Automated Integration Testing of Measurement Devices
A Case-Study at AVL List GmbH

Bachelor's Thesis

Graz University of Technology
Institute for Softwatretechnology
Supervisor: Ao.Univ.-Prof. Dipl-Ing. Dr.techn. Bernhard Aichernig
Co-Supervisor: Dipl-Ing. Robert Korosec (AVL List GmbH)

Graz, August 2013
Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, ___________________________  ___________________________
Date                        Signature

Eidesstattliche Erklärung¹

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ___________________________  ___________________________
Datum                        Unterschrift

¹Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008; Genehmigung des Senates am 1.12.2008
Abstract

This thesis discusses the challenges of generating complex test-suites with model-mutation and their execution. All work was performed during a research-project called TRUFAL ("TRUst via Failed FALSification of Complex Dependable Systems Using Automated Test Case Generation through Model Mutation"). In this project the AVL List GmbH is an industrial-partner and implements an use-case. The aim of this use-case is to develop an environment for test-transformation and -automation for executing abstract test-cases for software-tests. These tests are automatically generated by model-mutations. During the project all tools were successfully developed. The shown way for test-automation is used productively in AVL’s qualification tests.
# Contents

Abstract iii  

1 Introduction 1  
  1.1 Motivation and Goals ........................................... 1  
  1.2 Structure .......................................................... 2  

2 Mutation Based Testing 3  
  2.1 A simple mutant .................................................. 3  
  2.2 Test-case generation ............................................. 4  

3 Application Domain 6  
  3.1 PumaOpen Layers .................................................. 7  
  3.2 Test Setup .......................................................... 8  

4 AVL489 - Particle Counter 10  

5 UML Model 12  

6 AVL TestAutomationFramework 15  
  6.1 Overview ........................................................... 15  
  6.2 MeasurementDevice PageObject Base Class ...................... 16  
  6.3 AVL489 PageObject ............................................... 18  
  6.4 Automated Testcases ............................................ 20  
  6.5 Test Setup and Teardown ....................................... 21  

7 Testcase Transformer 22  
  7.1 Aldebaran File Format ........................................... 22  
  7.2 Convert a simple Testcase ...................................... 22  
  7.3 Mapping ............................................................. 24  
  7.4 Integration into the Framework ................................ 25  

8 Related Work 27  
  8.1 AVL internal test automation .................................... 27  
  8.2 Unit-testing ......................................................... 28  
  8.3 Test-case generation ............................................. 28  

9 Conclusion 29
Contents

Bibliography 30
# List of Figures

2.1 Correct Model of a simple Car-Alarm-System ........................................... 4  
2.2 Mutant of a simple Car-Alarm-System ..................................................... 4  
2.3 Testcase Generation and Evaluation ....................................................... 5  
3.1 Connection between PumaOpen and AVL489 ............................................. 6  
3.2 Network plan ................................................................. 8  
4.1 Simplified Flowchart ................................................................. 11  
5.1 Operating State Region ................................................................. 13  
5.2 Context-diagram ................................................................. 14  
6.1 PageObjects and Testcases ............................................................. 16  
6.2 MeasurementDevice Class Diagramm .................................................... 17  
6.3 AVL489 PageObject Class Diagramm .................................................... 19  
7.1 NUnit User Interface ................................................................. 26
Listings

6.1 Example Testcase .................................................. 20
6.2 Testcase Setup ..................................................... 21
6.3 Testcase Teardown ................................................ 21

7.1 A simple testcase defined in the Aldebaran File Format ............... 23
7.2 A simple testcase converted to C# .................................... 24
7.3 Mapping for an observable defined in XML .......................... 24
7.4 Mapping for a controllable defined in XML ............................ 25
1 Introduction

TRUFAL is a research project funded by the FFG (Österreichische Forschungsfördergesellschaft). The project is realized by four partners. These partners are the Austrian Institute of Technology (AIT) and Graz University of Technology (TUG) as research partners and Thales Austria GmbH and AVL List GmbH as industrial partners. The acronym TRUFAL stands for “TRUst via Failed FALSification of Complex Dependable Systems Using Automated Test Case Generation through Model Mutation”. This thesis deals with the experiences and results of the AVL-Usease.

1.1 Motivation and Goals

AVL List GmbH has a business unit called Instrumentation and Testsystems (ITS) [8]. This unit produces measurement devices and software for the automotive industry. Many of these hardware and software products are combined to testbeds for combustion engines. All these products are verified by the internal AVL Testcenter.

Product-updates and -extensions can only be released if they pass the qualification-tests in the Testcenter. These tests prove the compatibility and integrity of the interfaces. Many of them need to be executed manually. To increase the test-quality and to simplify the testers life there is a demand for test-automation and automated test-case-creation.

This thesis is a closer look to the improvements in automated integration-testing of measurement devices in the testbed-automation-system (PumaOpen) reached within TRUFAL. These devices are integrated with drivers into the PumaOpen-system.

The most important goals of this new approach are to increase the speed and test-coverage of device-driver-tests.

Some other arguments and requirements for large-scale test-automation are:

- Automated tests can automatically run over-night without user interaction.
- Precise error descriptions (e.g. device is in state A and the following change to state B failed)

To deal with these requirements measurement-device-tests and -interface were integrated into the already existing AVL TestAutomationFramework. This framework is used in the product test (development departments) as well as in the fully integrated system test
1 Introduction

(Testcenter). The TRUFAL-project provides a technique to generate test-cases from UML-models. With the developed automation-interface (TestAutomationFramework) it is possible to execute these generated abstract test-cases.

1.2 Structure

Section 2 is a description of Mutation Based Testing and how this approach for test-case generation works. In the scope of the AVL-Usecase the goal is to automatically generate test-cases for a measurement device called AVL489. Section 3 discusses the application domain and the testing environment. The used measurement device is described in detail in Section 4 "AVL489 - Particle Counter".

To run automated tests the AVL PumaOpen system was abstracted with a C#-framework. A description of this framework can be found in the corresponding Section 6. The tests generated with the TRUFAL-toolchain are defined in an abstract file format. To integrate these test-cases into the framework it is necessary to convert them to C#-Unit-tests. Section 7 is a description of this transformation process.

Other techniques for unit-testing, test-case-creation and test-automation are discussed in Section 8 "Related Work".
2 Mutation Based Testing

The goal of the automated generation of test-cases by mutation is to create tests with high test coverage. For this purpose a model of the System under Test (SUT) is created with the Unified Modeling Language (UML) [27]. In the next step this model is mutated. Mutants are defective variants of the model.

Some examples for defective variants are:

- Transitions with changed destinations (e.g. a transition from state A to C instead of A to B)
- Changed conditions (e.g. \(x \geq 10\) instead of \(x < 10\))
- Removed transitions

This sort of test-case-generation relies on the two following assumptions [5, p. 1]:

- The competent programmer hypothesis states that programmers are skilled and do not completely wrong. It assumes that they only make small mistakes.
- The coupling effect states that test cases which are able to detect simple faults (like faults introduced by mutations) are also able to reveal more complex errors.

2.1 A simple mutant

Figures 2.1 shows the correct version of a simple car alarm system. On the one hand the model represents the states of the door and if the car is locked (OpenAndUnlocked, ClosedAndUnlocked, OpenAndLocked, ClosedAndLocked). On the other hand it also shows the states of the alarm system (Alarm, SilentAndOpen, Armed).

In Figure 2.2 there is a simple mistake. It is a so called mutant. Instead of changing the state from OpenAndUnlocked to OpenAndLocked the mutant remains in OpenAndUnlocked when the transition Lock is triggered.
2 Mutation Based Testing

![Correct Model of a simple Car-Alarm-System](image1)

**Figure 2.1:** Correct Model of a simple Car-Alarm-System [2]

![Mutant of a simple Car-Alarm-System](image2)

**Figure 2.2:** Mutant of a simple Car-Alarm-System [2]

2.2 Test-case generation

Figure 2.3 shows the test-case-generation and -evaluation process. A Model Mutant is generated from the original Model by the Mutation Tool. The generation is successful if the mutant does not conform to the initial model. For each model-mutant a special test-case is generated. The purpose of this test-case is to find the mutation. The Test Case Generator compares the Model with the Model Mutant and builds a test-case to show up...
2 Mutation Based Testing

the difference between them. Within TRUFAL action-systems are used to compare the models [4]. This test-case "searches" its way through the state-machine till it reaches the difference between them. Each test-case is a sequence of observables and controllables. More precisely the sequence is a set of state-changes and expected observations. This sequence is called Abstract Testcase. With the Test Driver these generated test-cases are executed on the SUT. If all these test-cases are executed successfully (SUT conforms the initial Model), the SUT is not equal to one of the mutants. This result implies, that the system is correct in this region. If one or more test-cases fail (SUT conforms one Model Mutant), the SUT is equal to this mutant in the tested region. This indicates that the system has an error. "If not at least one test case is able to kill a mutant, the test suite has to be improved."[5]
3 Application Domain

The generated test-cases are applied to an AVL PumaOpen-system [10]. PumaOpen is a testbed-automation-system for combustion engines. It provides the integration of testing-tools and testbed-components. Testing a fully integrated PumaOpen-system is very complex. This system is an interaction of a high number of different software products. All these programs or subsystems are tested on its own in the specific development departments. In the AVL Testcenter these products are combined and tested in a compound (integration-tests). The prototypes for automated system tests focus on the integration of measurement devices. Figure 3.1 shows the relevant components of a PumaOpen-system for the device integration. It also shows the layers which have to be passed to access a measurement device. All these layers are tested. Only the graphical user interface is excluded. However, this graphical layer has to be tested manually anyway.

![Figure 3.1: Connection between PumaOpen and AVL489](image_url)
3 Application Domain

3.1 PumaOpen Layers

Activation Objects

Activation objects are the central access points for nearly all features of the PumaOpen-System. The objects are abstraction layers for these features, so the access to these resources is very simple. Examples for activation objects are measurement devices and blocks for recording measurement devices.

Configurable Device Handler

The Configureable Device Handler (CDH) is a layer embracing the connection to all measurement devices. It consists of abstract design rules for all the underlying devices. For example, there is a set of methods for each device which are executed when the PumaOpen-System changes its state (e.g. from Monitor to Manual). For each type of device an abstract definition exists which is implemented by the particular device driver.

Device Driver

The device driver of all measurement devices implements the abstract rules of the CDH-framework. In simplified terms the communication between the PumaOpen-System and the device works with two components. On the one hand there are sequences, which are compounds of commands sent to the device. For example there is a sequence to change the device state to Standby. Other examples are cyclic sequences which poll the device states and values or special dotted-sequences which are automatically executed when the PumaOpen-System changes its state. These sequences are called "dotted-sequences" because their name starts with a dot. On the other hand the device driver writes the device’s states and values to PumaOpen system-channels. System-channels are global accessible variables in the whole PumaOpen-system.

So if a sequence is executed, the results get visible on the system-channels after the next polling-process.

Ethernet

The communication between the PumaOpen-System and the measurement device is mostly based on the AK-protocol [1]. This is a proprietary protocol for communication with measurement devices used by AVL. A definition of the protocol is delivered for each measurement device only on request. In simplified terms this is a TCP/IP plaintext protocol. All commands have a length of 4 letters (e.g. AKEN, STBY) followed by parameters which are separated by spaces. A distinction is made between query- (A**), adjustment- (E***) and status-commands (S***). This nomenclature comes from the
German words "Abfrage" (query), "Einstellung" (adjustment) and "Status" (status). The answer-telegram starts with the executed command followed by an error value and all answer parameters.

### 3.2 Test Setup

A test of a fully integrated PumaOpen-System is very complex and expensive. Every update of the system should be tested on a testbed which is equal to the customers. To avoid the costs of physical testbeds and to reduce the complexity AVL uses virtual testbeds. A virtual testbed is an exact copy of a hardware-system, but all components are simulated. The combustion engine as well as all measurement devices are simulated in a real-time software called TBSimu. This improvement allows AVL to test a product only with two computers instead of a complex and very expensive testbed. The first computer is a PumaOpen installation equal to the customers installation. On the second computer the simulation runs and delivers all data via the network interface. The only difference for the PumaOpen-computer between a real and a simulated environment is that all measurement devices have the same IP-address. TCP-port numbers do not change.

The PumaOpen-System has two network interfaces. The first one is connected to the real-time part and the second one is connected to the non-real-time part of PumaOpen. The simulation computer has only one network interface. Figure 3.2 shows the detailed network plan of the installation.

The TBSimu-program is used to simulate the measurement devices. These simulated devices behave like real measurement devices. The values for these devices are calculated in a AVL ARTELab Studio real-time environment. AVL ARTELab Studio is a toolset which uses Tenasys InTime [26] for executing real-time programs. This InTime-kernel cuts one core and a few hundred megabytes (typically about 300MB) of ram to run beside Windows 7. The InTime-environment delivers a hard-real-time-system which is strictly separated to the Windows-system. ARTELab Studio delivers the functionality
3 Application Domain

to compile Matlab Simulink files to realtime executables. It also allows to execute these programs and to access the calculated values from the non-real-time-part of the system. To improve the simulation of all devices the simulated time is equal to the real time. If a TBSimu-measurement-device is treated as a black-box it is not possible to make out if it is a real or a simulated device.
4 AVL489 - Particle Counter

This Section is a description of the device and its measuring principle. It is based on the chapter named "Operation" in the device product guide [9, p. 27].

The sampled exhaust gas is extracted from the off-gas-tunnel and diluted with high-efficiency particulate air (HEPA). For the dilution the AVL Chopper Diluter is used. To leave behind only solid particles the thinned exhaust gas is heated. Caused by the temperature the volatile components are vaporized. After this process the gas is diluted a second time and fed into the particle number counter (PNC).

Both dilutions work with a rotating disk diluter (Chopper diluter). The dilution can be changed with a few parameters like rotation-speed, the number of holes or the diluter temperature.

To count the particles, butanol is condensed onto the diluted exhaust gas. The deposed butanol enlarges the particles so they scatter laser light on passing a laser beam. This makes it possible to count the particles per volume unit. With the known dilution ratios the real value of particles per $cm^3$ (ppccm) can be calculated.

The device has a mounted terminal and an ethernet interface. All parameters and measurement values can be changed and sampled with both techniques, but it is only possible to use one of these at the same time.
2.1 Measuring Principle
Exhaust gas is sampled from a CVS tunnel and diluted with HEPA filtered compressed air using the AVL Chopper Diluter. Inside the evaporation tube the diluted exhaust gas is heated to a degree that causes the volatile emission components to vaporize, leaving behind nothing other than solid particles. After that, the exhaust gas is diluted again using a porous tube diluter and fed into the condensation particle counter (PNC). In the PNC, butanol is condensed on to the particles inside the exhaust gas to enlarge them so that they become visually detectable. The enlarged particles are then counted based on the scattered light pulses generated when the particles pass through the laser beam. This makes it possible to determine the number of particles per volume unit.

According to PMP specifications, particles that exceed the size of 23 nm are measured with an efficiency 50 ± 12%.

2.2 System Overview
The AVL Particle Counter is a measurement device for counting particle numbers, which complies with the UN/ECE PMP and Regulation No. 83. The figure below shows a simplified flow chart – for a detailed overview of all valves, filters, etc., see Fig. 9 on page 38 and Fig. 10 on page 39.

Figure 4.1: Simplified Flowchart [9, p. 27]

Figure 4.1 shows a simplified flowchart of the measurement process. The first dilution is shown on the left side of the figure (chopper diluter). The sample inlet (red line) is diluted with filtered air (blue line). The diluted output (green line) is filled in the evaporation tube (green line ET) and then diluted a second time (green line PND_2). After this procedure the particles are counted by the particle number counter (green line PNC).
5 UML Model

The UML-Model of the AVL489 particle counter is made up of three independent sub states. These states are named “control state”, “transition state” and “operating state”. The operating state is the main state and defines the operation the device is currently performing. The control state determines if the measurement device is controlled by the terminal (Manual) or if it is controllable via remote control (Remote). The transition state defines if the device is processing the last action (Busy) or if it is waiting for now instructions (Ready). The model is separated in three parallel regions. Each region describes the behavior of one sub-state.

Figure 5.1 shows the operating-state region. When the system starts up it enters the state Pause_0. On entering the state the defined commands are executed. In this case (state Pause_0) the system sends the observables Pause (send SPAU_state) and Busy (send StatusBusy) to the test-environment. The behaviour of changing the state from Busy to Ready is defined in the parallel “transition state”-region.

To switch from Pause_0 to Purging_Pause_12 the transition SetPurge has to be triggered. Additionally the system has to be ready and online (condition [not Busy and not Manual]). On entering the state Purging_Pause_12 the system sends the observable Purging (send SPUL_state). After remaining 20 seconds in this state the system automatically switches back to Pause_0.

On the right side of Figure 5.1 seven states are embraced by the the state Active. This state models the capability of the system to switch back to Pause_0 and Standby_1 from all these substates.

A more detailed description of all three regions can be found in the corresponding chapters of the test-model-document called “TRUFAL Final Test Model AVL” [7].
Figure 5.1: Operating State Region
5 UML Model

In Figure 5.2 the commands and signals of the system under test are displayed as context-diagram. These commands and signals are called controllables and observables. The controllables are the commands PumaOpen is able to send to the device (CDH-sequences). These controllables are listed in the «system_under_test»-box called AVL489 and marked with the flag «from_environment». On the left side of the diagram the observables are defined within the «environment»-box called TestEnvironment. These observables are representing the values of PumaOpen-system-channels. The PumaOpen-system is polling with query-commands from the device and writes the returned values to system-channels. So the current states of the device are always accessible by simply reading the corresponding system-channel. The mapping from these abstract objects (observables and controllables) to the real ones (system-channels and commands) is implemented by the test-driver.

The used modelling-tool is Papyrus-UML. Based on this model the mutation-tool built by AIT generates mutants. In the context of this thesis AIT was supported with device-details and feedback to build and improve this model.
6 AVL TestAutomationFramework

6.1 Overview

The AVL TestAutomationFramework is an AVL-internal software tool with the goal of bundling all test automation code into one source tree. With this framework AVL avoids redundancies and expedites the reuse of test automation code. The development of this toolset started in the year 2012 with the following goals [23]:

- Common and modern programming language
- Integrated Development Environment
- Widely used
- Easy to learn
- Use existing APIs
- Refactoring
- Nearer at development
- Read- and writeable for non-programmers

Some of the design decisions are borrowed from the Google Selenium Project [15]. Especially the use of PageObjects is based on their concept. A PageObject is an abstraction layer for a specific PumaOpen component. A component can be a measurement device, a special parameter block or even the PumaOpen-System itself. These PageObjects are facing in two directions. On the one hand they declare all services provided by the component for the developer of the tests. On the other hand the PageObject is the only thing which has knowledge about the exact structure of the component [14].

A test case uses a subset of the provided PageObjects. Figure 6.1 shows a sample-usage of PageObjects within test-cases. For example Page Object 1 and Page Object 2 are used in Test Case 1. Page Object 2 already uses underlying interfaces (SOAP and COM). On the other hand Page Object 3 only uses HP QTP (HP Quick Test Professional [16]) as underlying interface to implement its functions.
The TestAutomationFramework uses C# as programming language (same as in development departments) and NUnit [22] for executing the tests. NUnit provides all necessary functionality to execute fully automated unit test with the AVL PumaOpen system.

In the scope of this thesis the infrastructure for testing device-drivers of measurement devices was built (integration-test). The classes discussed in the following Subsections were built on top of the base-functionality provided by the PumaOpen-test-department. The main parts of this work were to develop the design principles for the integration of measurement devices into the TestAutomationFramework and the concrete implementation of these principles for the AVL489.

6.2 MeasurementDevice PageObject Base Class

The MeasurementDevice class is the abstract base class for all automated measurement devices. It provides a standard set of the most important methods. Most of them are already implemented and are ready for usage by the subclasses.

- **SendAK** sends AK-command to the device. This is a low level functionality which is only used for debugging purposes and special commands. These special commands are actions which are not accessible with CDH-Sequences. To use this method it is necessary to specify the abstract properties `AKResponseChannel` and `AKScriptName`. `AKScriptName` is the name of the script which has to be triggered for sending an AK-command. The name of the system-channel which stores the answer to the sent command is stored in `AKResponseChannel`.

- **WaitForChannelValue** waits until the value of the specified channel has changed to a specified value. The timeout and polling-frequency are parameters of this method. If the desired value is not reached an exception is thrown.

- **WaitForEndOfChannelValue** has the same functionality like `WaitForChannelValue` with the difference that it waits until the value has changed.
Especially the wait-methods are very important for all measurement devices. Nearly every child-class implements another wait-method which is based on one of these methods. On the other hand this base-class defines a few design-principles for all measurement devices, for example how the device’s PumaOpen system-channels are accessed (getChannelList returns a list of Channel-objects to access the underlying system-channels).

Figure 6.2 shows the class-diagram of this abstract class. Italic written properties and methods are abstract. All others are already implemented.

![MeasurementDevice Class Diagramm](image)

On the other hand the MeasurementDevice-class also provides some protected methods for simplifying the access to sequences. For example SendAndWaitForSequence executes a sequence and waits until the value of a defined system-channel(with data-type T) has changed its value.
6.3 AVL489 PageObject

The AVL489-PageObject is an abstraction layer for accessing the measurement device using a design pattern called facade [11]. This PageObject implements the abstract base-class MeasurementDevice. The methods of the AVL489-class also use the methods which were already defined in the abstract base-class (e.g. WaitForChannelValue).

The following three points describe the provided main-functionality:

- Access to the device specific PumaOpen system-channels. The most important system-channels can be accessed by public properties (e.g. AVL489.State). All other channels, which aren’t used regularly, are accessible by the getChannelList-method.

- Methods to change the device-state (e.g. AVL489.Standby() switches the device to Standby). Most of these commands can only be executed if the device is in an appropriate state (e.g. the device has to be Ready).

- Methods to wait for events (e.g. AVL489.WaitForState() waits a defined time until the device reaches the state).

The PageObject provides all actions which are accessible by the PumaOpen Graphical User Interface. The measurement device supports much more commands than the Puma-Device-Driver implements. To access these special functions the SendAK-method can be used. Figure 6.3 shows the UML-class-diagram of this PageObject.
Figure 6.3: AVL489 PageObject Class Diagramm
6 AVL TestAutomationFramework

6.4 Automated Testcases

With the AVL489-PageObject it is possible to run all device-driver-tests automatically. Listing 6.1 shows an automated test-case which shows the usage of the base PageObjects and the AVL489 measurement device.

```
[Test]
public void TestCase()
{
    ApplicationManager applManager = new ApplicationManager();
    Puma puma = applManager.StartPumaOpen();
    puma.GoToManual(true);
    AVL489 avl489 = puma.OpenMeasurementDevice<AVL489>();
    avl489.Remote(true);
    avl489.Standby(true);
    avl489.Close();
    puma.GoToMonitor();
    puma.Close();
    applManager.Close();
}
```

Listing 6.1: Example Testcase

The ApplicationManager PageObject starts and represents the running ApplicationDesktop (Line 4). It provides a method which starts the PumaOpen-System and returns a Puma PageObject (Line 5). The Puma-PageObject describes the running PumaOpen-Testbed-Automation. The PumaOpen System changes its states from Monitor to Manual in Line 6 and provides the abstraction of the AVL489 measurement device in Line 7. The test switches the AVL489 to Remote and to Standby. These two methods also include the checks for the expected values of the system-channels (parameter true). This indicates if the execution was successful. If the system-channels do not have the expected values an exception is thrown. Indicated by the exception the test is cancelled and the test-case is marked as failed. With lines 12 to 15 the whole system performs a shutdown (Close-Methods).

All test-cases operated in the TRUFAL-context use at least the following three PageObjects:

- ApplicationManager
- Puma
- AVL489
6.5 Test Setup and Teardown

To reuse defined functionality for preparing, for ending and cleaning up the system NUnit delivers special marked methods. The code from 6.1 is splitted into three methods. The methods marked with TestFixtureSetUp and TestFixtureTearDown are automatically executed by NUnit before and after the execution of all tests.

```csharp
[TestFixtureSetUp]
public void Init()
{
    applManager = new ApplicationManager();
    puma = applManager.StartPumaOpen();
}
```

Listing 6.2: Testcase Setup

```csharp
[TestFixtureTearDown]
public void Cleanup()
{
    puma.Close();
    applManager.Close();
}
```

Listing 6.3: Testcase Teardown

There are also methods which are executed before and after every single test-case. These methods handle the creation and teardown of the AVL489-PageObject. The real test-case without setup and teardown (Listing 6.1 lines 7 to 12) is stored in a separate function marked with the [Test] sign.
7 Testcase Transformer

The TRUFAL tool chain generates test-cases in the abstract Aldebaran File Format [6]. The file extension for these files is .aut. To use the testcases with the AVL TestAutomationFramework they have to be converted to C#-NUnit-Tests. The transformer works with simple text substitution which is defined in a mapping file. This transformation-tool was developed in the scope of this thesis.

7.1 Aldebaran File Format

The Aldebaran file format is a simple and abstract way to describe test-cases.

The first line of each file is the definition about the scope of the test-case.
\[ \text{des}( \text{<first-state>}, \text{<number-of-transitions>}, \text{<number-of-states>}) \]

The following lines describe the transitions.
\[ (\text{<from-state>}, \text{<gate-name>}, \text{<to-state>}) \]

In the scope of TRUFAL the gate-name can be separated into two types:
- Observables are detected changes of the SUT’s state
- Controllables are “state-changing” actions which can be sent to the SUT by the automation system

7.2 Convert a simple Testcase

In Listing 7.1 there is a simple test-case in the aldebaran-file-format. This test-case is used to illustrate the transformation process.
Listing 7.1: A simple testcase defined in the Aldebaran File Format

In the correct order these statements have the following meanings:

1. Observe state Offline
2. Observe state Pause
3. Send device to Remote
4. Observe state Remote
5. Send the device to Standby
6. Observe state Busy
7. Observe state Standby
8. Wait until device is Ready

Listing 7.2 shows the output of the transformed version of this test-case. The transformation-tool reads the abstract test-cases, sorts the statements and replaces each one with a C#-statement.
7 Testcase Transformator

Listing 7.2: A simple testcase converted to C#

```csharp
[Test]
public void Simple_Testcase_1()
{
    avl489.WaitForManual(0);
    avl489.WaitForState(AVL489.States.Pause, 0);
    avl489.Remote(false);
    avl489.WaitForRemote(0);
    avl489.Standby(false);
    avl489.WaitForBusy(0);
    avl489.WaitForState(AVL489.States.Standby, 0);
    avl489.WaitForReady(30);
}
```

7.3 Mapping

All information about the output is stored in a XML-file. This includes the file- and function-headers and -footers for the target format.

Examples for the file header are the using-directives, namespace- and class-definition of the C#-class. The function-header defines the method-prototype of the testcase (e.g. [Test]-attribute, public, ...). The current used footers only consist of a defined number of braces to generate a correct method- and class-definition.

The definition for the transition substitution is shown in Listing 7.3. This is a sample for the observable `obs SPAU_state(0)`. The transformer can use the parameters of the gate-names. These parameters can be used with the `#*#`-Tags.

```xml
<Transition>
    <Type>Observable</Type>
    <Action>SPAU_state</Action>
    <Command>avl489.WaitForState(AVL489.States.Pause, #1#);</Command>
</Transition>
```

Listing 7.3: Mapping for an observable defined in XML

With this definition the observable `obs SPAU_state(0)` (with parameter 0) would be converted to `avl489.WaitForState(AVL489.States.Pause, 0);` (timeout is 0).

The definition for a controllable looks very similar and is shown in Listing 7.4.
For each observable or controllable a transition-definition has to be created. The mapping-file is a serialized C#-class [19]. During execution of all mapping-entries are deserialized and stored in a List of transition-entries. The lookup within this list is done by LINQ-statements [18].

If an entry is not found in the mapping-file, the tool prints an error-message in form of a comment into the file.

### 7.4 Integration into the Framework

Within the TestAutomationFramework there are already existing automated test-cases for the AVL489. These test-cases are the automated variants of the previously manually executed test-cases. The generated file is included in the standard-build-process of the TestAutomationFramework. To split the AVL489Test-class into two files the C#-concept of partial classes was used. If the TRUFAL-test-cases are enabled this makes it possible to see all tests within one category in the NUnit-tool. Figure 7.1 shows a running test in the NUnit user-interface.
7 Testcase Transformator

Figure 7.1: NUnit User Interface
8 Related Work

8.1 AVL internal test automation

Caused by the size of the company AVL uses a widespread toolset of test-automation and test-case-generation tools. The following points are the most interesting tools.

- **Cameo test-automation**
  Cameo is a software for engine and transmission calibration. For testing this software-product AVL uses a self-made test-deployment- and test-execution-framework based on HP quick-test [16]. This testing-environment is probably the most used test-automation-framework by AVL.

- **fOX test-automation**
  fOX is a post-processing software for engine calibration. To test the product-releases of this product AVL uses a model-based approach. In this process UML-Diagrams extended with keywords are used. This solution is also integrated in Microsoft Team Foundation Server and Visual Studio.

- **Cobra Unit-Test Framework**
  To test the AVL Cobra products, which are special extensions to MATLAB Simulink to generate real-time executables and a tool-set to run these, a special approach is used. The tools are based on Google’s C++ Testing Framework [13] and C++ Mocking Framework [12]. For the implementation and extension of this tool-set a research project with the University of Salzburg was founded.

- **Octopus with Test-Verification**
  AVL Testcenter uses a new special variant of combined unit-test and measurement-verification. This toolset is used to test an integrated PumaOpen System in Unmanned Mode and is based on the TestAutomationFramework. This test-method triggers unit-tests which switch the PumaOpen-system to Automatic-mode. In this mode measurement is enabled and the produced data is recorded. After finishing this process the recorded data is compared with a reference data set. A test is successful if the deviation is within defined bounds. This testing method is very complex but it delivers very detailed results.
8 Related Work

8.2 Unit-testing

In the scope of this thesis NUnit [22] was used. NUnit is a .Net-porting of JUnit [17] to use especially with C# and Visual Basic. There is a wide spectrum of other tools which could be used instead. NUnit was used because it is open-source software. Another example, which would also fulfil similar requirements, is Microsoft’s MSTest [25]. Other interesting examples for C++ unit- and module-testing are the Google C++ Testing and Mocking Framework [13, 12] and the QT QTestLib [24].

8.3 Test-case generation

The quality of the generated test-cases in the context of TRUFAL is not yet evaluated, but it might be better than the results of the preceding Mogentes project [21].

There also exist other techniques for automatically generating test-cases. A very interesting example is Microsoft PEX [20]. PEX automatically finds interesting input-output for methods. It provides a tight test-suite with high code coverage and is fully integrated into Microsoft’s Visual Studio.
9 Conclusion

On the preceding pages the mutation-based-testing, the application-domain and -goals were discussed. Then the developed tools and testing-environment were described.

The current state of the project is that the development of the transformation- and testing-tools are finished. For the test-case-generation currently the first working version of the new tool-chain (called MoMuT 2013.2) is used. The use-case of generating, transforming and executing test-cases is working successfully. Presently the test-case generation tool is improved by the academic-partners and on the other hand the industrial partners are reviewing the generated test-cases. At the moment the used modelling-tool is also switched from Papyrus UML to Visual Paradigm.

The future tasks in the context of the TRUFAL-project will be the improvement of the test-case-generation and the modelling-techniques with the new tools.

In the AVL Testcenter some alternative use-cases of test-automation were already discovered. For example it would be possible to extend the TestAutomationFramework with features to measure the timing-behaviour of the measurement devices. With this extensions timing-changes between device-firmware-versions could be detected exactly. On the other hand it would also be possible to create a specific transformer-mapping to generate test-documentations. The test-results could also be automatically stored in the HP Quality Center instance, AVL is using. In combination with the generated documentation the storage of test-results would be significantly improved.

Acknowledgement

Research herein was funded by the Austrian Research Promotion Agency (FFG), program line “Trust in IT Systems”, project number 829583, TRUst via Failed FALsification of Complex Dependable Systems Using Automated Test Case Generation through Model Mutation (TRUFAL).
Bibliography


Bibliography


