issues:

- How the controller module is bound to the GIO mapper and the driver under test. Static binding will be used for both of these channels.
- The relationship between the driver under test and any drivers required by the system on which the tests are running. For instance, in the NIC case, there will be two NSRs present: the

test controller and the normal NIC mapper. If the network is to be usable during tests, one driver must be linked under the test controller and another under the NIC mapper.

It is possible normal networking will not be available during test. However, what about SCSI devices? Currently, only implementation-specific mechanisms are provided to choose between multiple metalanguage providers.

6 Portability

The elements of the test suite outside of the UDI environment, such as the test cases and purposes, need to be constructed in such a way that they can be built to run on any UNIX platform.

Elements inside the UDI environment such as the bottom half of the Test Controller, the Test Mapper, and the UDI driver under test are portable by definition.

The GIO mapper lies between these two areas. Although it is part of the UDI environment, all details of its behavior are not specified. This problem will be addressed in two ways: the controller will be adapted to handle multiple common behaviors if necessary, and clarifications to the specification will be submitted to Project UDI where it makes sense.

7 Extensibility

The design considers two vectors of extensibility.

The test framework can be extended to test drivers based on other metalanguages. A set of test cases and purposes may be developed for a metalanguage other than NIC or SCSI, and the controller extended to implement the new purposes.

The test framework can be extended within existing metalanguages. NIC or SCSI test cases may require new test operations. Such extensions will impact all components of the existing metalanguage test implementation.

8 Packaging

Packaging considers source and binary distribution, versioning, installation, and removal of the test suite. The distribution format, install, and remove technology must be portable to all UNIX platforms. The packaging design must be extensible for easy addition or removal of components.

The standard UDI install format meets these requirements. If possible it will be used.

9 Documentation

UDI Test Purpose Utility Man Page
Test Case Functionality Specification
Installation and Configuration Guide