Test Environment for DTUsat-2

Andreas Foldager(093285)
Henrik Hannemose(093262)

Kongens Lyngby 2012
IMM-BSc-2012-26

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-BSc-2012-26

Summary (English)

The goal of this thesis is to create a solution for testing DTUsat-2, which is a student-built satellite at DTU. In this thesis, requirements for such a test environment are determined and existing solutions for testing are studied.

The design for a test environment is presented, built upon the foundation of the concepts found in the existing systems. The designed system contains several general concepts, which enables the user to set up and combine elements to create advanced test scenarios.

Based on the design considerations, a system has been implemented, which has been tested against a simulator of the satellite. Basic functionality has been tested to ensure the test environment is able to communicate with a functional copy of the satellite.

Ultimately, we have ended up with an operational test environment, which can be used to perform functional testing of DTUsat-2. Additionally, we have proposed a number of possible future improvements to the test environment.

Summary (Danish)

Målet for denne rapport er at komme op med en løsning til at teste DTUsat-2, som er en satellit bygget af studerende på DTU. I denne rapport vil kravene til et sådant testmiljø blive klarlagt, og eksisterende løsninger vil blive nærstuderet.

Et design af et testmiljø vil blive præsenteret, byggende på de koncepter vi har fundet i de eksisterende systemer. Det designede system indeholder flere generelle koncepter, som sætter brugeren i stand til at opsætte og kombinere elementer til avancerede testscenarier.

Baseret på designovervejelserne har vi implementeret et system, som er blevet testet imod en simulator af satellitten. Grundlæggende funktionalitet er blevet testet for at sikre, at testmiljøet er i stand til at kommunikere med en funktionel kopi af satellitten.

I sidste ende er vi endt op med et virkende testmiljø, som kan bruges til at udføre funktionelle test af DTUsat-2. Yderligere har vi foreslået en række mulige fremtidige forbedringer til testmiljøet.

Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark under the supervision of Associate Professor Hans Henrik Løvengreen, in fulfillment of the requirements for acquiring a BSc in Software Technology.

The project period was from February 1st to July 1st, 2012. The goal of the project is to create a test environment for DTUsat-2.

Lyngby, 01-July-2012

Andreas Foldager
s093285

Henrik Hannemose
s093262

Acknowledgements

We would like to thank our supervisor, Hans Henrik Løvengreen, for his role in helping us understand the structure of DTUsat-2, providing competent feedback to our ideas as well as being available for frequent meetings during the design of the project.

Additionally, we would like to thank Søren Bøg for his assistance with providing access to a functional copy of the satellite.

Abbreviations and Acronyms

We have in this report used several abbreviations and acronyms, which we will list here:

MCC	Mission Control Center
MPS	Mission Planning System
PPL	Primary PayLoad
SA	SUT Adapter
SUT	System Under Test
TIF	Test and Integration Facility: The facility at DTU where the satellite is being developed and FlatSat resides.
TTCN-3	Testing and Test Control Notification Version 3

Contents

Chapter 1

Introduction

1.1 Motivation & purpose

When launching a satellite it is of the utmost importance that the onboard software systems can be trusted to behave as expected under a wide range of conditions. For this reason, it is necessary to make extensive testing of the software on the satellite before launch.

The aim of this study is to come up with a solution to facilitate testing of DTUsat-2, the second student-built satellite at DTU. The mission of the satellite is to track bird migration routes while having students at DTU involved in the development of the satellite. The motivation for finding a suitable solution for testing the software of the satellite is high, which stems from fact that radio contact never was established with the first DTUsat launched in 2003.

Overall, the goal is thus to come up with a test environment that is suitable for the task of performing system testing of DTUsat-2. There are several possible paths that can be taken to accomplish this goal, so initially we want to explore the possibility of adapting existing test environments to suit the given needs. Based on our findings, a choice should be made of whether to adapt existing solutions or to build a test environment from scratch. In the end, the goal is to deliver an operational test environment to the DTUsat-2 team such that team

members can define thorough tests to verify the behavior of the satellite.

1.2 Problem statement

More formally, the (given) project statement for this project is as follows:

> DTUsat-2 is the next DTU student satellite expected to be launched in 2012/2013. The scientific mission of the satellite is to collect information about bird migration (see `www.dtusat.dtu.dk`).
>
> As part of the final preparation of the satellite, an environment for automatic, functional testing of the satellite is needed. The test environment should be able to issue sequences of satellite commands and verify their responses.
>
> In this project, existing test environments should be studied. These include FIT and the tele-communication environment TTCN-3. On basis of this, either one of these should be adapted for DTUsat-2 or ideas of these implemented in a dedicated test environment for DTUsat-2.
>
> A dedicated test environment should be implemented in Java (or Scala) in order to interface to existing components.

1.3 Report structure

In this report we will give a general introduction to DTUsat-2 for the reader who is not familiar with the satellite, followed by a more technical introduction to the technical details about the satellite that are relevant to our project. Then we will conduct a survey of four existing test systems, followed by a conclusion on what we have learned from these existing systems and which concepts we plan to utilize in our own test environment.

After performing an analysis of the existing test environments, we present an overview of the design choices we have made, followed by an introduction to our overall design. Afterwards we describe each of these design choices in further detail.

A chapter about the implementation process follows. Interesting details about the implementation are highlighted, after which the results we have gathered

from testing the implementation are presented.

After discussing possible improvements and further developments to the test environment, we draw an overall conclusion on the current state of the test environment.

It should be noted that we from now on refer to DTUsat-2 simply as DTUsat.

Chapter 2

Analysis

2.1 Introduction to DTUsat

Before we dive into our analysis of the project, we will give a brief introduction to DTUsat, aimed at readers who are not already familiar with the satellite and its workings. This introduction will be followed up by some technical details about the satellite relevant to this project in section 2.2 on page 7.

2.1.1 Mission

The overall mission of the satellite is to track the migration patterns of birds, specifically cuckoos. The birds are to be equipped with small radio transmitters, whose signals will be picked up by the satellite.

The functions relevant to the mission of tracking birds are contained in the so-called PPL (Primary PayLoad) of the satellite. From the perspective of creating a test environment the mission is not that relevant, since communication with the satellite isn't influenced by the PPL. More information about the mission can be found in *DTUsat Mission Definition Document* [1].

2.1.2 High level overview

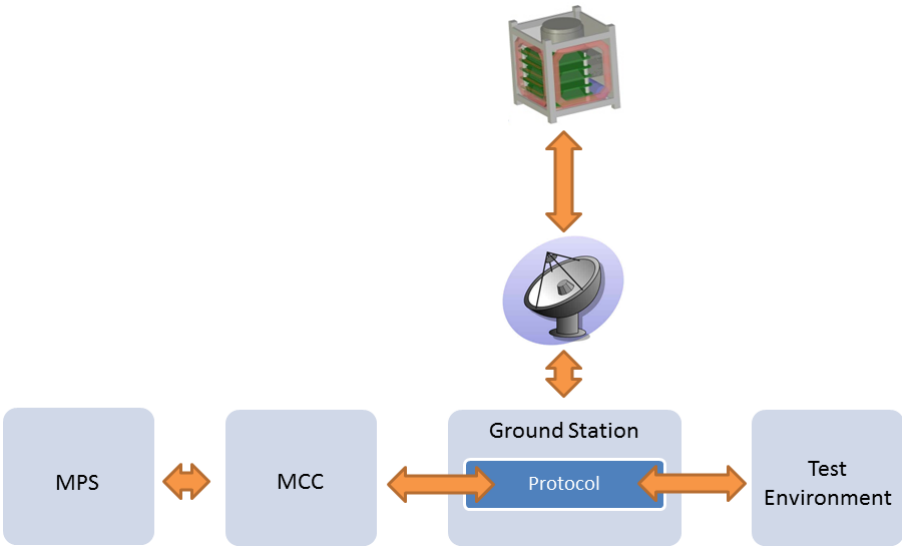

Figure 2.1: High level overview of DTUsat

As it can be seen in figure 2.1, the system consists of a number of components. The structure of the DTUsat project is such that the ground station is responsible for all communication with the satellite. In the diagram, MPS is the Mission Plannning System and MCC is the Mission Control Center. All ground components all communicate with the satellite by using a protocol contained in the ground station.

Since all communication is performed through the ground station, it would be reasonable to also have the test environment use this as a means of communication. Instead of interfacing with the whole ground station, the test environment should utilize the core component of the ground station responsible for communication with the satellite: the protocol. We will describe the protocol and the connection with DTUsat in section 2.2.5 on page 11.

2.1.3 Onboard software

On board the satellite there is software running to handle the operation of the satellite. The satellite is able to run in two modes of operation: Failsafe mode and nominal mode.

Failsafe mode is a minimal working edition of the satellite. If problems are detected during boot, the satellite can fall back to failsafe mode, where only the most critical functions deemed necessary to debug and repair any problems are present

Nominal mode is considered the normal mode of operation, where full functionality is loaded for the satellite and its components.

This project is not concerned with failsafe mode, which is considered robust through means such as code review and thorough testing. Instead, the aim is to conduct system testing of nominal mode where much more code is involved, increasing the risk of unforeseen errors occurring.

2.1.4 FlatSat

We won't have access to the final build of the satellite, since it has not yet been completed. Instead we can test against FlatSat, which is an installation of the satellite found in the TIF (Test and Integration Facility) at DTU. FlatSat is supposed to be functionally identical to the final build of the satellite and will thus be suitable for conducting testing against.

2.2 Technical details about DTUsat

In the following section we will discuss the technical details about DTUsat that are relevant for developing a test environment. All in all we aim to give a high level introduction to the parts of the DTUsat project required for this project.

2.2.1 Data

All data that will be sent to and received from the satellite is defined in XML files, describing all the information messages will have to contain, both for sending and receiving purposes. In total there exists four categories of messages that DTUsat can handle at the moment:

2.2.1.1 Message categories

Commands are used to send requests to the satellite, triggering a stream of responses which can be analyzed.

Confirmations are used to indicate the result of executing `commands` on the satellite.

Teledata is any data that is generated on the satellite as a response to a `command`, and can occur between the `command` itself and until a `confirmation` is generated.

Signals are used to control the communication between the ground and the satellite.

2.2.1.2 Types

A type defines the information associated with each category of messages. This includes the name, description, an id, parameter etc. The information is described in several XML files, one for each category of messages.

2.2.1.3 TypeMaps

In the DTUsat project there exists a static class, called `TypeMaps`, which can load all specified types from the given XML files, so they can be used locally. `TypeMaps` can be used to retrieve a specific type by using its name.

During start up one must make sure to load all data types into the system, which is done by calling `loadTypes()` on `TypeMaps`.

2.2.1.4 Parameters

The different types can contain multiple parameters, which have a name, description and additional information. Most importantly, a parameter also contains which type of input the given parameter accepts, such as `integer`, `enum`, `string` or `boolean`.

2.2.1.5 Arguments

Arguments are instantiations of parameters, giving them a specific value. For each parameter there is thus a corresponding argument with the same type.

2.2.2 Packets

As discussed in 2.2.1, DTUsat has different data types which can be sent to or received from the satellite. The data types are not sent directly to and from the satellite, however, but are encapsulated in a `packet`. In a `packet` both data and additional information is contained, as we will now explain:

2.2.2.1 Priority

A packet needs a priority for its execution when it has reached the satellite, describing the urgency of its execution. There are four priorities defined in the system: `high`, `normal`, `low` and `undef`. At the time of this writing, priorities are not used by the satellite.

2.2.2.2 Sequence number

Every packet has a sequence number. The ground and satellite software must each keep track of which sequence number they have reached, as the sequence numbers are used to identify `packets`. These sequence numbers must be unique during an execution of the test system.

2.2.2.3 Parameter-Argument mapping

The data type stored in a packet may have some parameters on it, which need to be instantiated through arguments. Parameters and arguments are not combined in a single object, but are mapped together utilizing two arrays of equal size. The mapping is done implicitly using positions in the arrays.

2.2.3 Sending

As described in 2.2.1 on page 7, we have four kinds of data that DTUsat can handle. The tester only needs access to one of them, namely the `command` data type. The `command` will cause the satellite to generate a stream of responses that should be analyzed.

Besides the `command` data type, the system should be able to send `signals` as well in order to set up the satellite to its required settings. The signals will mainly be used to ask the satellite to send all packets it generates back to us.

2.2.3.1 Execution time

The `command` type needs some extra information in order to be executed, namely when it should be run on the satellite. We will ignore this in the test system, since we want everything to be executed as soon as possible.

2.2.3.2 Modules

When sending `commands` to the satellite, one also needs to specify to which module the `command` should be sent. This is caused by the fact that the satellite is split up into modules, which each has its own unique purpose. The module could e.g. be the camera onboard or the flash memory system.

In the XML file containing all `commands` that the system recognizes, every `command` has a list of valid modules for the `command`. In case no modules have been declared, all modules will accept the `command`

2.2.4 Receiving

When receiving data from the satellite, the packet will contain the sequence number of the packet which was sent to the satellite, which can be used to map the response to the originating `command`. In general, replies from the satellite will always be either `confirmation`, `teledata` or `signals`.

2.2.5 Connection to DTUsat

Our overall goal is to test the functionality of the satellite itself. Instead of connecting directly to the satellite ourselves, we will instead use a component from the existing ground software:

2.2.5.1 Protocol

As mentioned, there is a component inside the ground station called the protocol. This protocol serves as method of communication and is used by all ground components that need to communicate with the satellite.

All communication from ground to the satellite can be assumed to utilize this protocol, and it is thus acceptable to test the functionality of the satellite through this component.

2.2.5.2 Requesting packets

Before any information will be received from the satellite, it is necessary to request to receive messages from the satellite. This is done by sending the signal `SIG_REQUEST_PACKETS`, which means everything will be reported back. By calling this whenever we establish a connection with the satellite, we ensure that we will receive all the information intended when running the tests.

To stop receiving any more packets, the following command can be issued to the satellite: `SIG_PACKETS_STOP_AND_RESET`.

2.3 Use cases

In the following we will show some of the use cases the test system should support. The use cases will be a high level description of how the system will behave, so any system specific details will be left out of the descriptions. This means that the indicated flow of events might vary from the final implementation of the system.

2.3.1 Use case: Simple test execution

The following use case symbolizes the process of executing a test whose purpose is to send a single `command` to a specific `module` and see if the right responses are coming back.

Pre-condition	A connection to the satellite has been established
Flow of events	• The user selects a `command` to be sent to a specific `module` • The user specifies what should be received • Results are reported to the user
Post condition	A finished test, with a result which can be analyzed upon.

2.3.2 Use case: Initializing the system to a specific state

The following use case shows the process of executing a test after the system has attained a specific state.

Pre-condition	A connection to the satellite has been established
Flow of events	• The user sets a sequence of commands to be executed to bring the system into a specific state. • A test that should be performed after the specific state has been reached is specified.
Post condition	A finished test, with a result for the particular state the system had reached.

2.3.3 Use case: Collection of tests

The user should be able to combine several tests into one collection, which this use case illustrates.

Pre-condition	A connection to the satellite has been established
Flow of events	• The user creates a collection of several `tests`, each specifying which `command` and `module` they will interact with and what they expect to get back in response. • The user specifies whether the collection should be executed sequentially or concurrently.
Post condition	A sequential or concurrent collection will be ready for execution

2.3.4 Use case: Use of parameterized tests

The user should be able to have parameterized tests, so the user won't have to specify a test for each specific value he wants to test for.

Pre-condition	Test with `command` and `module` specified
Flow of events	• The user specifies some `variables` on the test which can be used for specifying values for the `command`'s `parameters`. They can also be used for setting expected return values. • Specific values for the variables are set by the user.
Post condition	A parameterized test, which can get different `arguments` for the `parameters`

2.3.5 Use case: Different result paths

The user should be able to specify different paths that can be evaluated. This is closely related to parameterized tests, as these may state that another result should be expected.

Pre-condition	A test with `command` and `modules` specified

Flow of events	• The user sets up different series of valid replies, that might come back as a response to the given `command` • At each point in the given series, one may choose specific conditions that must be true in order for the system to continue on the given series.
Post condition	A test with different result paths depending on the situation.

2.4 Requirements

After getting an impression of some of the different use cases there are for the test environment, we have listed some requirements for the system. This list of requirements has expanded gradually during the course of the project when more requirements were discovered as a result of our increased knowledge of the DTUsat project, combined with input from Hans Henrik Løvengreen.

2.4.1 Functional requirements

Testing The most basic requirement for our test environment is that we can actually send a command and receive answers for the given command. From the received answers we should be able to conclude whether it has failed or passed.

Combining tests It should be possible to gather a collection of tests in order to create a scenario.

Sequential execution Combined tests should be able to execute sequentially, so different tests don't interfere with each other.

Concurrent execution Combined tests should also be able to execute the encapsulated tests concurrently. This should be possible so testers can see how the system responds to interleaved `commands`.

Templates One should be able to define tests which can be reused other places in the system.

Parameterized tests It should be possible to define tests which have variables that can be initialized, giving rise to more dynamic testing.

Specifying expected responses One should be able to specify which responses are expected, so the system can validate against the actual received values.

Result The system should give detailed results that state what went wrong, and not just whether it failed or passed. By stating what went wrong, one can easier find the source which caused the given test to fail.

Timeout It should be possible to set a test to time out at some point, in order to avoid waiting too long for a test to complete.

Delay between tests One should be able to specify a delay between tests (specifically, an example of having up to 48 hours of delay between sending two messages has been mentioned)

Intervals The system should provide the possibility of setting an expected interval as a result, instead of an exact value.

Execution with a sequence of values One should be able to execute the same test, or a collection of tests, multiple times, with the use of a sequence of values.

Automated execution of tests The system should be able to be run programmatically, e.g. for testing of nightly builds.

Manual execution of tests It should be possible to manually execute tests and have a visual indication of the results

GUI A GUI should be provided, for easier set up of tests. It should support analyzing the result that a scenario delivers. Additionally it should support saving and loading of the created scenarios.

Adapter We need an adapter to facilitate communication with the satellite.

2.4.2 Non-functional requirements

Defined interface We need a defined interface between the adapter and the test environment, so we are independent of changes on the satellite. By having this interface, we will only need to change the communication layer between the adapter and SUT in case changes happen on the SUT.

Performance requirements It should be possible to have the test system run on a standard machine

Choice of programming language(s) The other parts of DTUsat have been made in Java and C, utilizing Eclipse as an IDE. It is preferable that the test system fits into this environment.

2.5 Survey of existing test environments

After listing requirements for the system, we find it relevant to familiarize ourselves with existing test environments. In doing this, we will be aware of general concepts that the systems have in common, as well as any shortcomings which would need to be taken care of for the system to be used for our purposes.

In general, we have tried to take a broad look at which systems already exist. We have decided to focus on open source technologies. This is done, since we find that open source meshes well with the current technologies used in DTUsat, where no proprietary software is used (to our knowledge).

Our initial research made us focus on different existing products for testing, which each come from a different perspective. These test environments are:

TTCN-3 A system designed for conformance testing of communication systems and standards, which has been used for e.g. WiMAX.

TestNG A system building upon unit testing concepts but expanded to handle more than just unit testing.

JSystem A system including a GUI to write and manage automated system tests.

Fit A system with a focus on acceptance testing from a customer's point of view.

We will now line up some of the key features of each of these systems. Ultimately, we have decided to produce our own testing environment, and we will thus also list some of the features from the surveyed test environments we have used as sources inspiration.

2.6 Survey of TTCN-3

2.6.1 Overview

TTCN-3 is a language that was defined to perform testing. That is why it would be interesting to see if it could be incorporated into our solution. As a source for our research on TTCN-3 we have primarily used the book: *An introduction to TTCN-3* [2].

Originally TTCN was designed to perform communication testing, in order to test protocols. This can be seen as it has been used to test several communication protocols, such as WiMAX, IPv6 and more. Even though previous editions of TTCN were primarily designed for testing communication systems, TTCN-3 has upgraded the language with new features, so it allows for more than just communication based testing.

Like any other programming language, TTCN-3 needs a compiler in order to execute. This is one of the few downsides of TTCN-3, as there doesn't exist that many compilers and next to none which are free. Besides the compiler issue, the language itself is hard to get started on, as there aren't many examples on how to create a test system using TTCN-3.

2.6.1.1 Structure

The structure of TTCN-3 can be seen in figure 2.2 on the following page.

As it can be seen, the system has been split into several components, each with a unique purpose.

TTCN-3 Tools TTCN-3 Tools are the backbone of TTCN-3, which are responsible for actually compiling and executing the TTCN-3 code. TTCN-3 Tools consist of an execution environment and a compiler as well as possible default implementations of the surrounding components. The execution environment is responsible for actually running the compiled code, while the compiler translates TTCN-3 syntax into executable code.

Most tools are, as we will later discuss, proprietary. TTCN-3 Tools consist of the components seen in figure 2.2 on the next page.

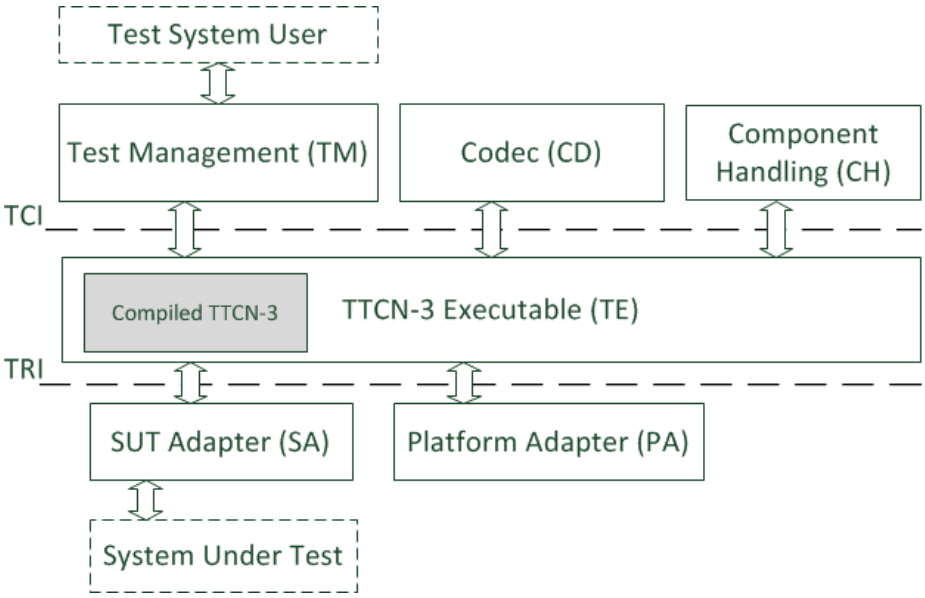

Figure 2.2: Structure of TTCN-3. Figure adapted from [2]

TCI The `TCI` is responsible for controlling the flow of the test system, as it should allow the user to set up which tests are run and in which order. It will be responsible for the communication with the following components:

Codec is used to code or decode any messages, so they can be understood by both the SUT and TTCN-3.

Test Management will grant the user control of which tests are to be run and in which order.

Component handling will enable the user to specify how components should be created and implemented in the given test system.

TRI The `TRI` will be responsible for the communication with the SUT and ensuring we have the same system behavior as the SUT. This is ensured by implementing the following components, which can be reached through the interface.

SUT Adapter is used to map any communication onto the actual system. This means that any test created won't have any knowledge of what it is actually communicating with, since it goes through the `SUT Adapter` for the given SUT.

Platform Adapter is used to get hold of platform dependent information. This could for example be a timer, which is implemented differently depending on the platform.

Besides these interfaces, the TTCN-3 code itself will be executed by the `TTCN-3 Executable`, which utilizes the two interfaces in order to send and receive messages from the SUT, and concludes if they succeed or fail depending on the responses.

2.6.1.2 Example

Due to the unique nature of TTCN-3 we have decided to show the following example of the syntax and how it is executed, in which we assume all components are set up and working.

Message types To standardize the messages to be sent and received, we can define `records` for later use, as seen in figure 2.3.

```
type record DNSMessage {
        Identification  identification,
        MessageKind     messageKind,
        Question        question,
        Answer          answer          optional
}
```

Figure 2.3: A simple definition of a message in TTCN-3. Figure reprinted from [2]

The types `Identification`, `MessageKind`, `Question` and `Answer` used in the example in figure 2.3 are defined as seen in figure 2.4

```
type integer    Identification( 0..65535 );
type enumerated MessageKind {e_Question, e_Answer};
```

Figure 2.4: Definitions in TTCN-3. Figure reprinted from [2]

Templates After the creation of the message type, one can use it to create several templates, which can be used when sending and receiving messages. An example of a template can be seen in figure 2.5.

```
template DNSMessage a_DNSQuestion(Identification p_id
                                  Question p_question ) := {
  identification := p_id,
  messageKind    := e_Question,
  question       := p_question,
  answer         := omit
}
```

Figure 2.5: Declaring a template in TTCN-3. Figure reprinted from [2]

Figure 2.5 shows a parameterized template. However a template doesn't have to parameterized, but can have specific value set as well.

Sending and Receiving After defining templates, we can actually create `testcases` which use them in order to communicate with a system. However, in order to do this, we need a `port`, which can send them out, and a `component` which the `testcase` can run on. A `port` defines which messages are able to go in and/or out through the port, while the `component` specifies which ports it will use. As example of defining a `component` and a `port` can be seen in figure 2.6.

```
type port DNSPort message {
  inout DNSMessage
}

type component DNSClient {
  port DNSPort serverPort
}
```

Figure 2.6: Definition of a port and component in TTCN-3. Figure reprinted from [2]

Finally, a test can be defined, as seen in figure 2.7 on the next page. It is assumed that `a_DNSAnswer` has been defined.

```
testcase ExampleTestResolveNokia1() runs on DNSClient {
  serverPort.send( a_DNSQuestion ( 12345, "www.nokia.com" ));
  serverPort.receive( a_DNSAnswer ( 12345, "172.21.56.98 ));
  setverdict( pass );
  stop;
}
```

Figure 2.7: Simple test case in TTCN-3. Figure reprinted from [2]

This test case seen in figure 2.7 will simply send a DNS request for `www.nokia.com` and expect to get `172.21.56.98` back. However, the above can potentially deadlock, since we are not guaranteed to get the exact IP-address back. This is why TTCN-3 allows alternating scenarios, depending on the result. We will show an example of alternate replies in 2.6.2.1.

2.6.2 Nice features

During the research process we encountered several features, which we consider to be good contributions to any test system.

2.6.2.1 Timeout

TTCN-3 allows timeouts by default, which is needed in all communication based systems, since the connection can break down, resulting in no messages being received.

2.6.2.2 Result

TTCN-3 introduces new types of results for test cases. These are as follows, in order of how bad they are considered to be:

$$\texttt{none} > \texttt{pass} > \texttt{inconc} > \texttt{fail} > \texttt{error}$$

Here, `error` symbolizes that an error occurs during runtime, and can't be set by a test itself. The additional result types are a good contribution, since they give the user a more concrete idea about went wrong.

2.6.2.3 Alternative replies

In TTCN-3, a tester has the possibility of choosing different paths depending on the result that comes back from the SUT. This is done with the concept of `alt` statements, which can be combined with guards to make more advanced control flow.

Guards evaluate to a boolean value that tells whether a certain branch of evaluation should be followed. These guards can utilize variables in their expressions. This generalizes test cases even more, since test cases can contain multiple result paths, thus testing more aspects of the SUT.

Example The use of guards and alternative replies is easy to set up in a test, as it can be seen from figure 2.8, which is an extension of the example from figure 2.7 on the previous page.

```
testcase ExampleTestResolveNokia1() runs on DNSClient {
  serverPort.send( a_DNSQuestion ( 12345, "www.nokia.com" ));
  alt {
        [x>2] serverPort.receive
                ( a_DNSAnswer ( 12345, "172.21.56.98 ))
                { setverdict( pass ) };
        [x<0] serverPort.receive
                { setverdict( fail ) };
        [else]
                { setverdict( fail ) };
  };
}
```

Figure 2.8: Simple test case in TTCN-3 with the possibility of getting different results depending on which messages arrive to the system. Figure reprinted from [2]

The above example shows a method of how to resolve the potential deadlock that could occur in the example from figure 2.7 on the preceding page. In the example we expect to get an answer with a specific IP address in case x has a value of 2 or higher. Otherwise, the testcase is considered to have failed.

2.6.2.4 Concurrency

TTCN-3 allows for concurrent testing, which needs to be used when testing DTUsat, since there are different sub-components which can be tested at the same time. Alternatively we would be able to stress test a single sub-component. The way concurrency is handled in TTCN-3 is by having multiple `components` running inside a test case. These `components` can communicate with the same, or different, ports.

2.6.2.5 Intervals

One can with ease define intervals in TTCN-3, meaning test cases can be even more dynamic as parameters can be used to define an interval which a return value should be located in.

2.6.2.6 Optional fields

TTCN-3 allows setting the expected value of a return message to be anything or nothing at all. This is nice, since it allows the tester to only test specific values in the received message, while he doesn't have to worry about the optional values.

2.6.3 Issues

Even though TTCN-3 seems to include all the features we need for our test environment, it has some issues which concern us:

Compilers The biggest problem with TTCN-3, as we see it, is the fact that most compilers are commercial. However, there are a couple of open-source compilers: LoongTesting[3] and BBT[4].

LoongTesting was not tested, due to problems with actually downloading the software from the only server in China. BBT was tested, but did not live up to our expectations, since it only supports basic TTCN-3 syntax. So even though there are some free compilers, these lacked the quality we would like from a compiler. All in all we could have chosen between paying for an excellent compiler or using a free compiler, which might lack functionality. In our case, we would have to choose the open-source compiler, meaning we could risk hitting a

wall at some point, in which the features we require could not be implemented due to the compiler not being advanced enough.

Lack of documentation Even though there is a lot of documentation of the syntax of TTCN-3, we feel it is a bit hard to get a grip of, since there are not that many examples on the usage. Many compiler manufacturers deliver courses in TTCN-3, which is how one would normally gain a complete understanding of the system.

2.7 Survey of TestNG

2.7.1 Overview

TestNG is a system created on the same concepts as JUnit, but adds some features the author thought was missing in JUnit. TestNG is intended to cover more areas of testing than just unit testing. Further details about TestNG can be found on the TestNG website[5].

In a top-down fashion, TestNG consists of four different layers. Each layer can contain a reference to any number of items from the next level:

- Suite: An XML-file containing a number of tests nested in a `<suite>` tag.
- Test: A Test is identified by a `<test>` tag and contains one or more TestNG classes.
- TestNG class: Identified by a `<class>` tag. Contains test methods.
- Test method: Tests annotated by `@Test` tags, JUnit-style.

In essence, if one already has knowledge about JUnit, TestNG should feel very familiar. TestNG does contain some extra features, though, which we will highlight in the following.

2.7.2 Nice features and concepts

2.7.2.1 Grouping

TestNG supports grouping of test methods (and suites) into groups. This eases the execution of only certain parts of tests, which could for example be used for running a subset of nightly tests each time one checks in code to a repository. Additionally, it is possible to create groups of groups.

2.7.2.2 Dependencies

It is possible to specify that a certain test depends on other test methods or entire groups. There are two kinds of dependencies, hard and soft.

A hard dependency specifies that a test depends on the other test or group of tests succeeding: If not, the test is skipped. This helps avoid having a lot of tests fail if a prerequisite is not met and helps pinpoint the culprit.

A soft dependency, on the other hand, doesn't depend on the result of the dependency. Instead, it simply requires that the dependency has finished executing before the current test is executed.

2.7.2.3 Rerunning of failed tests

TestNG has support for outputting an XML-file with only failed tests (and their dependencies). In this way it is easy to only execute the subset of tests that are failing when trying to correct any issues in the SUT.

2.7.2.4 Factories

Another concept in TestNG is factories. This helps automate the instantiation of a test case with a given sequence of arguments. In this way, one can avoid manually instantiating the test multiple times with the added overhead and complexity it brings.

2.7.2.5 Output of results

TestNG supplies interfaces for listeners and reporters. Methods on these interfaces are called at various stages during execution of tests, such as when a test starts, fails or passes. By implementing these interfaces, the user is given a high level of flexibility by choosing a presentation format at his own liking.

2.7.2.6 Parallelism

TestNG supports execution of methods, tests or classes concurrently while still satisfying the set dependencies.

2.7.3 Issues

2.7.3.1 Parallelism not geared for asynchronous messages

We like the thought of being able to execute tests concurrently, since the communication with DTUsat is asynchronous. In this way tests are able to run independently of each other. The problem with having each test send out their own messages it that, out of the box, there is no way to make sure received messages from the satellite find their way back to the right test. To accomplish this, one would have to somehow add in another layer to the system to keep track of which sequence numbers correspond to which tests and route messages accordingly.

2.8 Survey of JSystem

2.8.1 Overview

JSystem is a testing system consisting of many components. Information about JSystem has been gathered from the JSystem website[6]. JSystem has undergone a history of initially being open sourced, then being developed as a proprietary software for a period of years and then subsequently being open sourced again.

JSystem is partitioned into four separate layers, roughly corresponding to the separations made in the other test systems we have looked at. These separations

have been made such that each layer corresponds to a role for the users concerned with testing. The layers (called Automation Layers) can be seen in figure 2.9

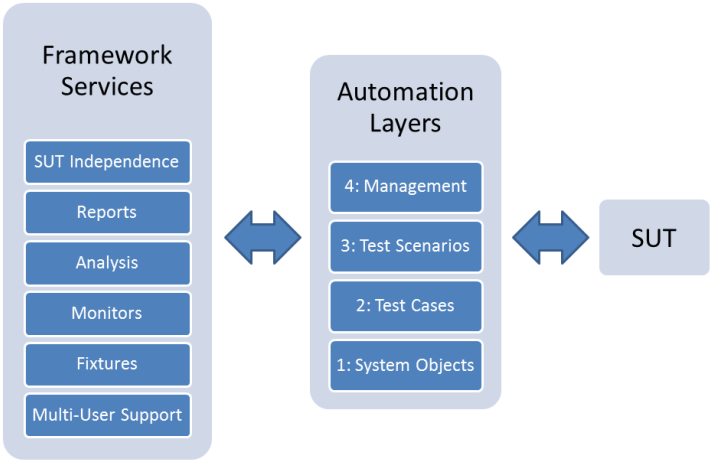

Figure 2.9: An overview of JSystem components. Figure adapted from [7]

As it can be seen, JSystem has been developed with extensibility in mind, where the framework supports expanding the scope of testing to encompass many users and components.

We will now give an overview of these four automation layers, starting from the lowest level of abstraction:

2.8.2 Automation Layers

2.8.2.1 1: System Objects

The System Objects automation layer is at the lowest level in JSystem. This level is managed by the Java developer role. Here, the developer specifies a system object that implements an interface supporting basic operations such as `init()` and `close()`. All interactions between the test system and the SUT pass through this layer. This ensures a clean separation between testing and the SUT and corresponds to the concept of a `SUT Adapter` in TTCN-3.

The System Object is even more customizable in that it is able to load parameters from an XML-file. In this way it can load information needed to connect to a specific device or SUT.

2.8.2.2 Layer 2: Test Cases

The Test Case layer is managed by the Test developer role. This layer basically consists of regular unit tests specified with JUnit, which is a familiar concept for many programmers.

To define tests, the test developer thus creates a series of JUnit tests annotated with the `@Test`-notation. All calls to the SUT are made through the System Object.

One of the advantages of using JUnit at this layer is that tests are then able to be executed normally as JUnit tests before they are hooked into the larger JSystem framework.

JSystem also extends the normal JUnit functionality with the ability to do some special parsing: If there in the class exists corresponding getter and setter methods, e.g. `getPingDestination` and `setPingDestination`, they are automatically interpreted as being parameters for the test which can be set in the Test Scenario layer.

2.8.2.3 Layer 3: Test Scenarios

Once Test Cases have been created, they can be grouped into larger collections called Test Scenarios. It is also at this level that one sets specific arguments for the parameters the test cases need. This layer is handled by the Quality Assurance engineer role.

Test Scenarios are saved as XML-files and can easily be created and maintained by the Management layer.

2.8.2.4 Layer 4: Management

The Management layer of JSystem consists of a GUI. A screenshot of the GUI can be seen in figure 2.10 on the facing page. In the right hand side of the GUI there is a tree view of all the tests defined on the Test Case layer. They

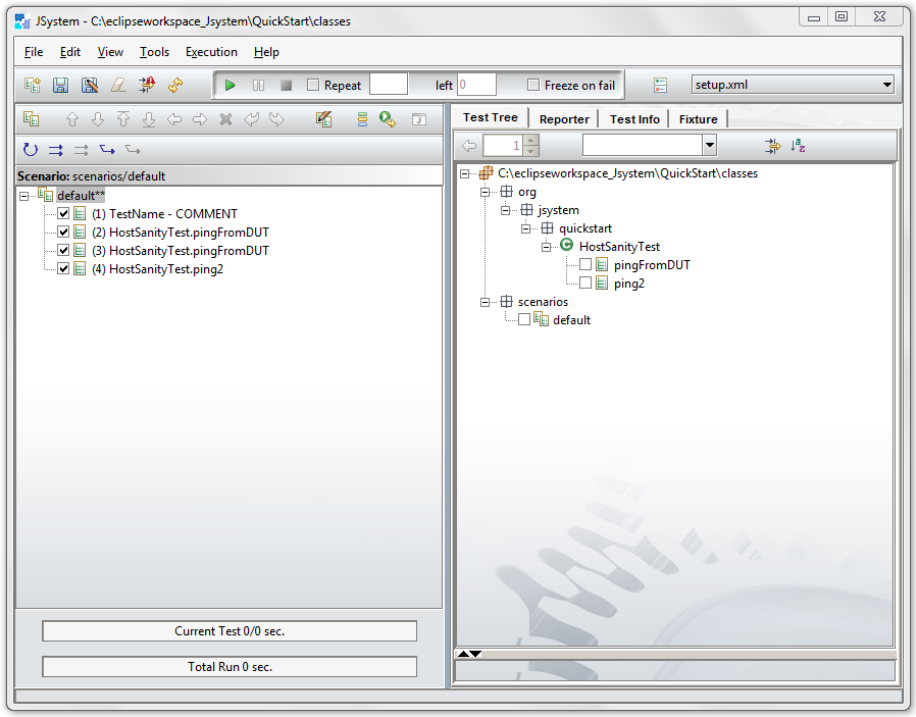

Figure 2.10: The JSystem Management GUI

can then, one by one, be added to the current scenario, which can be seen on the left hand side of the GUI. After they have been added to a scenario it is then possible to set the specific arguments that are to be used for that specific instantiation of the test case.

From the Management level it is also possible to execute scenarios and get visual feedback on the results of the execution. This is made possible by the Reports framework service, which we will briefly explain later.

2.8.3 Framework Services

The four Automation Layers are supported by a number of modular Framework Services, which have supporting roles in making the system work while keeping a very modular approach.

As an example, the SUT Independence Framework Service works in collaboration with the System Objects Automation Layer to ensure a complete separation between tests and the SUT. Likewise, the Multi-User Support Framework Service is a fairly recent addition to JSystem that takes the framework from being a standalone program to being a true client/server-based solution in which the GUI and initiation of execution is separate from the agents that are responsible for executing the tests.

We will now give an overview of the remaining services:

2.8.3.1 Analysis

The Analysis service is responsible for determining the outcome of test executions. When writing the JUnit tests one thus delegates the actual analysis of the result of the test to the Analysis service which, after analyzing the results, sends the results of the analysis to the Reports service.

2.8.3.2 Reports

The Reports service gathers results from analysis and presents them to the user. The usual method of output is to HTML reports, which are customizable with graphs and detailed results.

In addition to showing the results of the most recent runs, the Reports server also supports storing data which enables the user to compare current results to historical data.

2.8.3.3 Monitors

Another service is the concept of Monitors, which are separate threads responsible for monitoring the state of the SUT while tests are being executed. A monitor could for example be used for continuously checking that the connection to the SUT is up and alive.

2.8.3.4 Fixtures

To ensure that test methods and configuration of the SUT are kept separately, the concept of Fixtures is used. They can be run before tests or scenarios are executed to make sure the SUT is in the desired state. They thus have `setup` and `teardown` code to ensure the needed preconditions are met. Additionally, they also have `failteardown`-methods to handle failed tests.

2.8.4 Nice features and concepts

2.8.4.1 Modularity

As it can be seen from our overview of the different components of JSystem, it is highly modularized. While this approach might make it a bit more difficult for users to initially get familiarized with the system, it greatly enhances the possible uses for the system and the ease of extending or changing certain parts of the system.

2.8.4.2 GUI

As mentioned in 2.8.2.4 on page 28, JSystem comes with a fairly mature GUI.

We find the GUI fairly intuitive to use and have decided to draw inspiration from the JSystem GUI when creating our own GUI. Specifically, we like the visual structure of having tests on the left hand side as well as being able to specify

arguments from the GUI. Additionally, we also want to be able to execute tests from our own GUI in a way similar to the JSystem-implementation.

2.8.4.3 XML-files

JSystem utilizes an XML file to define test scenarios. This XML file is not easily human-readable, though, since the XML file refers to a .properties-file, where specific values are set. That said, we like the concept of having an XML-file, where one is able to specify and change tests.

2.8.5 Issues

Ultimately, we didn't end up choosing to use JSystem. Even though the system is highly extensible and modular, it is clear that it has been built upon the unit testing concepts of JUnit. It is thus still most geared towards making tests of individual methods that return values to the test after completing, and not so much testing asynchronous communication, where replies to commands might not be generated at once.

2.9 Survey of Fit

Fit is an abbreviation of Framework for Integrated Test. Our original problem description calls for surveying this framework to see if it could be usable in relation to DTUsat.

2.9.0.1 Fit from a customer's point of view

Fit comes from the perspective of making testing very easy for a customer who has no knowledge of software development. It aims at helping the customer list the business rules that software should implement. The goal for the customer is to condense the business rules into a tabular format, which is entered into a program able to generate HTML tables, such as Microsoft Word or Excel. An example of this format can be seen in figure 2.11 on the facing page. Since Fit is very focused on the customer, it is primarily used for acceptance testing.

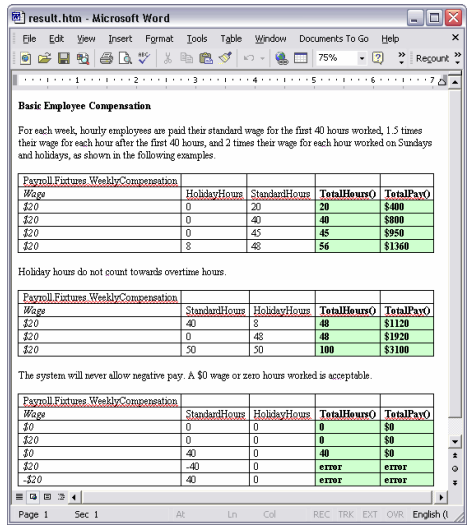

Figure 2.11: Specifying tests with Fit. Figure adapted from [8]

2.9.0.2 Fit from a programmer's point of view

Fit is responsible for parsing the tables provided by the customer. These tables are then passed to so-called `fixtures` that are written by the developer. These `fixtures` are then responsible for calling methods on the SUT. Different kinds of base classes are provided depending on which kind of logic one is looking to test.

2.9.1 Issues

Fit has not been updated since 2005, where version 1.0 was released. It thus seems like continued development of Fit is not happening. Instead, another project, FitNesse, started out by building on top of Fit. Later, FitNesse forked Fit, and it seems like more development has been happening in this fork than in the original code base. We haven't studied FitNesse in depth, but it might be an option to look at.

Overall, we find that the focus Fit has on acceptance testing by customers who are unaware of software development doesn't suit our project: The intended testers for DTUsat are highly aware of software development projects. Additionally, we find the low modularity of Fit to be a disadvantage if we were to

try to expand on this framework in order to support testing of communication with DTUsat.

2.9.2 Comparison of terminology across test systems

Since the systems we have looked at each have different layers of abstraction, and use different terminology for some of the layers, we will now present a brief comparison of the terminology used in the different systems. We have left Fit out of the comparison, since we don't find that it is relevant to compare it to the other systems. In the following table, we have only listed concepts that exist in at least two of the systems and are fairly similar:

TTCN-3	**TestNG**	**JSystem**
Test-case A single test, e.g. 1 DNS-query	**Test** <test> tag containing 1 or more TestNG classes **TestNG Class** <class> tag containing 1 or more TestNG methods **Test Method** Java method annotated by @Test	**Test** A single JUnit-test
Test Suite Collection of simple tests	**Suite** XML-file containing 1 or more tests	**Test Scenario** Contains tests grouped in a hierarchical manner
Test Management Control part to specify the order of tests in a test suite (test campaigns)		**Management** GUI. Ability to group tests into scenarios, set parameters for tests and execute them.
SUT Adapter Ties tests together with the SUT.		**System Object** Ties tests together with the SUT.
Verdict Defines the outcome of test cases.	**Assert** Checks that certain conditions are met on test cases.	**Analysis** Analyses operation results

TTCN-3	TestNG	JSystem
	Listeners+reporters Notified of results to output to various formats.	**Reports** Services for outputting results to users.

As it can be seen from the table, the different systems have roughly corresponding concepts and terminology.

2.10 Conclusion on surveys

After having surveyed the four different test environments mentioned above, we considered our options: We could choose to look for more test environments that might fit our requirements, we could choose to try to modify one of the existing environments to suit our needs or we could decide to make our own system.

We decided not to invest more time in looking for more existing test environments that might be useful to us, since we deemed it would use up too much time without guaranteeing a usable result.

Likewise, we chose not to start modifying one of the existing environments since it would require a deep level of understanding of the existing code base before we would be able to judge which areas to change (and whether adding and changing code to suit our needs would even be possible).

In conclusion, we ended up deciding to make our own system. In this process we have drawn upon the concepts and features that are used in the existing systems. This has the advantage that we use terminology and concepts that already exist in different projects while we at the same time customize the system to fit the needs of testing DTUsat.

2.10.1 Sources of inspiration

In making our own test environment, we have decided to find inspiration in different concepts from the surveyed test environments. We have thus decided to have concepts similar to the following:

GUI The GUI in JSystem is nice to use and has provided us with an idea of how a GUI for a test environment could be structured.

SUT Adapter We like the separation between tests and the SUT, which is provided by a `SUT Adapter` in TTCN-3 and a `System Object` in JSystem. We will thus utilize a similar layer of separation in our own test environment to improve reusability of the test environment and to be able to easily cope with changes in the satellite.

Results In TTCN-3 there are different kinds of states for results, such as `none`, `pass` and `inconc`. This helps in determining the status of tests and is more helpful than a simple true/false result.

Templates TTCN-3 uses the concept of templates, which are general messages that are subsequently used in actual tests. We also want to have a similar concept of reusablity in specifying smaller actions that can be combined into larger tests.

Chapter 3

Design

3.1 Design choices

In the following we will take a look at how the protocol for DTUsat works, and analyze how some of the basic requirements can be incorporated into our design. For a list of the requirements, see 2.4.1 on page 14.

3.1.1 Protocol

Figure 3.1 on the following page shows an overview of how the protocol for DTUsat works. It can be seen that we will send `Commands`, which will produce responses from the satellite in the form of either `Teledata` or `Confirmation` messages. It can also be seen that the response sequence will always end with a `Confirmation`. The received messages should somehow be analyzed to conclude on the overall sequence.

As seen in figure 3.1 on the next page, one can have multiple sequences running at the same time.

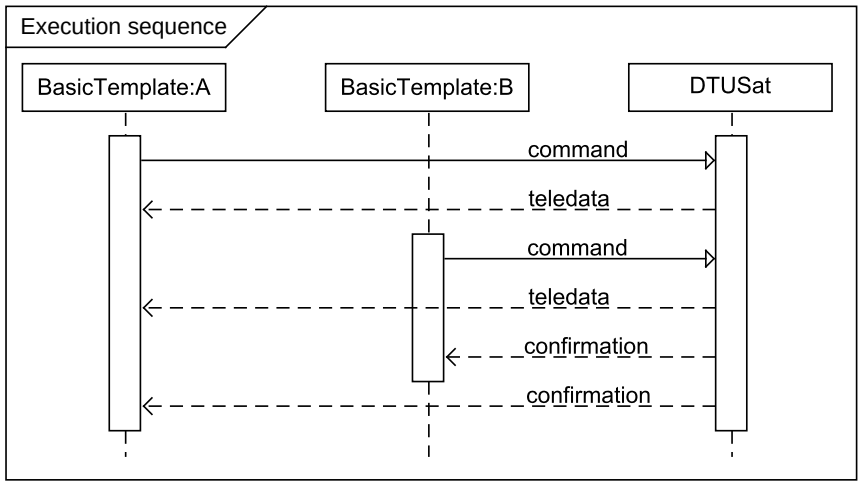

Figure 3.1: The process of sending and receiving packets from DTUsat of the given types.

3.1.2 BasicTemplate

The most basic requirement for our system is to perform tests on the different `Commands` found in DTUsat. We thus need a way to interpret the responses generated by the different `Commands`, as seen in figure 3.1, in which two `Commands` are generating responses. We have chosen to have `BasicTemplates` be responsible for the communication process for a single `Command`. Thus, a `BasicTemplate` is the most basic building block of our system. It is responsible for sending commands and for keeping track of the responses to the sent command.

The functionality of a `BasicTemplate` is similar to the concept of templates from TTCN-3. As described in 2.6.1.2 on page 19, templates from TTCN-3 have the purpose of defining standard messages to be sent and received. This means that our `BasicTemplate` can be seen as an extended edition of templates from TTCN-3, in which one can describe a sequence of messages consisting of a `Command` to be sent followed by `Teledata` and finished with a `Confirmation` to be received, instead of just a single message. To further build upon the concepts of templates from TTCN-3, `BasicTemplates` will be parameterized. `BasicTemplates` thus have variables, which can be used by different parts of the `BasicTemplate`, such as for defining arguments for the parameters of the command.

3.1.3 *Test*

In our system, a *Test* will be responsible for a single BasicTemplate, meaning it must instantiate any variables found on the BasicTemplate. The execution shown in figure 3.1 on the facing page could thus be viewed as two *Tests* running a BasicTemplate each.

We have chosen to not be able to directly execute BasicTemplates. Instead, one has to make a *Test* containing a BasicTemplate. We find that this enables us to have a cleaner separation between BasicTemplates, which are concerned with sending and receiving commands, and *Tests*, which are concerned with executing BasicTemplates.

As seen in 2.9.2 on page 34, we have based our ideas for the *Tests* on the analyzed systems, which all use the term *Test* for a similar element.

3.1.3.1 Scenario

In order to increase the functionality of the system, we have chosen to enable a *Test* to be able to contain references to other *Tests*. This is inspired by the Suites or Scenarios known from the other systems, whose purposes are to combine several *Tests* into a single execution. By having this, one could consider the example shown in figure 3.1 on the facing page to be one scenario, which at the bottom of the hierarchy has two BasicTemplates.

The purpose of the scenario in our system is not just to have a collection of *Tests*, but also to have different *Templates* that can be referenced in the Scenario. The *Templates* in a Scenario are called RootTemplates. Likewise, there is a single test acting as the root of the test hierarchy, called RootTest.

3.1.3.2 Interface

We have chosen to have both *Tests* and *Templates* implement some of the same methods for fulfilling the following goals:

Sending Both *Tests* and *Templates* should support sending. This could be by done by calling Send on the *Tests* or *Templates* under it or, in case of a BasicTemplate, by sending the actual command.

Receiving After the *Tests* or *Templates* have sent, it should be possible to receive messages. This is done by calling `Receive` on the *Tests* or *Templates* placed under the current object or, in case it is a `BasicTemplate`, by listening for responses and concluding upon these.

3.1.4 Validation

The other systems we have analyzed had the concepts of Verdicts (TTCN-3), Assert (TestNG) and Analysis (JSystem). We need a similar term for the component doing the actual validation. We have chosen to call these `Validators`. Every `Validator` has the option of referencing other `Validators`. In this way it is possible to create a tree of `Validators`. This tree of `Validators` will work as a nondeterministic finite automaton (NFA) in the way that it will have possible paths to take when receiving a `Packet`. The leaves of the tree can be considered to be the final states of the NFA.

3.1.4.1 Example of validation

An example of the aforementioned tree of `Validators` is shown in figure 3.2 on the next page, represented as a NFA. The input symbols correspond to messages. When an accepting state is reached, we consider the validation as having passed.

In case we for the given example would receive a sequence of messages consisting of `A - B - D`, it will make the whole tree pass, since it ends in a final state, i.e. a leaf. Per definition we have that the whole tree will stop when a final state is reached, as this indicates that we have reached what was expected, obviating any further validating.

3.1.4.2 Location of Validators

One could have chosen to have `Validators` on all levels of the solution. We however, have chosen to place validators only on `BasicTemplates`, since communication with DTUsat only takes place in `BasicTemplates`. This means that the result for *Tests* should be fetched from the underlying `BasicTemplates`.

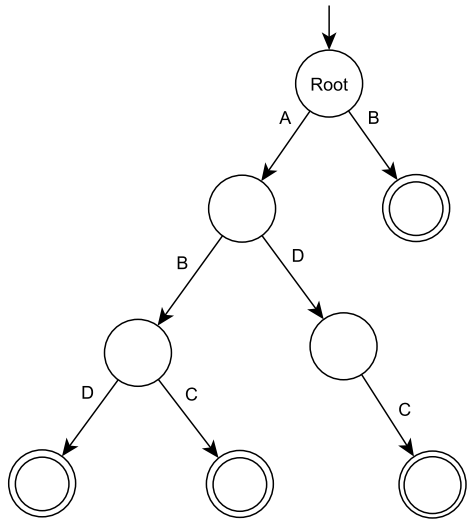

Figure 3.2: A validator tree, represented as an NFA. The tree will succeed in case it receives a sequence of messages consisting of `A - B - D`, since it would reach a final state, i.e. a leaf.

3.1.4.3 Guards

Since we have the possibility of making a `Validator` tree, we have added extra functionality to the tree in the form of guards. The user might not want to always proceed down all branches of the tree. For this reason, we have introduced the concept of guards which are meant to determine whether validation should continue down a particular branch. Thus `Validators` have the option of having guards set.

The concepts of having different paths in the validation tree, as well as guards, was adapted from TTCN-3, which uses it to conclude on the result depending on which messages it receives.

3.2 Overall design

After presenting a high level overview of how our system will communicate with the satellite, we will now present an overview of the components our system consists of as well as the general ideas for the different components

3.2.1 Component diagram

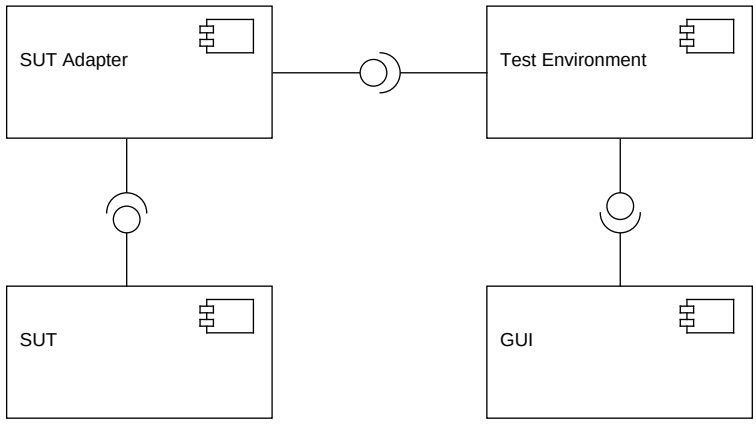

Figure 3.3: Component diagram showing the overall structure of the program

From the component diagram shown in figure 3.3, it can be seen that our solution will consist of three components. They are as follows:

Test Environment This will be the core component in our system, which will handle testing and be responsible for verifying that we get the expected data back in reply to a command.

SUT Adapter This acts as a layer between the test environment and the SUT. It will be responsible for maintaining the connection and making sure that the different tests receive the right answer, so they can analyze it. The `SUT Adapter` will have a defined interface, so the test environment itself does not have to be updated if changes happen on the SUT. Instead, changes would only have to be made to the `SUT Adapter`.

GUI This is the layer that will be presented to the user, in which one is able to create new tests, see results etc.

3.3 Test Environment

The major component of our system is the test environment, as it is here that all test logic will be placed.

3.3.1 Overview

In figure 3.4, a component diagram depicting the different sub components needed to implement the test environment, and the relations between these, can be seen.

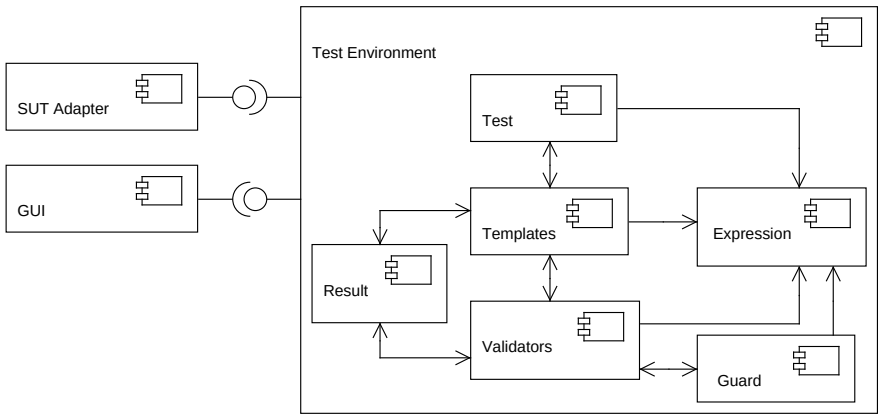

Figure 3.4: The different sub components found in the test environment and the relation between these

3.3.1.1 Subcomponents

To summarize how the structure of the test environment is going to be, we have the following components:

Expression will enable the possibility of using local `Variables` in the system, and instantiating these with `Arguments`.

Templates will define a `Command` with the use of variables. Additionally, expected responses can be defined with the use of either variables or arguments.

Tests will either instantiate templates with arguments, enabling them to be sent to the satellite, or consist of a collection of sub tests.

Result will contain the actual result for the templates and validators.

Validators are used to validate the messages that we get back from the satellite.

Guards will disable different validation paths depending on the value of a `Variable`.

3.3.1.2 Example of a send execution

Before any details are given about the design, the example shown on figure 3.5, will give an idea of how the system is going to work when sending a `Command`.

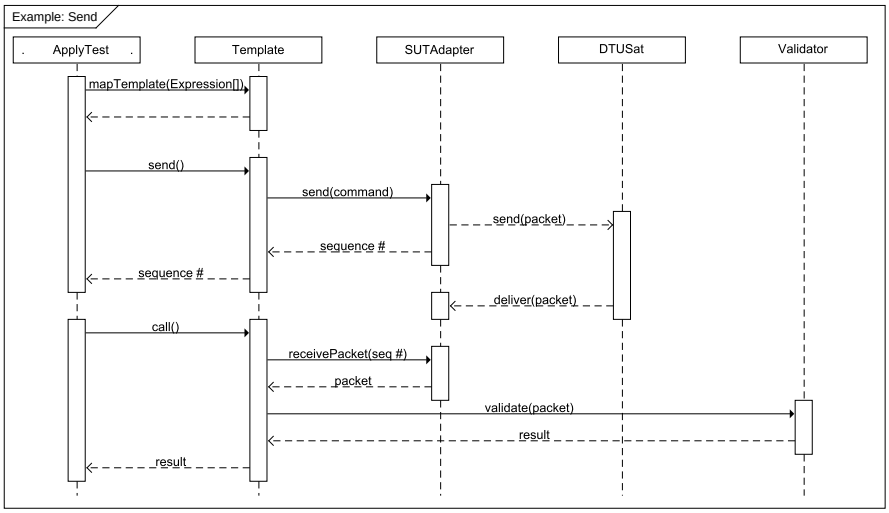

Figure 3.5: A simple example of how a `command` is sent and the return packet is validated

The figure shows the process of sending a command and validating the messages that are returned. This process starts with an `ApplyTest` mapping its expressions to the underlying *`Template`*. This results in the *`Expressions`* from the `ApplyTest` begin mapped together with the `Variables` found on the *`Template`*.

After the *`Expressions`* have been mapped to the *`Template`*, the `ApplyTest` will call send on the *`Template`*. This will, in turn, send the DTUsat `command` to the `SUT Adapter`, which will make an asynchronous call on the satellite sending the `command` so DTUsat can process it. This also means that we at some point in time will get a response from DTUsat. This response will be stored an appropriate place in the `SUT Adapter`.

Whenever the `ApplyTest` initiates the retrieval of the `Packet` through the *`Tem-`*

plate, there is a chance that it is already located on the `SUT Adapter`. In this case, the *Template* can easily retrieve it. Otherwise, it would have to wait for the `Packet` to arrive.

3.4 Expressions

An expression is an object which in the end should provide us with a concrete value to send alongside a parameter on the `Command`. Figure 3.6 shows an overview of the different *Expressions* the system will support (we will later, in 3.8.3.1 on page 61 explain about another class which extends *Expression*).

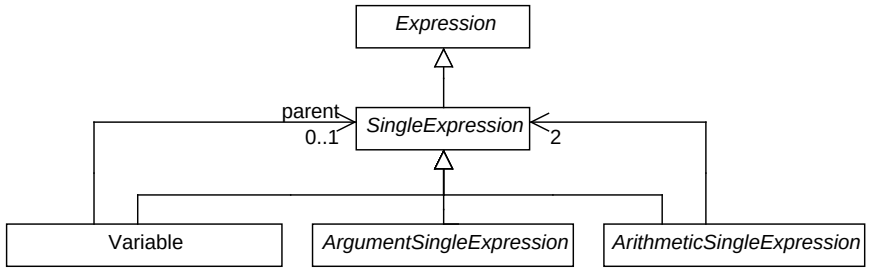

Figure 3.6: The different expressions found in the system

The idea is that *ArgumentSingleExpressions* are the only *Expressions* which will hold a concrete value. The other *Expressions* thus ultimately have to reference *ArgumentSingleExpressions* to be able to get a value at runtime, as we will explain in section 3.4.2.1 on the following page.

3.4.1 ArgumentSingleExpressions

ArgumentSingleExpressions is the most basic expression found in our system. It will basically encapsulate a DTUsat argument, giving us the option of adding more functionality to the argument. This also means that we will have the same types of arguments as found in DTUsat.

As seen in figure 3.7 on the next page, we will have multiple *ArgumentSingleExpressions*. In the first edition of the test environment we will focus on implementing `IntArgumentSingleExpression` as well as `EnumArgumentS-`

ingleExpression, since a majority of the defined messages currently can be sent and received using these two types of *ArgumentSingleExpression*.

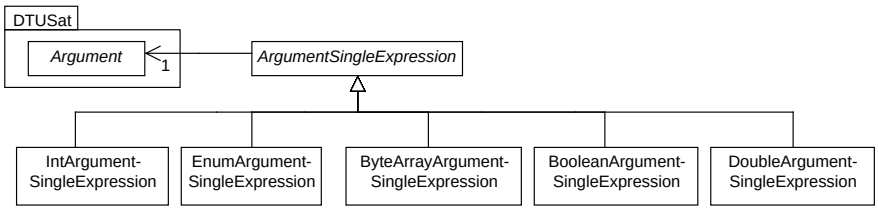

Figure 3.7: The different *ArgumentSingleExpressions* that should be available in the final system

3.4.2 Variables

In the system we also have the concept of Variables. The idea behind having a Variable is that one can have a dynamic test, in which the Variable can be bound to different values depending on the situation. As it can be seen from the class diagram in figure 3.6 on the preceding page, a Variable can have a reference to a *SingleExpression*, which is considered its parent.

An illustration of this hierarchy of variables can be seen in figure 3.8 on the next page. We will describe how the parent hierarchy is set up in section 3.4.2.1.

3.4.2.1 Mapping to arguments

If the parent hierarchy is assumed to already be set up, figure 3.8 on the next page shows a BasicTemplate with three variables. Some of the variables are references to other variables. They have each had their parent pointer set to point to other expressions. When the BasicTemplate needs the actual values of its parameters, the expression hierarchy is walked upwards through the parent pointers until an *ArgumentSingleExpression* is encountered. In the example on figure 3.8, the variables A, B and C will thus end up with the concrete values 10, 10 and 15, respectively.

In general, since a Variable has a reference to a parent *SingleExpression*, we can walk upwards in this parent hierarchy until we meet an expression with an actual value: An *ArgumentSingleExpression*.

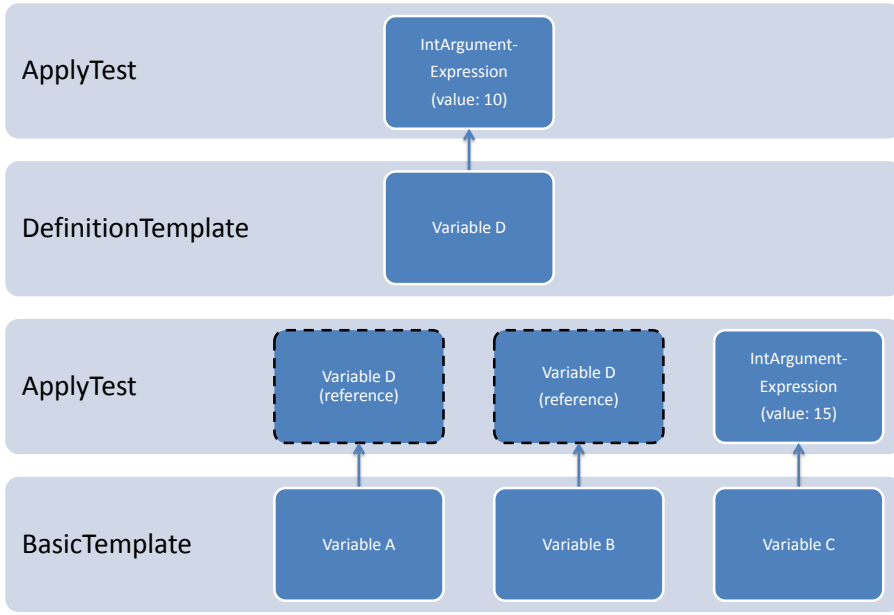

Figure 3.8: Variables with parent pointers to other expressions

The procedure of setting up this parent hierarchy of expressions is called *mapping*. We intend to do *mapping* before sending. The process of *mapping* will be described in 3.6.2.1 on page 54.

Requirements at runtime Since the program can only be run when all `Variables` are bound to actual values, it is a requirement that there are no free variables when tests are to be executed. The system should thus make sure that all variables are successfully mapped to concrete arguments through their parent hierarchy.

3.4.2.2 Type checking

The `Variable` itself doesn't care what value it gets assigned, which means there is no type checking in our system. Type checking can be added to the system at a later point if it is deemed necessary. The current system can be viewed as employing implicit type checking, in that errors will occur if the wrong types are used.

3.4.3 Arithmetic

The final expression found in the system is the arithmetic expression. The structure of arithmetic expressions can be seen in figure 3.9. As the name indicates, it will enable the user to set up arithmetic expressions using the expressions found in the system. It will be possible to create complex expressions using this type, as it can contain a reference to another arithmetic expression.

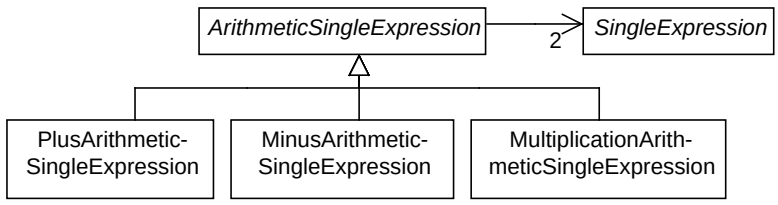

Figure 3.9: Different arithmetic expressions that the system should support

The first edition of the test environment should support the following three

arithmetic expressions: *plus*, *minus* and *multiplication*, as this will yield a fair amount of freedom in setting up arithmetic expressions. One could consider adding *division* in a later version of the system in case this is needed.

3.4.3.1 Improved generality

The idea behind having arithmetic expressions is that it enables the creation of complex *Expressions*, improving the amount of generality in the system. It also provides the backbone for creating scenarios that would not be possible to create without these expressions.

3.5 Templates

A central part of our test environment is the concept of *Templates* which have been introduced to improve reusability of defined tests. In essence, *Templates* resemble functions in programming languages, which also serve to improve code readability and reusability by separating the program into smaller components.

An overview of the components that are involved in our definition of *Templates* can be seen in figure 3.10.

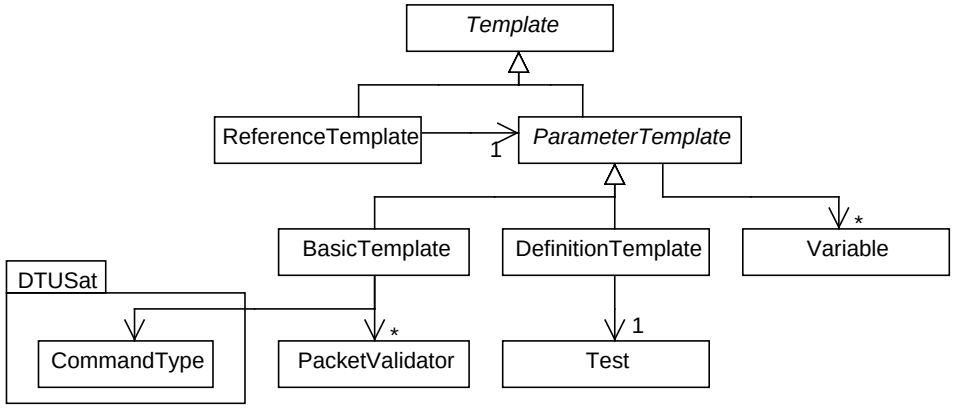

Figure 3.10: Class diagram for templates

3.5.1 Variables in *ParameterTemplates*

Like functions, our *ParameterTemplates* also have a (possibly empty) list of variables. These variables can then subsequently be used inside the *ParameterTemplate*. Also like functions, our *Templates* can't be used on their own: They need to be instantiated with an ApplyTest which we will describe in section 3.6.2 on page 53.

3.5.1.1 Visibility of variables

It should be noted that a template works as a sort of barrier in that only the Variables that are defined in the template are able to be used on the underlying levels. In this way the underlying levels are free to use the Variables.

In other words, Variables defined outside the nearest enclosing *Template* are not able to be used without being explicitly mapped through the nearest enclosing *Template*.

3.5.2 Reference Template

A ReferenceTemplates is, as the name implies, simply a reference to a *ParameterTemplate* (which represents either a BasicTemplate or a DefinitionTemplate). It is used for when one wants to reuse an already defined template.

We have chosen to have a ReferenceTemplate reference *ParameterTemplates* instead of *Templates*. This design choice has been made, since one doesn't lose any functionality by doing this. Additionally, we avoid having to check for infinite loops in case a ReferenceTemplate is set to reference itself.

3.5.3 Basic Template

The most basic form of template is called a BasicTemplate and extends *ParameterTemplate*. It is the only part of the test environment that is able to contain commands that should be sent to the satellite. In the class diagram, this DTUsat command has been represented by its CommandType. Since only BasicTemplates contain DTUsat commands, all test scenarios will be built on the foundation of BasicTemplates.

3.5.3.1 DTUsat commands

As mentioned, a `BasicTemplate` contains a command that should be sent to the satellite. Commands sent to the satellite can also require parameters. It is thus possible to map the `Variables` from the `BasicTemplate` together with the parameters for the command to the satellite. As a design choice we have decided that one can only map `Variables` to the satellite parameters. It is thus not possible to map `ArgumentExpressions` directly to the satellite parameters, which can instead be done in the `ApplyTest` encapsulating the `BasicTemplate`.

The mapping of the parameters from the satellite command together with expressions in our system is done with an object called `DTUSatParameterExpression`. The same example as in figure 3.8 on page 47, but with additional information symbolizing the mapping of DTUsat command parameters to expressions, can be seen in figure 3.11.

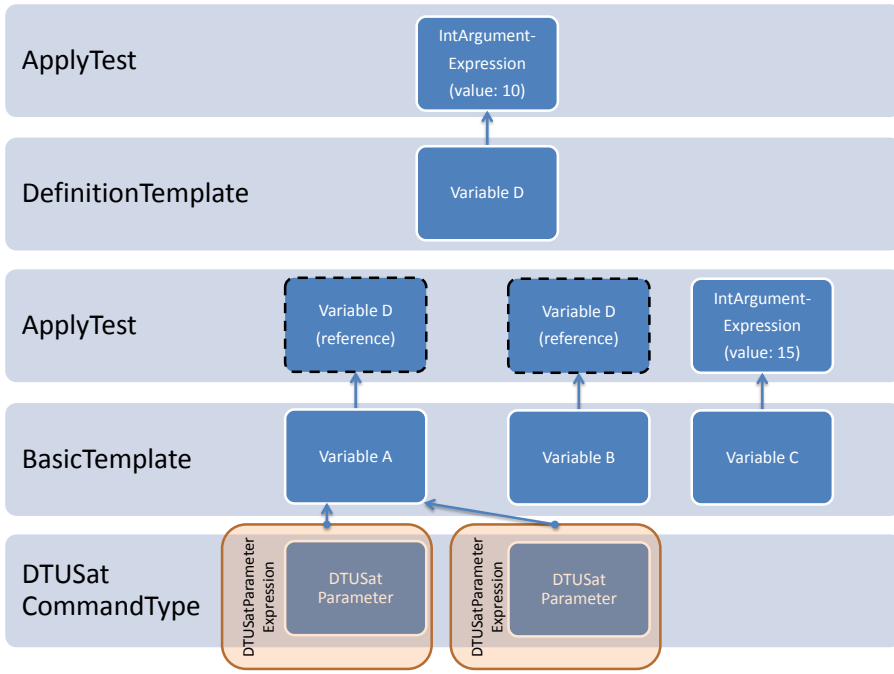

Figure 3.11: Mapping DTUsat command parameters to expressions using `DTUSatParameterExpressions`

When a command is set on a `BasicTemplate`, a series of `DTUSatParameterExpression` objects are created, corresponding to the parameters. These objects

can be seen in the figure as the boxes surrounding the DTUsat parameters. These DTUSatParameterExpressions then contain a reference to the actual variables that are to be mapped to DTUsat parameters. In our example, it can be seen that both parameters are mapped to Variable A, which in this case receives the IntArgumentSingleExpression with the value 10 at runtime.

As it can also be seen from the figure, not all variables from the BasicTemplate are mapped to DTUsat parameters. They could be used for other purposes, such as Validators or guards.

3.5.3.2 Other parts of Basic Templates

Since a BasicTemplate is responsible for sending commands to the satellite, it is also the level that is responsible for validating the received response against what was expected (as discussed in 3.1.4.2 on page 40). From this, it should be able to yield a result. PacketValidators and PacketResults are responsible for these two areas and will be described in detail in section 3.8 on page 58 and section 3.9 on page 61, respectively.

3.5.4 Definition Template

While a BasicTemplate is a template with variables that holds a DTUsat command, a DefinitionTemplate is a template with variables that holds a *Test*. Like a BasicTemplate, DefinitionTemplate also extends *ParameterTemplate*, where the common functionality for having variables is contained. Since a *Test* can consist of a whole hierarchy of *Tests* and *Templates*, one is essentially able to separate this hierarchy into a function with associated variables.

A DefinitionTemplate can thus also be viewed as a sort of barrier in the program where one explicitly specifies variables. These variables are then available to the underlying levels and should be set by the outer levels.

3.6 Tests

While *Templates* are a vital part of the test environment, *Tests* are another major building block. *Tests* can be combined with other *Tests* and *Templates* in various ways to form larger structures.

An overview of the structure of *Tests* can be seen in figure 3.12. In this section we will go through the elements of the diagram to describe their functionality.

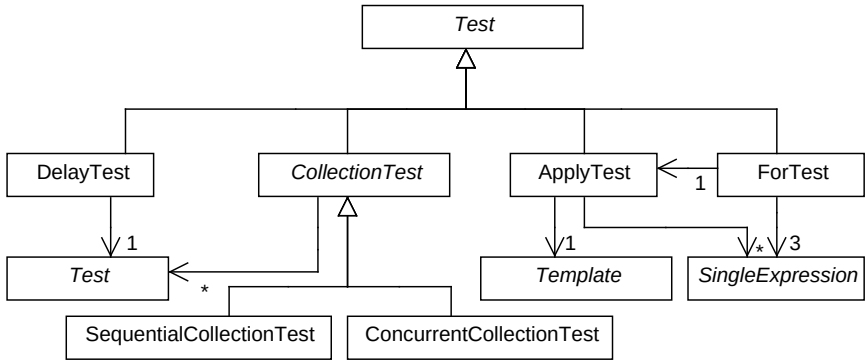

Figure 3.12: Class diagram for tests

3.6.1 Common functionality for tests

As mentioned in 3.1.3.2 on page 39, tests share a common interface which guarantees they implement a `send()` method. We rely on having designed this method such that it returns after the *Test* has actually sent something.

Additionally, all *Tests* will be able to make `BasicTemplates` under it retrieve any responses to the `Commands` they send. This is achieved by making a recursive call down the *Test* hierarchy.

3.6.2 Apply Test

An `ApplyTest` is the only *Test* that can contain *Templates*. If *Templates* are thought of as being functions, `ApplyTests` can be thought of as being calls to these functions.

When calling a function that takes parameters, all parameters must be set. Likewise for *Templates* that contain a number of `Variables`: For each `Variable` on the *Template* there must be a corresponding *SingleExpression* on the `ApplyTest`.

3.6.2.1 Mapping ApplyTests to *Template*

As discussed in 3.4.2.1 on page 46, we map Variables together in a parent hierarchy. This is done when an ApplyTest needs to be mapped to an underlying *ParameterTemplate*. This mapping is done before send is called. We will now describe the process of mapping an ApplyTest to its underlying *Template*.

When send is called on an ApplyTest, it first makes sure to map its *SingleExpressions* to the underlying template. This is done in a 1:1 fashion, where each *SingleExpression* from the ApplyTest is mapped to the corresponding Variable in the *Template*. By looking at figure 3.11 on page 51 again, it can thus be seen that the mapping from ApplyTest to *Template* is always done in the same order. This can be compared to function calls, where arguments are also given in a specific order.

Since this mapping is done recursively each time send is called in an ApplyTest, the end result is that the whole hierarchy of *SingleExpressions* is set up. Thus the Variables in a BasicTemplate will be able to retrieve their actual value by traversing this hierarchy. It should be noted that there won't be conflicts in case multiple ApplyTests are referencing the same *Template*, since these *Templates* and their Variables need to be cloned before sending, which will be discussed in 3.11.2 on page 66.

3.6.3 Collection test

We find that a needed feature in our test environment is the possibility of combining a number of tests into a larger collection of tests. This functionality has been placed in the abstract class *CollectionTest* which supports adding a number of *Tests* as children.

We have two actual classes extending *CollectionTest*: *SequentialCollectionTest* and *ConcurrentCollectionTest* to handle two different use cases:

3.6.3.1 Sequential Collections

To handle the case where one wants to execute a single test at a time, we have defined a *SequentialCollectionTest*. A sequence diagram of the flow of execution of this type can be seen in figure 3.13 on the facing page.

The figure depicts a *SequentialCollectionTest* containing three tests. When

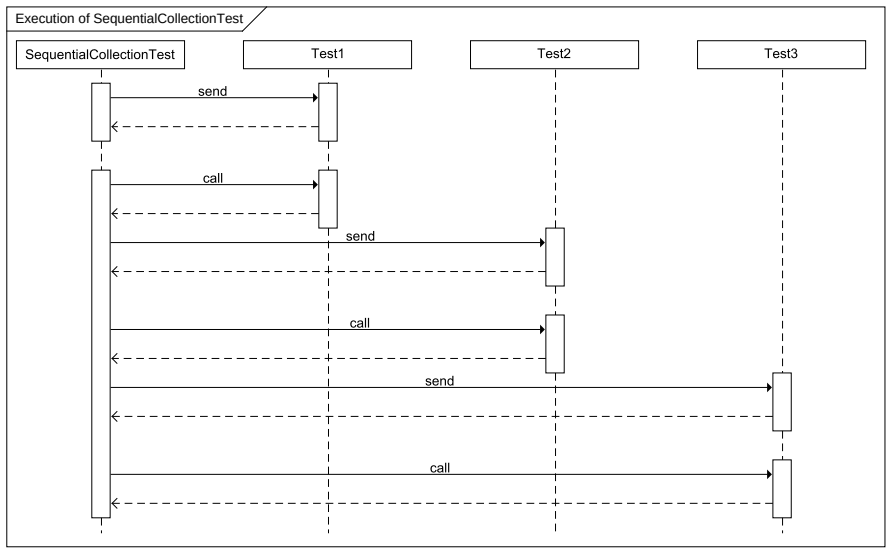

Figure 3.13: Sequence diagram of the execution of a SequentialCollectionTest

`send()` is called on the *SequentialCollectionTest* it sends the first *Test* in the collection and then returns. When one later wants to fetch the results, the result from the first *Test* is fetched after which each of the remaining *Tests* in the collection is sent and has results fetched before moving on to the next *Test*. The *SequentialCollectionTest* doesn't return until it has been through all its sub *Tests* in this sequential manner.

As a result of this way of calling send and fetching results two *SequentialCollectionTests* combined together sequentially will behave the same way as if their sub *Tests* were combined together in one large *SequentialCollectionTest*. We consider this intuitive behavior.

3.6.3.2 Concurrent Collections

At times one might want to execute the tests in a collection concurrently to see how the satellite behaves when the execution of commands is interleaved. For this purpose a *ConcurrentCollectionTest* can be used. A sequence diagram of the behavior of a *ConcurrentCollectionTest* can be seen in figure 3.14 on the next page.

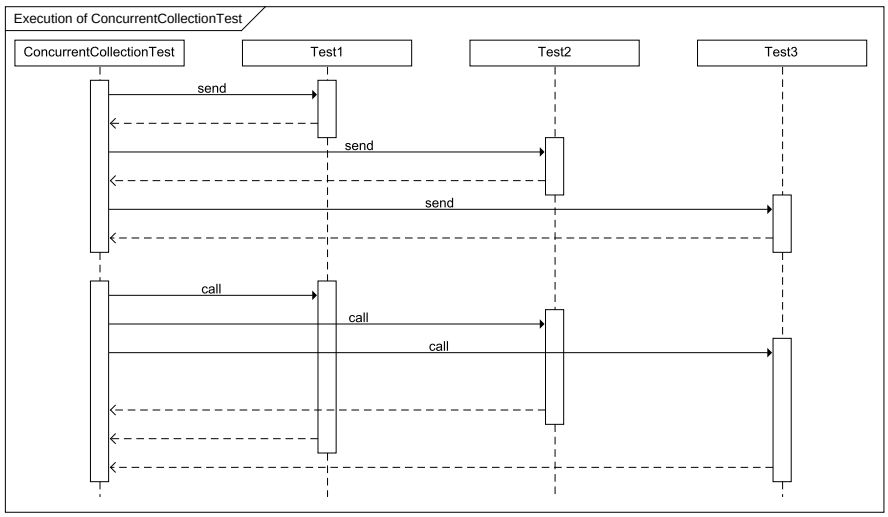

Figure 3.14: Sequence diagram of the execution of a ConcurrentCollectionTest

As it can be seen from the figure, a *ConcurrentCollectionTest* sends all its sub *Tests* in a sequential manner before returning when `send` is called. In this way one can be certain that all tests are sent in a reproducible manner. This is a *ConcurrentCollectionTest* though, which means that one can fetch results as soon as all the sub *Tests* have been sent (as opposed to a *SequentialCollectionTest*, where each sub *Test* is sent and fetched before the next sub *Test* is sent).

Also in this case, combining multiple *ConcurrentCollectionTests* together concurrently results in the same behavior as if they were simply one large *ConcurrentCollectionTest*.

3.6.4 Delay Test

Since it should also be possible to have a delay between sending tests, our test environment has the concept of a `DelayTest`. A `DelayTest` is a fairly simple component which contains a reference to another test. In this way, one can wrap a test in a `DelayTest` to be able to introduce a delay before the test is executed.

There should be a default value set on a `DelayTest`, which we have chosen to

be 0 seconds. In this way, a DelayTest by default does not change the behavior of the underlying tests.

3.6.5 For Test

A ForTest is an automatic instantiation of the referenced ApplyTest, making it run multiple times, depending on the input. Besides this, it will provide a Variable, which the ApplyTest can use as input to the template it has under it. This means that the ForTest provides the option of having a *Test* vary on the Arguments that are passed to the satellite.

A ForTest will send and receive in a manner similar to the one for the *SequentialCollectionTest*, seen in figure 3.13 on page 55, since it will send one ApplyTest at a time and wait for it to be completed before more ApplyTests can be sent and received.

3.7 Results

Like all test systems, we need a way of finding out what happened in a test, i.e. what the result of it was. We have decided that we will use the following types as an indication of the status of a test:

None Indicates the initial result on a test case, before the test has been run.

Pass This means that what was expected came back.

Inconclusive Adapted from TTCN-3. Indicates that we couldn't conclude anything on the given test. This will typically happen when the test case times out, which clearly is not a fail nor pass, as we got nothing to test upon.

Guard Is used to state that no branch was completed due to a guard.

Fail Will be set when a test is getting an unexpected result back.

By having several kind of results, one can with ease see what went wrong during a test execution.

3.7.1 Ordering

We have decided to order the results as follows:

$$\text{None} > \text{Pass} > \text{Inconclusive} > \text{Guard} > \text{Fail}$$

One could argue that the three worst results, Inconclusive, Guard and Fail, are equally bad, since they all state that something didn't go as planned. However, in case one gets Fail, an unexpected packet was received. The same could be said about the Guard result, meaning we consider Guard and Fail to be equally bad. Inconclusive, on the other hand, means we couldn't conclude anything upon the received packets, or the lack of packets, meaning it can be considered less severe than a Fail or Guard.

3.7.2 Summarizing results

In case we have a *ConcurrentCollectionTest*, *SequentialCollectionTest* or any other *Test* consisting of other *Tests*, the overall result for these will be the worst result from all the sub-*Tests* in the collection, according to the ordering shown above. For an `ApplyTest`, the result will be based from the `BasicTemplate` or `DefinitionTemplates` it is referencing.

3.8 Validators

Validators are the backbone of our test environment, as they will be responsible for the actual validation of the received packets (thereby the name validators). They will be responsible for delivering a result to a test.

In figure 3.15, the structure of validators is shown. As it can be seen, there exists two kinds of validators:

Packet Validator has the purpose of testing whether an incoming packet is of the right type and has the correct name. For more information on this validator, see 3.8.2 on page 60

Expression Validator will be responsible for testing the different parameters found on the incoming packet. For more information on this validator, see 3.8.3 on page 60

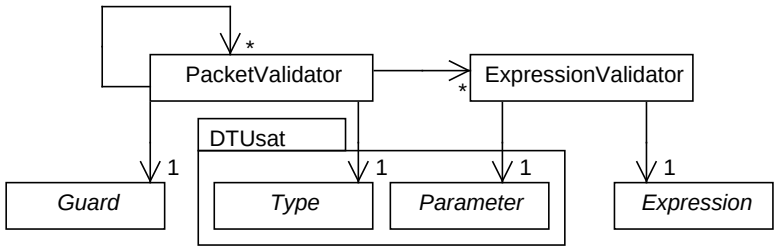

Figure 3.15: Structure for the validators.

3.8.1 Tree structure

As discussed in 3.1.4 on page 40, validators can be considered to be NFAs. Since they have references to other validators, they can also be viewed as being structured as a tree. An example of this tree can be seen in figure 3.16.

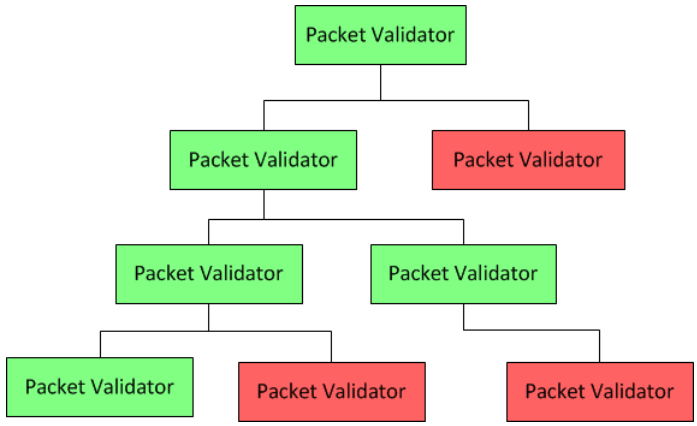

Figure 3.16: The tree structure that can be created from the packet validators. Green means the state has succeeded, while red means it has failed.

3.8.1.1 Evaluation of a BasicTemplate

When the result of a BasicTemplate has to be determined, the tree of PacketValidators can be traversed. As discussed in 3.1.4 on page 40, the structure of PacketValidators can be seen as a NFA. We have chosen to have the PacketValidators act as the edges in the NFA.

The example shown in figure 3.16 has thus succeeded, since a leaf in the tree has successfully validated, corresponding to the NFA having reached a final state. This means that the BasicTemplate containing it has succeeded.

3.8.2 Packet Validators

The sole purpose of a PacketValidator is to check that the name of the received Packet corresponds to what is expected and to conclude on the results of the ExpressionValidators held inside it.

3.8.2.1 Result

How the different results for a PacketValidator are reported will be explained in details in 3.9 on the facing page. However we have a special case, in which the PacketValidator has no ExpressionValidators on it.

No expression validators One situation that might occur is that a packet validator has no expression validators set. This is a peculiar situation as there is not anything to validate upon. However, since expression validators are validating different parameters found on the given packet, one can get in a situation in which no expression validator can even be added. After having taken this into account, we have concluded that a packet validator should succeed in the case where: no expression validators are assigned, we receive that correct packet and the hierarchy of guards succeeds.

3.8.3 Expression Validators

The sole purpose of an expression validator is to test if a given parameter on a received packet is as expected. It should be able to validate the Argument that

belongs to the parameter. This is done by having a reference to an expression, which is able to validate a received argument, to check if both the type and value are as expected.

3.8.3.1 Interval expressions

In 3.4 on page 45, we didn't describe `IntervalExpressions`, since they are only used in expression validators. A class diagram for the `IntervalExpression` can be seen in figure 3.17.

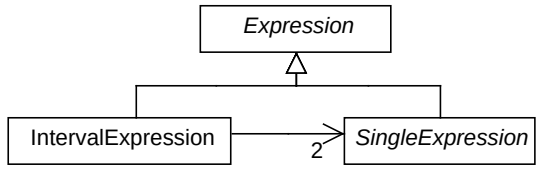

Figure 3.17: The interval expression hooked into the overall expression hierarchy

The `IntervalExpression` can specify a range in which an expected `Argument` is located by keeping a reference to two *SingleExpressions*.

The means that the interval expression will check if the returned value is located in between the two *SingleExpressions* is has references to. However, these must be bound to a numeric value. If not, the interval expression should fail, since an interval cannot contain strings, boolean etc.

3.9 Reporting of results

For the test result to be as accurate as possible, we have chosen to make a class encapsulating the result for each level of our validators, as seen in figure 3.18 on the next page.

This means that we can get an accurate result for each template, stating in which packet or expression validator any errors have occurred. The classes shown in figure 3.18 on the following page have the following roles:

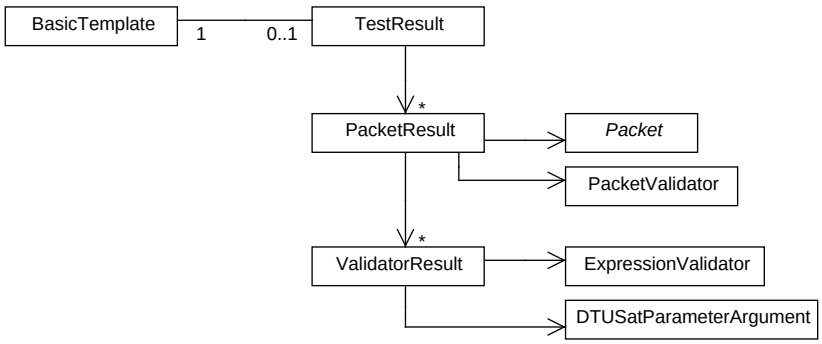

Figure 3.18: The structure used for the results

`TestResult` Is the highest level of the result, and contains the overall result of all the packet validators that exist under it. Besides this, it is the only level which can state if a test case has reached a state in which it is inconclusive.

`PacketResult` Will describe the result from analyzing a received `Packet` on a `PacketValidator`. It will only be `Pass` in case all `ValidatorResults` under it are `Pass` and the expected `Packet` was received. However, in case we received the wrong `Packet`, we will set the given `PacketResult` to `Fail` and not generate any `ValidatorResults`. The result will likewise be `Fail` in case an `ExpressionValidator Fails`. In case the guard on the `PacketValidator` doesn't pass, we set the overall result for the `PacketValidator` to `Guard`, and avoid generating any `ValidatorResults`. For more details about Guards, see 3.10 on the next page.

`ValidatorResult` Is going to indicate how the validation on an `ExpressionValidator` went. It will simply take a parameter and the corresponding `Argument` from the received `Packet` and compare it with the `Argument` set on the `ExpressionValidator`. Since this is the actual validation, one can consider it to be the backbone of the overall result.

3.9.1 Tracking of errors

In order to increase the usability of the result one gets when conducting tests we have different references on the levels of the result structure, as seen in figure 3.18. The idea behind this is to enable one to track the errors that

might have occurred during a test run. Each level of the result structure has a reference to the object that generated the result in addition to what we actually validated upon. For the `PacketResult` this means we have a reference to the `Packet` which was received, while we for the `ValidatorResult` have a reference to an object called `DTUSatParameterArgument`.

By having all the information received available on the different levels of the result, we can track down the source of the result all the way down to the `Arguments` received.

`DTUSatParameterArgument` A class diagram of a `DTUSatParameterArgument` can be seen in figure 3.19. The idea behind this object is to group a pa-

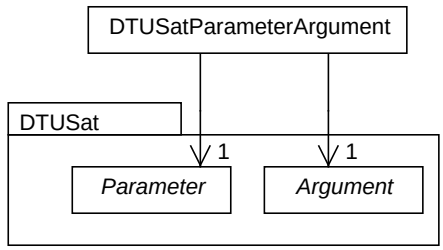

Figure 3.19: Information contained in the `DTUSatParameterArgument`

rameter together with the appropriate `Argument` from DTUsat. This has been created since parameters and `Arguments` by default are not linked together in the `Packet`, but are instead located at the same index in two separate arrays. We have made this object since we find it is good object oriented practice to group related information together in this manner.

3.10 Guards

As discussed in 3.1.4.3 on page 41, our system also has guards. As shown in figure 3.15 on page 59, a `PacketValidator` has a reference to a guard. We have introduced the concept of guards to improve the flexibility one has in specifying trees of `PacketValidators`. Since it is not necessarily relevant to continue down a particular branch in the `PacketValidator` tree, the guard on each `PacketValidator` is checked to see if validation along that branch should continue.

Since the purpose of guards is to determine whether or not execution should continue on a given branch, they need to evaluate to a true/false value. Thus all guards should implement an `evaluate`-method with a boolean return value.

The structure we have chosen for guards can be seen in figure 3.20.

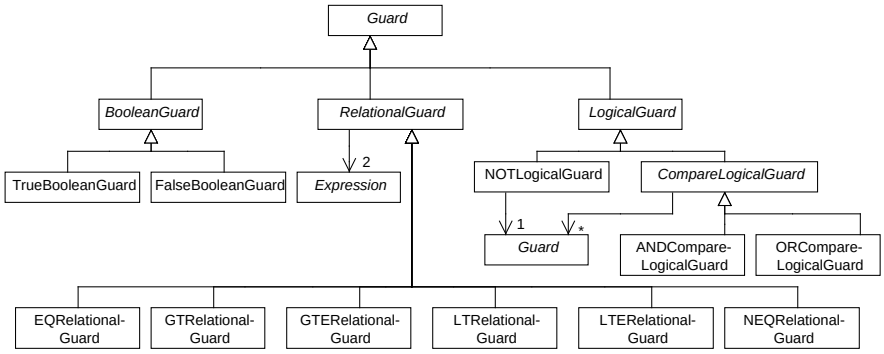

Figure 3.20: Class diagram showing the structure of guards

As it can be seen from the figure we have three major categories of guards: *BooleanGuards*, *RelationalGuards* and *LogicalGuards*. Each of these categories correspond to already known concepts.

3.10.1 Boolean Guards

BooleanGuards are the most basic form of guards. A `TrueBooleanGuard` always evaluates to true while a `FalseBooleanGuard` always evaluates to false. They thus represent the same thing as a true/false value in a normal programming language

3.10.2 Relational Guards

A *RelationalGuard* is a guard that compares two *Expressions*. There exists a series of *RelationalGuards*, each representing a form of comparison. As an example, `EQRelationalGuard` compares two expressions for equality (like == in a normal programming language). The remaining *RelationalGuards* are interpreted as follows:

- GTRelationalGuard: Greater Than (>)
- GTERelationalGuard: Greater Than or Equal to (>=)
- LTRelationalGuard: Less Than (<)
- LTERelationalGuard: Less Than or Equal to (<=)
- NEQRelationalGuard: Not Equal to (!=)

3.10.3 Logical Guards

The third type of guards are *LogicalGuards*. Their use is for logical operators to be used on other guards. There thus exists the concept of a NOTLogicalGuard which inverts the value of the guard. Likewise ANDCompareLogicalGuards and ORCompareLogicalGuards compare the values of a number of guards using && and || operators, respectively.

3.10.4 Default Guard

As a placeholder for the guards found on the PacketValidator, a special guard, called *Root Guard* has been added. By default, i.e. when no guards have been added, we will validate to true. Otherwise, the *Root Guard* will act as an ANDCompareLogicalGuard with only one child.

3.11 General concepts

3.11.1 Timeouts

To have a robust test environment we have decided to have the concept of timeouts. Even though the communication protocol utilized between the satellite and ground should be a reliable protocol, there is still the possibility of the satellite not responding to messages as expected in a timely manner or other unforeseen things happening.

We have thus considered which parts of the system would benefit from being able to have a timeout set. Since BasicTemplates are the only part of the system which send and receive messages from the satellite, we have chosen to

give the user the option of setting a timeout on a `BasicTemplate`. By default the system should have a sufficiently high timeout set as to not have the system erroneously abort while waiting for a reply.

Once the value set for a timeout has been reached, the system should abort the execution of the `BasicTemplate` in question and continue with the remaining execution if possible.

3.11.2 Maintaining state in referenced objects

Since we have introduced the concept of `ReferenceTemplates` we need to consider the case where multiple *Tests* are referencing the same *Template*. If we did nothing special to handle this case, all *Tests* would be operating on the same instance of the *Template*. This would not be optimal if each *Test* assumed to be working on its own copy of the *Template*.

We thus need to be able to handle referencing the same *Template* multiple times. This requires a decision of how to save state in the referenced templates. We have considered a solution where a `ReferenceTemplate` is only instantiated once and is itself responsible for storing the state relevant to the different callers in a suitable object. An alternative solution would be to have unique instances of the referenced objects for each caller. This is the solution we have ended up utilizing, where a `ReferenceTemplate` must make sure to clone the referenced objects before actually using them.

3.12 SUT Adapter

The `SUT Adapter` will be responsible for all communication with the satellite. By having an adapter in the system, our test environment doesn't rely on how the communication with the satellite will work, as we always go through the `SUT Adapter` when trying to send commands. In case the communication with the satellite would change, we can with ease change the adapter and have a functional solution, as long as the new code supports the given interface which the test environment is using. The idea of having an adapter taking care of the communication was adapted from TTCN-3, which also uses a `SUT Adapter` to ensure flexibility in the solution.

The interface which the `SUT Adapter` should implement in order for our test environment to work is described in section 3.12.5 on page 69

Flexibility By having the `SUT Adapter`, we allow ourselves to easily change the system which our test environment will be sending data to. We have utilized this while developing the test environment by creating a simple simulator, which we can test simple features on. A more detailed description of this simulator can be found in 3.13 on page 70.

3.12.1 Structure

As mentioned, the main purpose of the `SUT Adapter` is to be responsible for all communication with the SUT. In order to be able to do this, we use the design seen in figure 3.21.

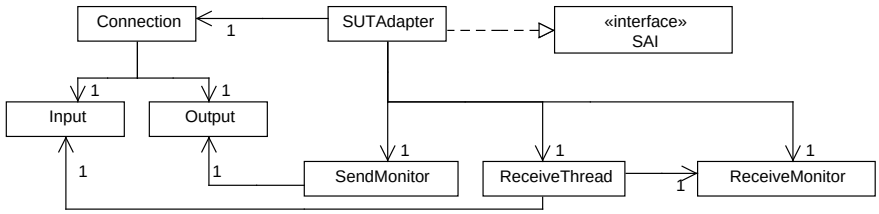

Figure 3.21: The overall structure of the `SUT Adapter`

We have constructed the `SUT Adapter` such that each class has a unique purpose, i.e. send messages, receive messages etc. The different classes will be explained in more details in the following sections.

3.12.2 Connection

The connection class is only responsible for opening a TCP connection to the protocol.

The connection will, after establishing a connection, provide two streams: an input and output stream. Other parts of the system, mainly the `ReceiveThread` and `SendMonitor`, can use these streams in order to send and receive messages.

3.12.3 Sending

In some cases, there will be several test cases running at the same time. This gives rise to concurrency issues when sending, as race conditions could occur. Areas where we should take concurrency issues into account are:

Only one message at a time In case we have multiple test cases trying to send a command at the same time, they could risk interrupting each other's messages. This means that the `OutputStream` should be considered a critical resource.

Unique sequence numbers Every message in the system has a sequence number, which should be unique. This means that the variable which holds the current sequence number should be considered a critical resource, since a race condition might occur when trying to increase the sequence number.

These potential concurrency issues must be taken care of, for example with the use of a monitor, as the class name, `SendMonitor`, indicates.

3.12.4 Receiving

When testing, we need the answers that come back from the satellite, so they can be analyzed to see if they are as expected.

3.12.4.1 Filter

The valid categories of messages can be seen in 2.2.1.1 on page 8. Amongst these messages, `teledata` and `confirm` should always be sent on to the test environment.

Some of the messages we receive from the satellite need to be handled differently, though. Many of the packets we receive are signals from the satellite, which are concerned with the communication between ground and satellite. Signals should thus always be filtered out. Additionally, the satellite can autonomously generate housekeeping teledata, which should be filtered out as well. It is thus the responsibility of the filter to remove unwanted packets from the stream before sending them on to the test which made the original request.

3.12.4.2 Receiving the packets

Retrieval of packets We intend to handle the receiving of `Packets` by having templates wait for a response in case no `Packets` are currently stored. Otherwise the template can just retrieve the waiting `Packet`. This can be seen as an instance of the readers-writers problem, where we could encounter a starvation problem in case a packet arrives at the same time as a test is trying to retrieve a `Packet`.

This means that the set of `Packets` must be considered to be a critical resource, only allowing one thread at a time to have access to it.

When the concurrency issues have been taken care of, one can easily find a `Packet`, which matches a test case. To do this, the `SUT Adapter` will simply have to look for the sequence number corresponding to the originally sent `command`.

Detecting incoming `Packets` Even though we can retrieve the `Packets` when they are located in memory, we need a method to get them into memory. This can be handled by having a thread listening for anything on the `InputStream` contained in the `Connection`. Any response from the satellite would be received on this stream. Whenever we get an incoming message, we will decode the bit stream with a method already implemented in the `Packet`.

3.12.5 Interface

Since the main purpose of the `SUT Adapter` is to maintain the connection to the SUT, which in this case is the satellite. This means that we need a specific interface, so we are sure we only send messages of the correct format to the satellite. Our `SUT Adapter` must thus implement an interface with the methods seen in table 3.1 on the next page.

Function	Parameters	Returns
`createConnection()`	`null`	`null`
`closeConnection()`	`null`	`null`
`changeConnection()`	host name, port number	`null`
`receivePacket()`	sequence number	`packet`
`sendCommand()`	execTime, PacketPrio, CommandType, Module-ID, Arguments[]	sequence number

Table 3.1: The functions supported by the interfaces used by the `SUT Adapter`

createConnection establishes a connection to the satellite, so we can begin testing on the satellite. This methods will have to be called in order for the system to function correctly.

closeConnection takes down the connection to the satellite.

changeConnection will establish a new connection with the given arguments. Additionally, it will have to clean up the current connection and set up a new connection to the given host using the port. This can be achieved by calling the methods `createConnection` and `closeConnection`. In case the connection is connecting to the same address, no action will be taken, but the old connection will keep running.

receivePacket will get the next packet to arrive to the `SUT Adapter` with the given sequence number.

sendCommand is the most important function in our system, as it enables us to send commands to the satellite and thereby test the reply we get.

3.13 Simulator

As discussed in chapter 3.12 on page 66 we have the option of using different `SUT Adapters`. We have utilized this functionality to create a separate `SUT Adapter` which doesn't interface directly with the satellite, but instead connects to a simulator. This simulator has been designed such that we are able to test the basic functionality of our test environment without having access to FlatSat.

Early in the design process a choice was made to create a simple simulator for emulating the flash module, which is responsible for handling access to the onboard flash memory. We have chosen to simulate this module since a series of commands were already defined that targeted this module early on in the

project. Later on, the simulator could potentially be expanded to simulate other parts of the satellite, if needed.

The primary goal of the simulator is not to be an exact functional copy of the satellite, but more to be a simple element for testing the functionality of our test environment. For more elaborate testing with advanced scenarios we plan to test against FlatSat.

Chapter 4

Implementation

In this chapter we'll highlight some of the issues we have encountered during the implementation and the methods we've used for solving them.

If the reader wants to look at the source code, or run the program, while reading this chapter, a guide with instructions on how to run the program, as well as instructions on finding the source code, can be found in appendix A on page 103.

4.1 Implementation process

4.2 `SUT Adapter`

We have had to take some issues into account when implementing the `SUT Adapter` regarding concurrency and the storing of messages for later retrieval:

4.2.1 Concurrency-issues

Since the `SUT Adapter` will handle all communication with the SUT, we can experience concurrency issues, as mentioned in 3.12.3 on page 68.

4.2.1.1 Receiving

For example, a `Packet` could be received at the same time as we are trying to retrieve it. In order to resolve this issue, we have utilized that Java supports monitors, meaning only one thread at a time can access critical resources/sections inside the monitor. This has been implemented by making the critical methods `synchronized`, the Java implementation of monitors. The code snippets shown in figure 4.1 and figure 4.2 are the only methods that can be accessed in the receiving monitor. Since both are `synchronized` the whole class can be considered to be a monitor. As it can be seen from figure 4.1, the current thread will wait in case there are no messages with the given sequence number, until it is notified about changes.

```
public synchronized Packet receivePacket(long seqNo)
        throws InterruptedException {
    while (!queue.containsKey(seqNo) || isQueueEmpty(seqNo)) {
        this.wait();
    }
    // Get, delete and return the packet
}
```

Figure 4.1: The process of trying retrieve a packet from the Receiving Monitor. This, together with code shown in figure 4.2 is the code for the monitor.

The process of letting a waiting thread know when it can proceed, is in Java implemented such that another thread will have to `notify()` the threads waiting for packets. However, as the code snippet indicates in figure 4.2, we use `notifyAll()`, which simply notifies all waiting threads that an action has occurred, which they must respond to. We notify all waiting threads, since we don't know if the thread, which the packet belongs to, is waiting.

4.2.1.2 Sending

During the process of sending `Commands`, we can also experience concurrency issues, since multiple `BasicTemplates` may attempt to send at the same time.

```
public synchronized void gotMessage(Packet packet) {
    // Add packet to the queue with the sequence number for
    // the command that the packet is a response to.
    if (queue.size() > 0) {
        this.notifyAll();
    }
}
```

Figure 4.2: Code snippet for when a packet is arriving to the monitor.

This could lead to a corrupt stream, meaning the satellite can't make any sense of the packets it receives. This could, as the previous problem, be resolved with the use of a monitor. The critical resources located here are the output stream, as well as the current sequence number the system has reached.

Sequence number To ensure that we have unique sequence numbers, we have made the method containing the incrementing `synchronized`. This ensures that only one thread at a time can use the method, ensuring unique sequence numbers, as seen in figure 4.3.

```
private synchronized long nextSeqNo() {
    return seqNoCounter++;
}
```

Figure 4.3: The method for fetching a sequence number for a `Command`

Sending In order to resolve that only one `BasicTemplate` at a time will be able to use the output stream, we have used `synchronized` on the output stream itself, which means that only one thread can write on the output stream at a time, as seen in figure 4.4 on the following page.

4.2.2 Storing old messages

In order for us to store and keep track of which messages have arrived for which `BasicTemplates`, we have used a hash table in which a list with all received packets are stored in a first in, first out manner for each sequence number. In this way we can easily check if there are any packets waiting with a given sequence number. In case there is, we return the first packet with the given sequence number, otherwise we wait for one to arrive.

```
private void sendPacket(Packet pkt) throws IOException {
    if (out != null) {
        synchronized (out) {
            out.writeObject(pkt.toBytes());
            out.flush();
        }
    } else {
        // No connection
    }
}
```

Figure 4.4: Ensuring that only one `Packet` is sent at a time

4.3 Test environment

4.3.1 Maintaining state

As mentioned in 3.11.2 on page 66, we need to clone objects when the same template is referenced multiple times. Whenever `send` is called on a `ReferenceTemplate` we thus make sure to recursively clone its children. Thus, both *Templates* and *Tests* have `clone` methods defined.

In general, an object makes sure to clone itself and makes the cloned copy contain a cloned edition of its children. Additionally, an object makes sure to clone any relevant state as well when `clone` is called.

Due to the fact that we will clone everything, the variable hierarchy becomes corrupted, since variables don't correctly reference objects in the cloned hierarchy. This means that we need to update the cloned variables so they correspond to the clone of the original variable (the variable that was first created). We do this by using the unique names, which variables have, to setup the object references again.

This is a consequence of our object oriented approach, where variables can be used several places in the system. To avoid this issue, we could instead have used the unique names as references. This approach would in essence result in us having to create the object references each time the variable hierarchy was to be traversed, though.

4.3.2 Concurrency

4.3.2.1 Results

Since we have the possibility of performing concurrent testing, multiple `BasicTemplates` can be executed at once. Thus, every `BasicTemplate` should be able to be executed on its own. There are several approaches to solving this issue:

Own thread Every `BasicTemplate` could be its own thread, meaning a new thread would be created whenever a `BasicTemplate` is run.

Utilize futures As we are using Java as our programming language, one can utilize the Future implemented in Java. By using Futures, we are using a known technology, which can be assumed to be working as intended.

After considering these two options, we chose to go with utilizing Futures. In this way we avoid some of the extra overhead of creating threads.

Choosing an ExecutorService Since we have chosen to utilize Futures, an ExecutorService has to be provided in order for the Futures to be executed. We have chosen to use a CachedThreadPool as our ExecutorService, as it will create new threads if needed, and recycle any idle threads. This will reduce some of the overhead of creating new threads.

Getting the results Due to the use of Futures, we have all functionality of receiving results inside a `call()` method, which will be called whenever a *Test* or *Template* is submitted to the ExecutorService. However, the results will not be returned when running the test, but rather stored locally on the `BasicTemplates`, that will be responsible for the actual testing.

4.3.2.2 Interrupts

Due to the fact that one might want to abort a test execution, it should be possible to interrupt the whole execution. However, due to the many threads potentially started in the ExecutorService previously mentioned, the interrupt must be passed through all levels of the *Tests*.

In order to achieve this, we have simply made a method, `interruptFutures`, as seen in figure 4.5, on the different layers in the test system (*Templates* and *Tests*). This method will call `cancel(true)` on all Futures. This will cause the Future to interrupt, enabling us to handle the interrupt gracefully. One must ensure the interrupt flag is set every time we catch an `InterruptedException`, as it would otherwise have been reset, which would result in parents of the *Test* or *Template* not being interrupted.

```
public void interruptFuture() {
    if (futureResults != null) {
        for (Future<Void> future : futureResults) {
            future.cancel(true);
        }
    }
    for (Test test : tests) {
        test.interruptFuture();
    }
}
```

Figure 4.5: `interruptFuture()` method taken from *ConcurrentCollectionTest*, in which we interrupt all Futures and call `interruptFutere()` on all *Tests*, to ensure that all levels of the hierarchy receive the interrupt.

One should also check whether the current thread is interrupted, before starting on either sending or getting results, as seen in figure 4.6. In case the current thread is interrupted, we should simply skip any action that would otherwise have been performed.

```
if (!Thread.currentThread().isInterrupted()) {
    // Send, getting results
}
```

Figure 4.6: Check for whether the current thread is interrupted. In case of an interrupted thread, one should avoid doing anything.

4.3.2.3 Timeout

As stated in 3.11.1 on page 65, `BasicTemplates` should support timeouts. Due to the usage of Futures, one could think that we would be able to use the built in timeout of Futures. However, due to the design we have come up with, this is not an option. We don't call the Future until some time after we have called send. However, the timeout should already start when we send, in

order to avoid that we get stuck when sending due to a lost or slow connection. This means that another approach should be used. For this purpose, one can use a ScheduledExecutorService, which allows one to schedule a task for later execution. This can be used to assign the task shown in figure 4.7, which will interrupt the thread running the `BasicTemplate`, and set the result to be inconclusive.

```
scheduleExecutor.schedule(new Runnable() {
    @Override
    public void run() {
        synchronized (synchronizer) {
            if (thread != null && thread.isAlive()) {
                thread.interrupt();
                testResult.setTimedOut();
            }
        }
    }
}, timeout, TimeUnit.MILLISECONDS);
```

Figure 4.7: The task that will be scheduled to the ScheduledExecutorService, interrupting the thread currently set in the `BasicTemplate` as the currently running thread.

4.3.3 XML-files for save/load

Since we want to be able to save and load the defined tests, we have decided to do this via an XML-format. As an alternative, we could have saved and loaded files using the built-in Java serialization. A downside to this would be that it wouldn't be possible for users to edit the files manually. Additionally, serialization breaks easily when changes are made to the program.

Before diving into the technical details about how we do serialization, we'll start out by showing an example of our XML format.

4.3.3.1 Example of the XML format

A minimal example of the XML format can be seen in figure 4.8 on the next page. Here, an `ApplyTest` has been defined. The `ApplyTest` contains a `BasicTemplate` which sends a `CMD_GEN_SUB_ALIVE` command to the satellite and expects a `CONF_OK` back. As it can be seen, the format is human readable and should grant users the ability to manually edit the XML file in case they need to.

```
<Scenario id="0">
  <RootTemplates/>
  <RootSequentialCollectionTest id="1">
    <Tests>
      <ApplyTest id="2">
        <ApplyExpressions/>
        <BasicTemplate id="3" name="Example test"
           description="Description of example" cmdName=
           "CMD_GEN_SUB_ALIVE" moduleID="EPS">
          <Variables/>
          <DTUSatParameterExpressions id="4"/>
          <PacketValidators>
            <PacketValidator id="5" typeName="CONF_OK"
               packetType="CONFIRM">
              <RootGuard id="6"/>
              <ExpressionValidators/>
              <Children id="7"/>
            </PacketValidator>
          </PacketValidators>
        </BasicTemplate>
      </ApplyTest>
    </Tests>
  </RootSequentialCollectionTest>
</Scenario>
```

Figure 4.8: An example of a test serialized to XML

A more complex example of serialized XML with the use of Simple can be found in appendix B on page 105.

4.3.3.2 Simple XML

Simple XML is an open source framework for serializing and deserializing to and from XML files. Other parts of DTUsat already utilized this framework, so we found it relevant to also use the framework for our project.

Simple XML supports generating the XML shown in figure 4.8 on the preceding page out of the box. This is done in the following way:

Annotations Simple XML automatically serializes and deserializes fields that are annotated with special annotations. As an example, a `PacketValidator` contains an `@Attribute` annotation on the `typeName` field and an `@Element` annotation on its `RootGuard` which automatically results in XML elements being added.

Annotations for abstract classes Abstract classes require a special notation to be used by Simple XML, since it otherwise won't know which class to instantiate. We thus utilize the `@ElementListUnion` annotation when wanting to have a collection containing an abstract class. The syntax of this annotation can be seen in figure 4.9.

```
@Path("RootTemplates")
@ElementListUnion({
    @ElementList(name = "BasicTemplate", entry = "BasicTemplate",
        type = BasicTemplate.class, inline = true, required = false)
        ,
    @ElementList(name = "DefinitionTemplate", entry = "
        DefinitionTemplate", type = DefinitionTemplate.class, inline
         = true, required = false) })
ArrayList<ParameterTemplate> rootTemplates = new ArrayList<
    ParameterTemplate>();
```

Figure 4.9: The @ElementListUnion annotation

As it can be seen, we create an `@ElementListUnion` from two ElementLists. They each contain entries with distinct names. We create both these lists as being inline (meaning that they don't have a tag surrounding them) and then wrap

them in a single tag, called RootTemplates. The final result is that the RootTemplates tag can contain both BasicTemplates and DefinitionTemplates.

Handling object references Simple XML has built in functionality to handle object cycles. First off, Simple XML gives all objects a unique id. Then, whenever there is a reference to an object, Simple XML has the ability to insert a reference to the given id. This is made possible by using a so-called CycleStrategy when serializing and deserializing.

This functionality results in users being able to define *Templates* they want to use multiple times in the <RootTempates> part of the XML file. They can then later refer to the *Templates* using ref tags.

4.4 GUI

To fulfill the requirements, we have created a GUI to ease the creation and execution of tests. To get an idea of how the GUI looks, a screenshot of the program with a series of tests, for testing the onboard camera, can be seen in figure 4.10 on the next page.

4.4.1 Structure

In this section will we give an introduction to the structure of our GUI. The main functionality of the GUI is to set up a Scenario. As mentioned in 3.1.3.1 on page 39, a scenario contains a collection of RootTemplates and a RootTest. In the GUI, the RootTemplates can be seen in the treeview at the top left. At the bottom left, children can be added to the RootTest.

4.4.1.1 Selecting items in trees

Each time an element in one of the trees is clicked, a view for setting up that particular element appears on the right hand side. In the screenshot, the view shown on the right is for setting up a BasicTemplate.

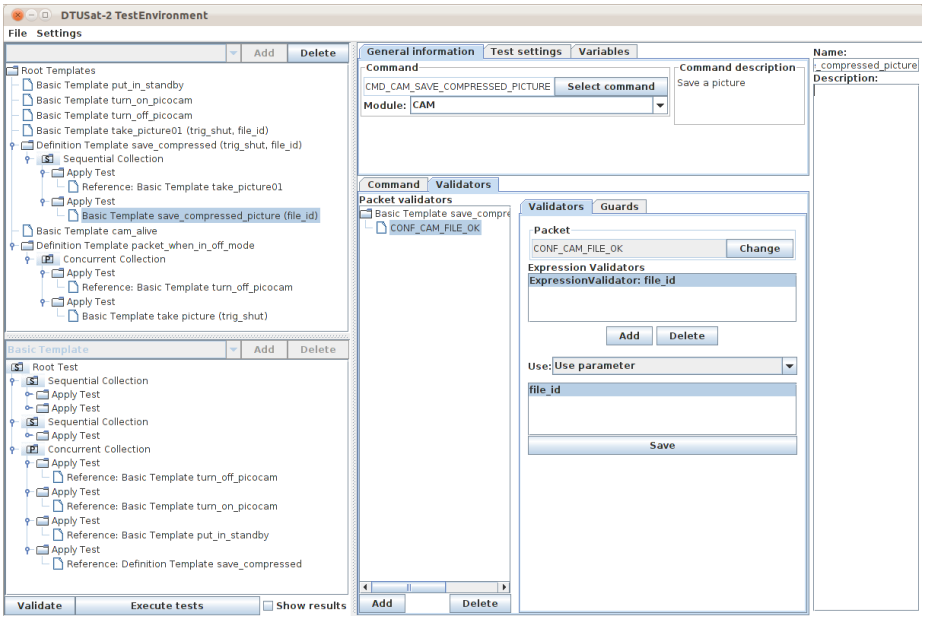

Figure 4.10: A screenshot of the GUI containing tests related to the onboard camera

4.4.1.2 Selecting `ReferenceTemplates`

When a `ReferenceTemplate` is selected in the tree to the left, a list of the `RootTemplates` appears to the right, so a template to reference can be selected.

4.4.1.3 Validating before execution

At the bottom there is a `Validate`-button, which ensures that the Scenario is in a state where it is ready to execute. In case it is not in an executable state, the user is informed of the error with a popup. Additionally, the element with the error is automatically selected in the tree view, such that the user can correct the problem.

4.4.1.4 Executing tests

When the button `Execute Tests` is pressed, the scenario is first validated to ensure it is ready to be executed. If so, the `RootTest` is executed. We will describe how execution is done in section 4.4.5 on page 87.

4.4.1.5 Showing results

When `Show results` is selected, the tree view is expanded such that all referenced objects are shown in the tree as well. Additionally, the `PacketValidators` on `BasicTemplates` can be seen as children in this tree. All items are colored according to the result on the given item, such that the user can easily see what the results were. The colors given to the different results are as follows:

- Pass = Green
- Inconclusive = Gray
- Fail = Red
- Guard = Orange
- None = White

When a `PacketValidator` is selected, another view is shown to the right, wherein the user can view both the received and expected result for each expression validator.

4.4.2 Design choices

Before implementing the GUI, we decided to maintain a clean separation between our model and the GUI using the Model-View-Controller architectural pattern. In doing this we ensure that all relevant functionality is contained in the model, and that it will be easier to maintain the GUI in the future.

To facilitate the connection between the model and the view we have used the Observer-Observable design pattern, to avoid having the model reference the view.

4.4.3 Overview of panels

The topmost level in our view is called `MainFrame`. An overview of the panels contained in the `MainFrame` can be seen in 4.11 on the following page. As it can be seen, we have tried to use a modular approach in designing these panels. This means that we are able to reuse codes different places in the view. As an example, the two views containing trees, `TemplateTreeView` and `TestTreeView` both extend a `TreeView` class, which contains all the common functionality for presenting trees. This class is also reused in many of the panels shown on the right hand side where trees are present.

We have decided to not show an overview of all our panels, but only to show an overview of the one containing the most components: The `BasicTemplateView`. An overview of this can be seen in figure 4.12 on the next page. From the figure it can be seen that the view consists of a series of different panels contained in tabs.

4.4.4 TreeItem

As mentioned, we have reused a view for trees multiple times. To actually be able to do this, we have made all items in our model, which should be able to be shown in a tree view, extend a certain class. We call this class *TreeItem*.

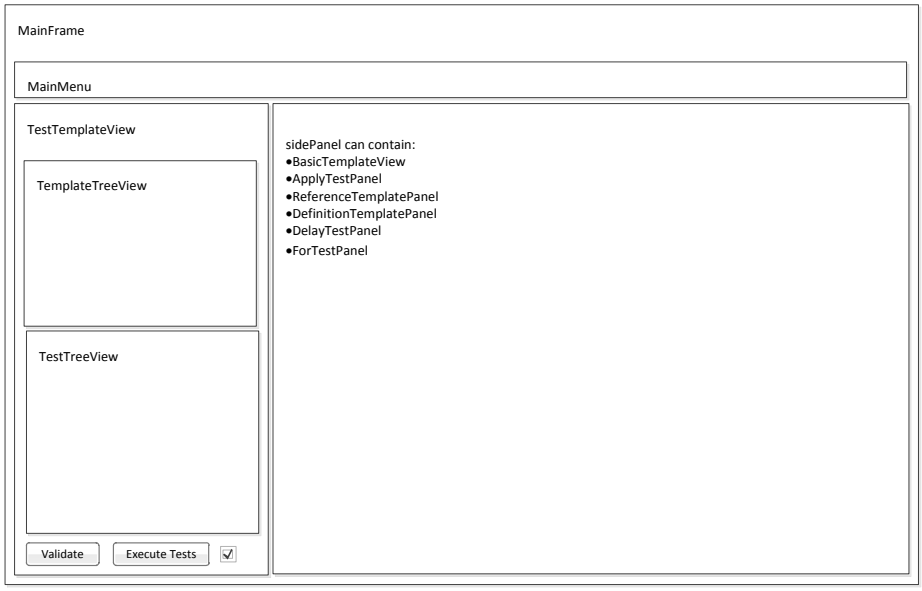

Figure 4.11: The panels contained in MainFrame

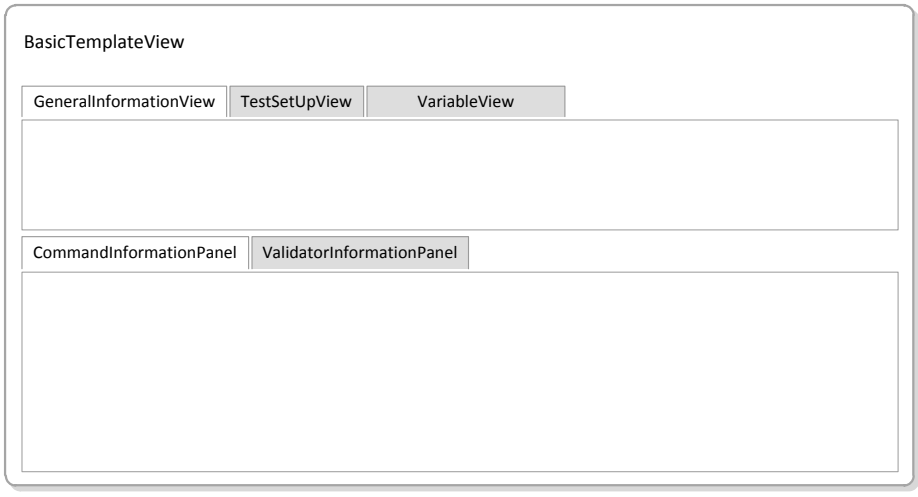

Figure 4.12: The panels contained in BasicTemplateView

Since many of the concepts in our model are shown as trees in the GUI, a lot of our classes extend this class.

The *TreeItem* class mainly contains abstract methods to add, remove and retrieve the *TreeItem*'s children. In this way we are able to draw *TreeItems* in a tree. Likewise, classes extending *TreeItem* are required to be able to return a result. In this way we are also able to color *TreeItems* according to the result they have after execution.

4.4.5 Execution from the GUI

From the view, we have given the user the possibility of executing the constructed tests. After execution it is possible to visually see the results by looking at colors in the tree view. Additionally, it is possible to dig down and view on each `PacketValidator` which results were actually received.

Since it is possible to execute from the GUI, we need to make certain that the user doesn't change anything in the GUI while tests are executing. We don't want to have the GUI freeze completely during execution, though. Thus, we have decided to actually run execution in a separate thread, while we prevent the user from using the GUI. Instead, he is shown a dialog box with the option of aborting execution. If he chooses to do so, we interrupt the thread running the tests and return control of the GUI to the user.

4.5 Logging

We have decided to utilize the loggers from `java.util.logging.Logger`. In this way we are given a predefined structure for logging, and the user has the option of choosing the desired level of logging. We have aimed at giving most debug information at the `info` level, while unexpected events are logged at the `severe` level.

We have configured the loggers to output to a file located in the root directory, `nominaltest2.log`, as well as `System.out`. The default settings for log rotation are used. Since we are utilizing the default logging framework, this can of course be changed fairly easily.

4.6 Command line execution

As mentioned in the requirements, the test environment should support automated execution of tests. This could be used for running the tests automatically at certain times, e.g. at every commit to the revision control system or as a part of nightly builds.

We have thus, as explained in 4.4.2 on page 85, kept the GUI completely separate from the model. In this way, we are easily able to execute scenarios without the GUI.

Our test environment supports being executed with command line arguments. Only the first argument is read. It is expected to be a path to an XML file containing a scenario in the following way:

```
java -jar nominaltest2.jar scenario.xml
```

The user can follow the progress of the execution by viewing the output to console, or alternatively the logs, as described in 4.5 on the previous page.

4.6.1 Outputting results

Even though the user has the option of viewing the debug logs, he is probably more interested in seeing the actual results. For this reason, at every execution of the test environment, we output the results to file. They are output when the complete execution has finished. We have utilized Simple XML for this, in the same manner as described in 4.3.3 on page 79.

An example output from the program can be seen in figure 4.13 on the next page. As it can be seen from the figure, the results are viewable on each level. Additionally, it is possible to see what was expected and actually returned in validatorResults.

```
<testRunResults time="2012-06-28 20:37:16.783 CEST">
  <results>
    <testResult basicTemplateName="Basic Template 1" res=
       "PASS">
      <packetResult>
        <packetResult packetResult="PASS"
           expectedPacketName="CONF_FLA_ERASE_OK"
           receivedPacketName="CONF_FLA_ERASE_OK">
          <validatorResults>
            <validatorResult ExpectedArgument="
               blocks_erased" result="true">
              <Received_Integer value="5"/>
              <Expected>Integer with value of = 5</
                 Expected>
            </validatorResult>
          </validatorResults>
        </packetResult>
      </packetResult>
    </testResult>
  </results>
</testRunResults>
```

Figure 4.13: An example of the output format for results

Chapter 5

Results

In this chapter we will present the methods we have used for testing our system and the conclusions we have drawn on the results that were gathered during this testing.

5.1 Simulator

As mentioned in 3.13 on page 70, we have created a simulator which emulates the flash module. This simulator has been designed such that it has a set amount of memory, currently 32 blocks. It should be noted that our simulator is not intended to emulate the real flash module in the satellite, but is simply a representation of how we would expect such a module to behave.

The simulator responds to `CMD_FLA_ERASE_BLOCKS` commands. This command accepts a parameter, `max_blocks`, which has a lower limit of 1 and an upper limit of 62 blocks to be erased. The simulator is able to give the following responses:

- A `CONF_FLA_ERASE_OK` confirmation is sent if erasing is successful, along with an integer argument with the actual amount of erased blocks.

- If erasing is not possible, a CONF_FLA_FLASH_ERROR confirmation with the enum argument FLASHFS2_OUT_OF_BOUNDS is returned.
- In case a command that is not handled by the flash simulator is received, a CONF_FAILED confirmation is returned.

We have created a simple scenario for testing the functionality of our program together with the flash simulator. This scenario can be found in appendix C on page 113 or in the file: scenarios/simulatorscenario.xml.

We have basically created a BasicTemplate with two Variables: One for the amount of blocks to be erased on the satellite, and one used for checking the amount of flash blocks actually erased by the simulator.

The BasicTemplate has two PacketValidators: One for checking when we correctly erase blocks, and one for checking for the correct response when we are out of bounds. These PacketValidators are made such that only one is active by creating guards on them: The first PacketValidator is only active when the variable for erasing blocks lies in the interval from 1-62. The other PacketValidator is only active outside this interval.

After creating this BasicTemplate, we have created a series of tests actually using the template. We try to test different aspects of the program, including concurrent and sequential collections, delay tests and for tests. Additionally we try testing both validators by erasing a valid amount as well as erasing an invalid amount. The result is that the program is able to execute as expected, which can be seen in the screenshot in figure 5.1 on the facing page. As it can be seen, the first expanded template only expects a CONF_FLA_FLASH_ERROR due to the guards on the other PacketValidator. Likewise, the bottom template successfully receives and validates against a CONF_FLA_ERASE_OK.

5.2 Flatsat

In addition to testing against our own simulator, we have also tested that the system actually works against FlatSat. At the time of testing, not many subsystems were ready, but we ensured that the basic communication with FlatSat worked through the SUT Adapter. This basic testing was done by sending commands to turn a subsystem on and off. Confirmation messages were successfully received by the test environment, and we could visually verify that the subsystem in question was actually turned on and off correctly.

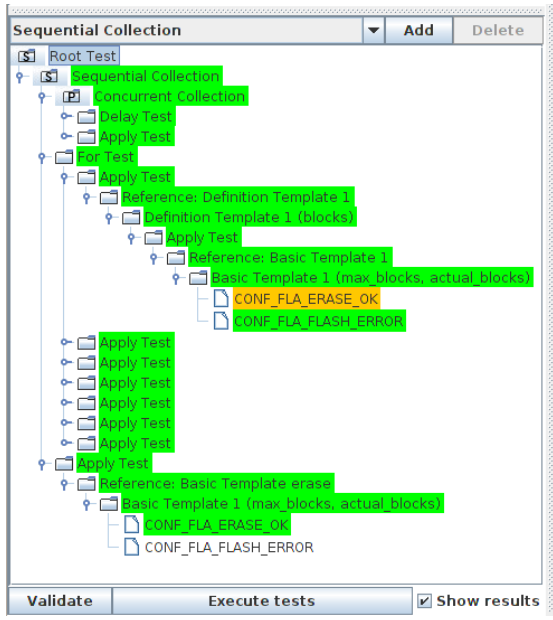

Figure 5.1: A screenshot showing the execution of tests against the simulator

5.3 Setting up tests for the CAM module

To get an idea of whether our system is capable of performing the kinds of real life tests that other students have previously created for certain modules, we have set up a series of tests for testing the CAM module. To do this, we have looked at the use cases in the report "Camera control and data handling on DTUsat"[9]. The use cases in the report are concerned with testing the camera module while it is in various states.

We were able to recreate many of the tests of the camera module. They can be found in appendix B on page 105. A screenshot of the scenario loaded into the GUI can be found in figure 4.10 on page 83.

While looking at the use cases for the camera, we did discover some of the use cases that cannot be set up in our system in its current state. This is especially related to not being able to send commands of type `CMD_CAM_CMD_SEQ`, which require the DTUsat parameter type `ByteArrayParameter`. This type would have to be implemented to be able to send this command type. We will discuss the implementation of more types in 6.1.4 on page 96.

5.4 Overall results

As mentioned in this chapter, we have used different means of testing: Testing against a simulator as well as testing against FlatSat.

In addition to these forms of testing, we also want to consider our other options for testing, especially unit testing. It would be possible to create unit tests of certain parts of our model. We have not found it worthwhile to do so, though, since unit tests would be able to test small units of the model. We instead want to verify the behavior of the test environment in its entirety. For this purpose we found that writing a simulator of the satellite was the right approach. The simulator is not that sophisticated at this point, though. An option for further testing would thus be to write a more advanced simulator, or to simulate other parts of the satellite.

After conducting the tests mentioned in this chapter, we can conclude that we have ended up with an operational program. There are certain areas where we would like to further expand and improve on the program, though, which we will elaborate on in 6 on the facing page.

Chapter 6

Discussion

6.1 Improvements and further developments

We have currently implemented all the features listed in the requirements. However more and more useful features were discovered during the development. This meant that there is some functionality that can be added, which we will discuss in this chapter.

6.1.1 Arithmetic expressions

Since it was a relatively recent addition to our test environment, we did not finish the implementation of arithmetic expressions, which was described in 3.4.3 on page 48. The logic should be relatively complete, however it needs proper testing before it can be released. Furthermore, arithmetic expressions need to be hooked into the GUI.

6.1.2 Conditional test

We had thought of another test that should be implemented in the system. This was a conditional test, that should run in case a condition set on it was true. This could be achieved with the use of the guards already defined in the system, originally intended for the `Validators`. A class diagram of the structure of the proposed conditional test can be seen in figure 6.1.

By having a reference to a guard, one can state which values a `Variable` must have before the test is run. In this way a scenario can test different aspects of the satellite depending on the value of a variable which could be defined globally.

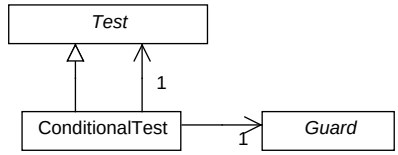

Figure 6.1: The structure of the proposed Conditional test

6.1.3 Defining out-variables on *Templates*

As an addition to the in-variables that one can define on *Templates*, one could also have the option of declaring `Variables` to be used outside the given *Template*. These `Variables` could get their values from the result on the *Template* and could be considered return values or out-variables. By having this feature, one could make other *Tests* in the system depend on the result of a *Test*.

Additionally, this could be combined with the aforementioned possibility of creating conditional tests. In this way, one would be able to have dependencies between *Tests* such that some *Tests* are only run if certain conditions from other *Tests* are met.

6.1.4 More datatypes

The system needs to support all data types found in the DTUsat project. If all data types were supported in the test environment, all aspects of the satellite could be fully tested. This is the functionality one should consider implementing first.

We have currently implemented IntArgument and EnumArgument. The remaining data types in DTUsat are BooleanArgument, ByteArrayArgument and DoubleArgument and should be implemented in a future edition of the test environment.

6.1.5 Merging scenarios

In order to increase the usability of the system, one could implement the possibility of merging two or more scenarios. In this way, the user would avoid having to create a new scenario from scratch. This improvement should be possible to do, since we already save scenarios to XML files. A function to import multiple XML files into the system would thus have to be developed.

6.1.6 Marking which *Tests* to be run

It has been mentioned that it should be possible to state which *Tests* should be run in a scenario. This will increase the usability of the system, since the user would be able to selectively run a subset of *Tests* in a scenario.

6.1.7 Parsing arguments

In the current edition of the GUI, the user has to set up arguments and other elements in a tree structure. Some users might find this cumbersome. They might prefer being able to write e.g. expressions, which are then parsed, instead of setting up a tree structure.

Especially if arithmetic expressions are added to the GUI, it would probably be best for the user to be able to actually write an arithmetic expression, which is then parsed by the system.

If parsing was implemented, several constraint on the naming of variables should be enforced. As an example, the name of a variable shouldn't start with a numeric value. In total, we should impose the same restrictions on naming of variables as variables have in Java.

6.1.8 Loading results

When one uses the test environment to conduct testing, a file will be created. This file contains the results of the specific test run, as explained in 4.6.1 on page 88. In the current state of the system, it is not possible to load these results back into the system for analyzing what exactly went wrong during the testing. If the results could be loaded into the system, one could get a visual representation of what caused the result. We find that this would be a worthwhile improvement.

6.1.9 Automated test generation

Since we have a test environment with many different capabilities, it would be interesting to consider the option of having it automatically generate tests.

As mentioned in 2.2 on page 7, we have a list of valid destinations for each command. Additionally, some commands have a list of legal confirmations. It would thus be possible to automatically generate a series of tests, sending each command and expecting either of the legal confirmations as a reply.

Some commands also have additional information on their parameters, defining the valid interval for the parameter. Thus, we would be able to create a series of tests with arguments in the valid range. Currently, we don't really know what to expect back from the satellite, except for which confirmations are legal. Additional information could potentially be added to the XML file to set up relations between the parameters on the command and the values that can be expected to be returned. Additionally, it could be relevant to add information about the legal teledata that can be returned in reply to a command. In this way, we would be able to create a series of tests automatically, which would test some of the basic functionality of the satellite.

Chapter 7

Conclusion

We have developed a test environment for DTUsat-2. This was done after establishing the requirements for such a test environment through examining the current workings of the satellite as well as determining a series of typical use cases the system should support.

The current edition of the test environment fulfills the established requirements. This was ensured by verifying the functionality of the test environment in two ways: We primarily ensured that the solution was working as intended by having it communicate with a simple simulator of the flash module in the satellite. Additionally, we tested that the test environment can be used with the real satellite by having it communicate with a functional copy of the satellite, FlatSat.

Before choosing to develop our own solution, four existing test environments were surveyed. In doing this, we were able to compare the terminology used across the different test systems, thus realizing that many of them use approximately the same concepts. After concluding that the systems had some shortcomings if they were to be used for testing DTUsat-2, we decided to reuse some concepts from the existing test environments when designing our own solution.

Throughout the project, we have aimed for a high level of generality. In this way, it is possible for the user of the test environment to create advanced test scenarios by combining different elements. As part of the generality, we have

thus made a sharp distinction between *Templates* and *Tests*:

- *Templates* are reusable components in the system. They are the only part of the system that is able to contain commands that should be sent to the satellite. Additionally, they can contain logic for validating that the correct responses are received from the satellite. *Templates* are also able to have `Variables`, which makes them able to be called with different arguments to improve reusability even more.
- *Tests* are able to be combined with other tests to create advanced scenarios. Additionally, they are able to instantiate *Templates* by providing them with actual arguments.

The concepts of *Templates* and *Tests* have further been refined such that there are different kinds of each to improve the possibilities of combining them. Additionally, some of the other concepts we have introduced are:

- A `SUT Adapter` to enhance the flexibility in being able to test different systems. It acts as a layer between the test environment and DTUsat-2.
- The system has the concept of *Expressions*, corresponding to the concept of variables and arguments in programming languages. This facilitates higher reusability of components.
- `Validators` and guards are used for evaluating the received responses from the satellite. They can be combined with *Expressions* to create advanced test scenarios.

In addition to these concepts, we have also provided the ability to combine tests both sequentially and concurrently, to have a delay between tests and to have tests timeout. The creation of test scenarios can be done via a GUI, where execution of tests is also possible. Additionally, it is possible to execute tests via the command line for automated testing.

When looking back, we find it was worthwhile to develop our own test environment. It has resulted in an operational test environment, which has been tailored for the needs of the DTUsat-2 team, while at the same time keeping a high level of generality. We have presented a number of possible future improvements, which could be supplemented with the input from users actually utilizing the test environment. In this way, it is our hope that extensive testing can be performed of DTUsat-2 such that stable operation of the satellite is ensured before launch.

Bibliography

[1] René W. Fléron. *DTUsat Mission Definition Document*. Apr. 2007. URL: http://www.dtusat.dtu.dk/fileadmin/docs/DTUsat_MDD_1_21.pdf.

[2] Colin Willcock et al. *An introduction to TTCN-3*. John Wiley & Sons Ltd, 2005.

[3] *LoongTesting*. Website down at the time of writing. Fetched via archive.org. June 2012. URL: http://web.archive.org/web/20100620000343/http://ttcn.ustc.edu.cn/.

[4] *BTT - BroadBit Test Tool*. June 2012. URL: http://www.broadbit.net/portal/?page_id=392.

[5] *TestNG Website*. June 2012. URL: http://testng.org.

[6] *JSystem Website*. June 2012. URL: http://www.jsystemtest.org.

[7] *JSystem Automation Framework General Description*. June 2012. URL: http://www.jsystemtest.org/sites/default/files/help/Chapter%202%20JSystem%20Automation%20Framework%20General%20Description.htm.

[8] *Fit Customer Guide*. June 2012. URL: http://fit.c2.com/wiki.cgi?CustomerGuide.

[9] Kenneth Skovhus Andersen and Lasse Bach Nielsen. "Camera control and data handling on DTUsat". Technical University of Denmark, 2009.

Appendix A

User guide

In this appendix we will present a short guide to using the program, including where to find the source code and how to change the `SUT Adapter`.

A.1 Location of source code

The newest edition of the code for our test environment can be found in the (password protected) SVN repository located at:

`svn://dtusat.dtu.dk/Ground/trunk/GS`

Our test environment is located in the following package:

`util.nominaltest2`

The `nominaltest2` package contains the following packages:

- `gui` contains all the Java Swing code related to the GUI.
- `model` contains all logic related to the test environment.

- `runner` contains main methods.
- `scenarios` contains a few sample scenarios in the XML format.

In the `runner` package there are two main methods:

- `NominalTest2GUIRunner` contains a main method for running the GUI.
- `NominalTest2Main` contains a main method for executing the program from a command line.

An introduction to the use of the GUI can be found in section 4.4 on page 82.

Command line execution is described in section 4.6 on page 88.

A.2 Changing the `SUT Adapter`

Since the system has the option of using multiple `SUT Adapters`, it is necessary to change the `SUT Adapter` when another system is to be tested. The current `SUT Adapter` is found in `TestManager.Java` in the package `model` in the following field:

```
private static SAI sa = new SUTAdapterDummy();
```

This field could thus be changed as follows, if the `SUT Adapter` connecting to FlatSat or DTUsat is to be used instead:

```
private static SAI sa = new SUTAdapter();
```

Appendix B

XML for the CAM tests scenario

Below is the XML for a series of tests of the onboard camera, CAM. A screenshot of the program before exporting this XML can be seen in figure 4.10 on page 83.

The XML contains some tests for testing the CAM module. The tests are recreations of some of the tests found in "Camera control and data handling on DTUsat" [9]. It should be noted that the tests haven't actually been run against the CAM module, since it is not in a testable state at the time of writing.

The source code for this file can be found in scenarios/simulatorscenario.xml.

```
<Scenario id="0">
  <RootTemplates>
    <BasicTemplate id="1" name="put_in_standby"
       description="Corresponds to use_case_test.cpp,
       method put_in_standby()" cmdName="CMD_CAM_STANDBY
       " moduleID="CAM">
      <Variables/>
      <DTUSatParameterExpressions id="2"/>
      <PacketValidators>
        <PacketValidator id="3" typeName="CONF_OK"
           packetType="CONFIRM">
```

```
        <RootGuard id="4"/>
        <ExpressionValidators/>
        <Children id="5"/>
      </PacketValidator>
    </PacketValidators>
  </BasicTemplate>
  <BasicTemplate id="6" name="turn_on_picocam"
     description="Corresponds to use_case_test.cpp,
     method turn_on_picocam()" cmdName="
     CMD_CAM_POWER_IS_ON" moduleID="CAM">
    <Variables/>
    <DTUSatParameterExpressions id="7"/>
    <PacketValidators>
      <PacketValidator id="8" typeName="CONF_OK"
         packetType="CONFIRM">
        <RootGuard id="9"/>
        <ExpressionValidators/>
        <Children id="10"/>
      </PacketValidator>
    </PacketValidators>
  </BasicTemplate>
  <BasicTemplate id="11" name="turn_off_picocam"
     description="Corresponds to use_case_test.cpp,
     method turn_off_picocam()" cmdName="
     CMD_CAM_POWER_IS_OFF" moduleID="CAM">
    <Variables/>
    <DTUSatParameterExpressions id="12"/>
    <PacketValidators>
      <PacketValidator id="13" typeName="CONF_OK"
         packetType="CONFIRM">
        <RootGuard id="14"/>
        <ExpressionValidators/>
        <Children id="15"/>
      </PacketValidator>
    </PacketValidators>
  </BasicTemplate>
  <BasicTemplate id="16" name="take_picture01"
     description="Corresponds to master_use_case_test.
     cpp, method use_case_operation_take_picture01.
     trig_shut using trigger(0xf0) or shutter (0x00)"
     cmdName="CMD_CAM_TAKE_PICTURE" moduleID="CAM">
    <Variables>
      <Variable id="17" name="trig_shut"/>
      <Variable id="18" name="file_id"/>
```

```
    </Variables>
    <DTUSatParameterExpressions id="19">
      <DTUSatParameterExpression id="20" paramName="
          capture_type">
        <Variable ref="17"/>
      </DTUSatParameterExpression>
    </DTUSatParameterExpressions>
    <PacketValidators>
      <PacketValidator id="21" typeName="
          CONF_CAM_FILE_OK" packetType="CONFIRM">
        <RootGuard id="22"/>
        <ExpressionValidators>
          <ExpressionValidator id="23" paramName="
              file_id" validatorType="INT">
            <Variable ref="18"/>
          </ExpressionValidator>
        </ExpressionValidators>
        <Children id="24"/>
      </PacketValidator>
    </PacketValidators>
  </BasicTemplate>
  <DefinitionTemplate id="25" name="save_compressed"
     description="Corresponds to master_use_case_test.
     cpp, method use_case_save_compressed(). First
     takes a picture and then saves the compressed
     edition.">
    <Variables>
      <Variable id="26" name="trig_shut"/>
      <Variable id="27" name="file_id"/>
    </Variables>
    <SequentialCollectionTest id="28">
      <Tests>
        <ApplyTest id="29">
          <ApplyExpressions>
            <Variable ref="26"/>
            <Variable ref="27"/>
          </ApplyExpressions>
          <ReferenceTemplate id="30">
            <BasicTemplate ref="16"/>
          </ReferenceTemplate>
        </ApplyTest>
        <ApplyTest id="31">
          <ApplyExpressions>
            <Variable ref="27"/>
```

```
            </ApplyExpressions>
            <BasicTemplate id="32" name="
                save_compressed_picture" cmdName="
                CMD_CAM_SAVE_COMPRESSED_PICTURE"
                moduleID="CAM">
              <Variables>
                <Variable id="33" name="file_id"/>
              </Variables>
              <DTUSatParameterExpressions id="34"/>
              <PacketValidators>
                <PacketValidator id="35" typeName="
                    CONF_CAM_FILE_OK" packetType="
                    CONFIRM">
                  <RootGuard id="36"/>
                  <ExpressionValidators>
                    <ExpressionValidator id="37"
                        paramName="file_id"
                        validatorType="INT">
                      <Variable ref="33"/>
                    </ExpressionValidator>
                  </ExpressionValidators>
                  <Children id="38"/>
                </PacketValidator>
              </PacketValidators>
            </BasicTemplate>
          </ApplyTest>
        </Tests>
      </SequentialCollectionTest>
    </DefinitionTemplate>
    <BasicTemplate id="39" name="cam_alive" description="
        Corresponds to use_case_test.cpp, method alive"
        cmdName="CMD_GEN_ALIVE" moduleID="CAM">
      <Variables/>
      <DTUSatParameterExpressions id="40"/>
      <PacketValidators>
        <PacketValidator id="41" typeName="CONF_OK"
            packetType="CONFIRM">
          <RootGuard id="42"/>
          <ExpressionValidators/>
          <Children id="43"/>
        </PacketValidator>
      </PacketValidators>
    </BasicTemplate>
    <DefinitionTemplate id="44" name="
```

```
 packet_when_in_off_mode" description="Corresponds
  to master_use_case_test.cpp, method
 packet_when_in_off_mode(). Should turn off
 picocam and send a command reqeust to take a
 picture with a fixed capture type. Should receive
  error back.">
<Variables/>
<ConcurrentCollectionTest id="45">
  <Tests>
    <ApplyTest id="46">
      <ApplyExpressions/>
      <ReferenceTemplate id="47">
        <BasicTemplate ref="11"/>
      </ReferenceTemplate>
    </ApplyTest>
    <ApplyTest id="48">
      <ApplyExpressions>
        <IntArgumentExpression id="49" longValue=
            "0"/>
      </ApplyExpressions>
      <BasicTemplate id="50" name="take picture"
          cmdName="CMD_CAM_TAKE_PICTURE" moduleID=
          "CAM">
        <Variables>
          <Variable id="51" name="trig_shut"/>
        </Variables>
        <DTUSatParameterExpressions id="52">
          <DTUSatParameterExpression id="53"
              paramName="capture_type">
            <Variable ref="51"/>
          </DTUSatParameterExpression>
        </DTUSatParameterExpressions>
        <PacketValidators>
          <PacketValidator id="54" typeName="
              CONF_CAM_IN_OFF_MODE" packetType="
              CONFIRM">
            <RootGuard id="55"/>
            <ExpressionValidators/>
            <Children id="56"/>
          </PacketValidator>
        </PacketValidators>
      </BasicTemplate>
    </ApplyTest>
  </Tests>
```

```
      </ConcurrentCollectionTest>
    </DefinitionTemplate>
  </RootTemplates>
  <RootSequentialCollectionTest id="57">
    <Tests>
      <SequentialCollectionTest id="58">
        <Tests>
          <ApplyTest id="59">
            <ApplyExpressions/>
            <ReferenceTemplate id="60">
              <BasicTemplate ref="39"/>
            </ReferenceTemplate>
          </ApplyTest>
          <ApplyTest id="61">
            <ApplyExpressions>
              <IntArgumentExpression id="62" longValue=
                  "0"/>
              <IntArgumentExpression id="63" longValue=
                  "1"/>
            </ApplyExpressions>
            <ReferenceTemplate id="64">
              <BasicTemplate ref="16"/>
            </ReferenceTemplate>
          </ApplyTest>
        </Tests>
      </SequentialCollectionTest>
      <SequentialCollectionTest id="65">
        <Tests>
          <ApplyTest id="66">
            <ApplyExpressions/>
            <ReferenceTemplate id="67">
              <DefinitionTemplate ref="44"/>
            </ReferenceTemplate>
          </ApplyTest>
        </Tests>
      </SequentialCollectionTest>
      <ConcurrentCollectionTest id="68">
        <Tests>
          <ApplyTest id="69">
            <ApplyExpressions/>
            <ReferenceTemplate id="70">
              <BasicTemplate ref="11"/>
            </ReferenceTemplate>
          </ApplyTest>
```

```
            <ApplyTest id="71">
              <ApplyExpressions/>
              <ReferenceTemplate id="72">
                <BasicTemplate ref="6"/>
              </ReferenceTemplate>
            </ApplyTest>
            <ApplyTest id="73">
              <ApplyExpressions/>
              <ReferenceTemplate id="74">
                <BasicTemplate ref="1"/>
              </ReferenceTemplate>
            </ApplyTest>
            <ApplyTest id="75">
              <ApplyExpressions>
                <IntArgumentExpression id="76" longValue=
                    "0"/>
                <IntArgumentExpression id="77" longValue=
                    "2"/>
              </ApplyExpressions>
              <ReferenceTemplate id="78">
                <DefinitionTemplate ref="25"/>
              </ReferenceTemplate>
            </ApplyTest>
          </Tests>
        </ConcurrentCollectionTest>
      </Tests>
    </RootSequentialCollectionTest>
</Scenario>
```

Appendix C

XML for the flash simulator scenario

Below a scenario for testing the flash simulator can be found.

The source code for this file can be found in scenarios/simulatorscenario.xml.

```
<Scenario id="0">
  <RootTemplates>
    <BasicTemplate id="1" name="erase" cmdName="
       CMD_FLA_ERASE_BLOCKS" moduleID="FLA">
      <Variables>
        <Variable id="2" name="max_blocks"/>
        <Variable id="3" name="actual_blocks"/>
      </Variables>
      <DTUSatParameterExpressions id="4">
        <DTUSatParameterExpression id="5" paramName="
           max_blocks">
          <Variable ref="2"/>
        </DTUSatParameterExpression>
      </DTUSatParameterExpressions>
      <PacketValidators>
        <PacketValidator id="6" typeName="
           CONF_FLA_ERASE_OK" packetType="CONFIRM">
```

```
    <RootGuard id="7">
      <AND id="8">
        <GTE id="9">
          <Expression1>
            <Variable ref="2"/>
          </Expression1>
          <Expression2>
            <IntArgumentExpression id="10"
                longValue="1"/>
          </Expression2>
        </GTE>
        <LTE id="11">
          <Expression1>
            <Variable ref="2"/>
          </Expression1>
          <Expression2>
            <IntArgumentExpression id="12"
                longValue="62"/>
          </Expression2>
        </LTE>
      </AND>
    </RootGuard>
    <ExpressionValidators>
      <ExpressionValidator id="13" paramName="
          blocks_erased" validatorType="INT">
        <Variable ref="3"/>
      </ExpressionValidator>
    </ExpressionValidators>
    <Children id="14"/>
  </PacketValidator>
  <PacketValidator id="15" typeName="
      CONF_FLA_FLASH_ERROR" packetType="CONFIRM">
    <RootGuard id="16">
      <OR id="17">
        <GT id="18">
          <Expression1>
            <Variable ref="2"/>
          </Expression1>
          <Expression2>
            <IntArgumentExpression id="19"
                longValue="62"/>
          </Expression2>
        </GT>
        <LT id="20">
```

```
                <Expression1>
                  <Variable ref="2"/>
                </Expression1>
                <Expression2>
                  <IntArgumentExpression id="21"
                      longValue="1"/>
                </Expression2>
              </LT>
            </OR>
          </RootGuard>
          <ExpressionValidators>
            <ExpressionValidator id="22" paramName="code
                " validatorType="ENUM">
              <EnumArgumentExpression id="23" longValue
                  ="-1"/>
            </ExpressionValidator>
          </ExpressionValidators>
          <Children id="24"/>
        </PacketValidator>
      </PacketValidators>
    </BasicTemplate>
    <DefinitionTemplate id="25" name="maxEqualsActual">
      <Variables>
        <Variable id="26" name="blocks"/>
      </Variables>
      <ApplyTest id="27">
        <ApplyExpressions>
          <Variable ref="26"/>
          <Variable ref="26"/>
        </ApplyExpressions>
        <ReferenceTemplate id="28">
          <BasicTemplate ref="1"/>
        </ReferenceTemplate>
      </ApplyTest>
    </DefinitionTemplate>
  </RootTemplates>
  <RootSequentialCollectionTest id="29">
    <Tests>
      <SequentialCollectionTest id="30">
        <Tests>
          <ConcurrentCollectionTest id="31">
            <Tests>
              <DelayTest id="32" wait="5000">
                <ApplyTest id="33">
```

```
            <ApplyExpressions>
              <IntArgumentExpression id="34"
                  longValue="1"/>
            </ApplyExpressions>
            <ReferenceTemplate id="35">
              <DefinitionTemplate ref="25"/>
            </ReferenceTemplate>
          </ApplyTest>
        </DelayTest>
        <ApplyTest id="36">
          <ApplyExpressions>
            <IntArgumentExpression id="37"
                longValue="3"/>
          </ApplyExpressions>
          <ReferenceTemplate id="38">
            <DefinitionTemplate ref="25"/>
          </ReferenceTemplate>
        </ApplyTest>
      </Tests>
    </ConcurrentCollectionTest>
    <ForTest id="39" upwards="true">
      <StartValue>
        <IntArgumentExpression id="40" longValue=
            "0"/>
      </StartValue>
      <ToValue>
        <IntArgumentExpression id="41" longValue=
            "6"/>
      </ToValue>
      <StepValue>
        <IntArgumentExpression id="42" longValue=
            "1"/>
      </StepValue>
      <ApplyTest id="43">
        <ApplyExpressions>
          <Variable id="44" name="For Loop
              Parameter"/>
        </ApplyExpressions>
        <ReferenceTemplate id="45">
          <DefinitionTemplate ref="25"/>
        </ReferenceTemplate>
      </ApplyTest>
      <Variable ref="44"/>
    </ForTest>
```

```
            <ApplyTest id="46">
              <ApplyExpressions>
                <IntArgumentExpression id="47" longValue=
                    "15"/>
                <IntArgumentExpression id="48" longValue=
                    "7"/>
              </ApplyExpressions>
              <ReferenceTemplate id="49">
                <BasicTemplate ref="1"/>
              </ReferenceTemplate>
            </ApplyTest>
          </Tests>
        </SequentialCollectionTest>
      </Tests>
    </RootSequentialCollectionTest>
</Scenario>
```

Appendix D

Source code for the model

In this appendix all source code from the `model` package can be found.

errors/ErrorValue.java

```
1  package util.nominaltest2.model.errors;
2
3  import util.nominaltest2.model.TreeItem;
4
5  public class ErrorValue {
6
7      String info;
8      TreeItem treeItem;
9
10     public ErrorValue(String info, TreeItem relatedTreeItem) {
11         this.info = info;
12         this.treeItem = relatedTreeItem;
13     }
14
15     @Override
16     public String toString() {
17         return info + ": " + treeItem.toString();
18     }
19
20     public TreeItem getTreeItem() {
21         return treeItem;
22     }
23 }
```

exceptions/FailedTestException.java

```
1  package util.nominaltest2.model.exceptions;
2
3  import util.nominaltest2.model.test.Test;
4  import common.dtusat2.packet.Packet;
5
6  public class FailedTestException extends Exception {
7
8      private Test test;
9      private Packet packet;
10
11     public FailedTestException(Test test, Packet packet) {
12         this.test = test;
13         this.packet = packet;
14     }
15
16     public String getDescription() {
17
18         return "The test got a packet of type " + packet.
               getPacketType()
19                 + " which was not expected";
20     }
21 }
```

expression/intervalExpression/ IntIntervalExpression.java

```
1  package util.nominaltest2.model.expression.intervalExpression;
2
3  import org.simpleframework.xml.Element;
4  import org.simpleframework.xml.ElementUnion;
5  import org.simpleframework.xml.Path;
6
7  import common.types.argument.Argument;
8  import common.types.argument.IntArgument;
9
10 import util.nominaltest2.model.expression.Expression;
11 import util.nominaltest2.model.expression.SingleExpression;
12 import util.nominaltest2.model.expression.singleExpressions.
       ArgumentSingleExpression;
13 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
14 import util.nominaltest2.model.expression.singleExpressions.
       argument.EnumArgumentSingleExpression;
15 import util.nominaltest2.model.expression.singleExpressions.
       argument.IntArgumentSingleExpression;
16 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.MinusArithmeticSingleExpression;
17 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.MultiplicationArithmeticSingleExpression;
18 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.PlusArithmeticSingleExpression;
19
20 public class IntIntervalExpression extends Expression {
21    public static final String typeName = "Int Interval";
22
23    @Path("Expression1")
24    @ElementUnion( {
25          @Element(name = "EnumArgumentExpression", type =
                EnumArgumentSingleExpression.class, required = false),
26          @Element(name = "IntArgumentExpression", type =
                IntArgumentSingleExpression.class, required = false),
27          @Element(name = "Variable", type = Variable.class,
                required = false),
28          @Element(name = "MinusArithmeticSingleExpression", type =
                MinusArithmeticSingleExpression.class, required =
                false),
29          @Element(name = "MultiplicationArithmeticSingleExprssion",
                 type = MultiplicationArithmeticSingleExpression.class
                , required = false),
30          @Element(name = "PlusArithmeticSingleExpression", type =
                PlusArithmeticSingleExpression.class, required = false
                ) })
31    SingleExpression expr1;
32
33    @Path("Expression2")
34    @ElementUnion( {
```

```
35          @Element(name = "EnumArgumentExpression", type =
                EnumArgumentSingleExpression.class, required = false),
36          @Element(name = "IntArgumentExpression", type =
                IntArgumentSingleExpression.class, required = false),
37          @Element(name = "Variable", type = Variable.class,
                required = false),
38          @Element(name = "MinusArithmeticSingleExpression", type =
                MinusArithmeticSingleExpression.class, required =
                false),
39          @Element(name = "MultiplicationArithmeticSingleExprssion",
                 type = MultiplicationArithmeticSingleExpression.class
                , required = false),
40          @Element(name = "PlusArithmeticSingleExpression", type =
                PlusArithmeticSingleExpression.class, required = false
                ) })
41      SingleExpression expr2;
42
43      public void setLowerLimit(SingleExpression singleExpression) {
44          expr1 = singleExpression;
45      }
46
47      public SingleExpression getLowerLimit() {
48          return expr1;
49      }
50
51      public void setUpperLimit(SingleExpression singleExpression) {
52          expr2 = singleExpression;
53      }
54
55      public SingleExpression getUpperLimit() {
56          return expr2;
57      }
58
59      @Override
60      public IntIntervalExpression clone() {
61          IntIntervalExpression clonedSelf = new IntIntervalExpression
                ();
62          if (expr1 != null) {
63              clonedSelf.setLowerLimit((SingleExpression) expr1.clone())
                    ;
64          }
65          if (expr2 != null) {
66              clonedSelf.setUpperLimit((SingleExpression) expr2.clone())
                    ;
67          }
68          return clonedSelf;
69      }
70
71      @Override
72      public String getTypeName() {
73          return typeName;
74      }
75
76      @Override
77      public boolean evaluateAgainstArgument(Argument arg) {
```

```
78          if (arg instanceof IntArgument && expr1 != null && expr2 !=
                null) {
79              Long value = ((IntArgument) arg).getValue();
80              SingleExpression lowerLimit = expr1.getValue();
81              SingleExpression upperLimit = expr2.getValue();
82              if (lowerLimit instanceof IntArgumentSingleExpression
83                      && upperLimit instanceof IntArgumentSingleExpression
                            ) {
84                  if (((IntArgumentSingleExpression) lowerLimit).
                        getLongValue() <= value
85                          && ((IntArgumentSingleExpression) upperLimit)
86                                  .getLongValue() >= value) {
87                      return true;
88                  } else if (((IntArgumentSingleExpression) lowerLimit)
89                          .getLongValue() >= value
90                          && ((IntArgumentSingleExpression) upperLimit)
91                                  .getLongValue() <= value) {
92                      return true;
93                  } else {
94                      return false;
95                  }
96              }
97          }
98          return false;
99      }
100
101     /**
102      * Return the {@link IntIntervalExpression} as a string,
            representing the
103      * interval it spans over.
104      *
105      * @return
106      */
107     public String getValues() {
108         if (expr1 != null && expr2 != null) {
109             Expression lowerLimit = expr1.getValue();
110             Expression upperLimit = expr2.getValue();
111
112             if (lowerLimit instanceof ArgumentSingleExpression
113                     && upperLimit instanceof ArgumentSingleExpression) {
114                 long lower = ((ArgumentSingleExpression) lowerLimit)
115                         .getLongValue();
116                 long upper = ((ArgumentSingleExpression) upperLimit)
117                         .getLongValue();
118                 return lower + " - " + upper;
119             }
120         }
121         return "One of the values is not bounded";
122     }
123
124     @Override
125     public boolean isBounded() {
126         if (expr1.isBounded() && expr2.isBounded()) {
127             return true;
128         }
```

```
129         return false;
130     }
131 
132     @Override
133     public String getStringValue() {
134         return getValues();
135     }
136 
137 }
```

expression/singleExpressions/argument /EnumArgumentSingleExpression.java

```
1  package util.nominaltest2.model.expression.singleExpressions.
       argument;
2  
3  import org.simpleframework.xml.Attribute;
4  
5  import util.nominaltest2.model.expression.SingleExpression;
6  
7  public class EnumArgumentSingleExpression extends
       IntArgumentSingleExpression {
8  
9      // We will use that integer that a enum has on it to identify it
           .
10 
11     public static final String typeName = "Enum Argument";
12 
13     @Override
14     public String getTypeName() {
15         return typeName;
16     }
17 
18     public EnumArgumentSingleExpression(@Attribute(name = "longValue
           ") long i) {
19         super(i);
20     }
21 
22     @Override
23     public SingleExpression clone() {
24         return new EnumArgumentSingleExpression(dtuSatArg.getValue())
               ;
25     }
26 
27     @Override
28     public boolean isBounded() {
29         return true;
30     }
31 }
```

expression/singleExpressions/argument /IntArgumentSingleExpression.java

```java
package util.nominaltest2.model.expression.singleExpressions.argument;

import org.simpleframework.xml.Attribute;

import util.nominaltest2.model.expression.SingleExpression;
import util.nominaltest2.model.expression.singleExpressions.ArgumentSingleExpression;

import common.types.argument.Argument;
import common.types.argument.IntArgument;

public class IntArgumentSingleExpression extends ArgumentSingleExpression {
    public static final String typeName = "Int Argument";

    @Override
    public String getTypeName() {
        return typeName;
    }

    // We save the long-value locally to ease serialization with simple xml
    // Assumes that one can't change values when an IntArgument has been
    // instantiated (true at the time of writing)
    @Attribute
    long longValue;

    IntArgument dtuSatArg;

    public IntArgumentSingleExpression(@Attribute(name = "longValue") long i) {
        this.longValue = i;
        dtuSatArg = new IntArgument(i);
    }

    @Override
    public SingleExpression clone() {
        return new IntArgumentSingleExpression(dtuSatArg.getValue());
    }

    @Override
    public IntArgument getDTUSatArgument() {
        return dtuSatArg;
    }

    @Override
    public long getLongValue() {
        return dtuSatArg.getValue();
```

```
45      }
46
47      @Override
48      public boolean evaluateAgainstArgument(Argument arg) {
49          if (arg instanceof IntArgument) {
50              return dtuSatArg.getValue() == ((IntArgument) arg).
                    getValue();
51          }
52          return false;
53      }
54
55      @Override
56      public boolean isBounded() {
57          return true;
58      }
59
60      @Override
61      public String getStringValue() {
62          return "Integer with value of = " + longValue;
63      }
64  }
```

expression/singleExpressions/arithmetic /ArithmeticSingleExpression.java

```
1   package util.nominaltest2.model.expression.singleExpressions.
        arithmetic;
2
3   import org.simpleframework.xml.Element;
4   import org.simpleframework.xml.ElementUnion;
5   import org.simpleframework.xml.Path;
6
7   import common.types.argument.Argument;
8   import common.types.argument.IntArgument;
9
10  import util.nominaltest2.model.expression.Expression;
11  import util.nominaltest2.model.expression.SingleExpression;
12  import util.nominaltest2.model.expression.singleExpressions.
        ArgumentSingleExpression;
13  import util.nominaltest2.model.expression.singleExpressions.
        Variable;
14  import util.nominaltest2.model.expression.singleExpressions.
        argument.EnumArgumentSingleExpression;
15  import util.nominaltest2.model.expression.singleExpressions.
        argument.IntArgumentSingleExpression;
16
17  public abstract class ArithmeticSingleExpression extends
        SingleExpression {
18
19      @Path("Expression1")
```

```
20      @ElementUnion({
21              @Element(name = "EnumArgumentExpression", type =
                    EnumArgumentSingleExpression.class, required = false),
22              @Element(name = "IntArgumentExpression", type =
                    IntArgumentSingleExpression.class, required = false),
23              @Element(name = "Variable", type = Variable.class,
                    required = false),
24              @Element(name = "MinusArithmeticSingleExpression", type =
                    MinusArithmeticSingleExpression.class, required =
                    false),
25              @Element(name = "MultiplicationArithmeticSingleExprssion",
                     type = MultiplicationArithmeticSingleExpression.class
                    , required = false),
26              @Element(name = "PlusArithmeticSingleExpression", type =
                    PlusArithmeticSingleExpression.class, required = false
                    ) })
27      SingleExpression expr1;
28
29      @Path("Expression2")
30      @ElementUnion({
31              @Element(name = "EnumArgumentExpression", type =
                    EnumArgumentSingleExpression.class, required = false),
32              @Element(name = "IntArgumentExpression", type =
                    IntArgumentSingleExpression.class, required = false),
33              @Element(name = "Variable", type = Variable.class,
                    required = false),
34              @Element(name = "MinusArithmeticSingleExpression", type =
                    MinusArithmeticSingleExpression.class, required =
                    false),
35              @Element(name = "MultiplicationArithmeticSingleExprssion",
                     type = MultiplicationArithmeticSingleExpression.class
                    , required = false),
36              @Element(name = "PlusArithmeticSingleExpression", type =
                    PlusArithmeticSingleExpression.class, required = false
                    ) })
37      SingleExpression expr2;
38
39      public ArithmeticSingleExpression(SingleExpression expr1,
40              SingleExpression expr2) {
41          this.expr1 = expr1;
42          this.expr2 = expr2;
43      }
44
45      @Override
46      public boolean isBounded() {
47          return expr1.isBounded() && expr2.isBounded();
48      }
49
50      /**
51       * Will return the evaluated value for the {@link
                ArithmeticSingleExpression}
52       * @return
53       */
54      public abstract Long getLongValue();
55
```

```
56      /**
57       * Will return the left and rights sides expressed as a long
             value.
58       * @return
59       */
60      protected long[] getLongValues() {
61          SingleExpression e1value = expr1.getValue();
62          SingleExpression e2value = expr2.getValue();
63          if (e1value instanceof ArgumentSingleExpression
64                  && e2value instanceof ArgumentSingleExpression) {
65              ArgumentSingleExpression e1arg = (ArgumentSingleExpression
                        ) e1value;
66              ArgumentSingleExpression e2arg = (ArgumentSingleExpression
                        ) e2value;
67              return new long[] { e1arg.getLongValue(), e2arg.
                        getLongValue() };
68          }
69          return null;
70      }
71
72      @Override
73      public SingleExpression getValue() {
74          Long thisValue = getLongValue();
75          if (thisValue != null)
76              return new IntArgumentSingleExpression(thisValue);
77          return new Variable(
78                  "ArithmeticSingleExpression not bound");
79      }
80
81      @Override
82      public boolean evaluateAgainstArgument(Argument arg) {
83          Long thisValue = getLongValue();
84          if (thisValue != null) {
85              if (arg instanceof IntArgument) {
86                  IntArgument dtuIntArg = (IntArgument) arg;
87                  return dtuIntArg.getValue() == thisValue;
88              }
89          }
90          return false;
91      }
92  }
```

expression/singleExpressions/arithmetic /MinusArithmeticSingleExpression.java

```
1  package util.nominaltest2.model.expression.singleExpressions.
       arithmetic;
2
3  import util.nominaltest2.model.expression.Expression;
4  import util.nominaltest2.model.expression.SingleExpression;
```

```
5
6  public class MinusArithmeticSingleExpression extends
       ArithmeticSingleExpression {
7      public MinusArithmeticSingleExpression(SingleExpression expr1,
8              SingleExpression expr2) {
9          super(expr1, expr2);
10     }
11
12     public static final String typeName = "Minus";
13
14     @Override
15     public String getTypeName() {
16         return typeName;
17     }
18
19     @Override
20     public Long getLongValue() {
21         long[] values = getLongValues();
22         if (values != null) {
23             return values[0] - values[1];
24         }
25         return null;
26     }
27
28     @Override
29     public Expression clone() {
30         return new MinusArithmeticSingleExpression((SingleExpression)
               expr1.clone(), (SingleExpression)expr2.clone());
31     }
32
33     @Override
34     public String getStringValue() {
35         return "" + getLongValue();
36     }
37 }
```

expression/singleExpressions/arithmetic /MultiplicationArithmeticSingleExpression.java

```
1  package util.nominaltest2.model.expression.singleExpressions.
       arithmetic;
2
3  import util.nominaltest2.model.expression.Expression;
4  import util.nominaltest2.model.expression.SingleExpression;
5
6  public class MultiplicationArithmeticSingleExpression extends
7          ArithmeticSingleExpression {
8      public MultiplicationArithmeticSingleExpression(SingleExpression
           expr1,
9              SingleExpression expr2) {
```

```
10        super(expr1, expr2);
11    }
12
13    public static final String typeName = "Multiplication";
14
15    @Override
16    public String getTypeName() {
17        return typeName;
18    }
19
20    @Override
21    public Long getLongValue() {
22        long[] values = getLongValues();
23        if (values != null) {
24            return values[0] * values[1];
25        }
26        return null;
27    }
28
29    @Override
30    public Expression clone() {
31        return new MultiplicationArithmeticSingleExpression((
              SingleExpression)expr1.clone(), (SingleExpression)expr2.
              clone());
32    }
33
34    @Override
35    public String getStringValue() {
36        return "" + getLongValue();
37    }
38 }
```

expression/singleExpressions/arithmetic /PlusArithmeticSingleExpression.java

```
1  package util.nominaltest2.model.expression.singleExpressions.
       arithmetic;
2
3  import util.nominaltest2.model.expression.Expression;
4  import util.nominaltest2.model.expression.SingleExpression;
5
6  public class PlusArithmeticSingleExpression extends
       ArithmeticSingleExpression {
7      public static final String typeName = "Plus";
8
9      public PlusArithmeticSingleExpression(SingleExpression expr1,
10           SingleExpression expr2) {
11         super(expr1, expr2);
12     }
13
```

```
14      @Override
15      public String getTypeName() {
16          return typeName;
17      }
18
19      @Override
20      public Long getLongValue() {
21          long[] values = getLongValues();
22          if (values != null) {
23              return values[0] + values[1];
24          }
25          return null;
26      }
27
28      @Override
29      public Expression clone() {
30          return new PlusArithmeticSingleExpression((SingleExpression)
                expr1.clone(), (SingleExpression)expr2.clone());
31      }
32
33      @Override
34      public String getStringValue() {
35          return "" + getLongValue();
36      }
37  }
```

expression/singleExpressions /ArgumentSingleExpression.java

```
1   package util.nominaltest2.model.expression.singleExpressions;
2
3   import util.nominaltest2.model.expression.Expression;
4   import util.nominaltest2.model.expression.SingleExpression;
5   import common.types.argument.Argument;
6
7   public abstract class ArgumentSingleExpression extends
        SingleExpression {
8
9       /**
10      * Gets the {@link Argument} associated with the {@link
            Expression}
11      * @return
12      */
13      public abstract Argument getDTUSatArgument();
14
15      @Override
16      public SingleExpression getValue() {
17          return this;
18      }
19
```

```
20      /**
21       * Gets the exact value of the Argument in the form of a long,
             if the
22       * current {@link Expression} supports this.
23       *
24       * @return
25       */
26      public abstract long getLongValue();
27
28  }
```

expression/singleExpressions /Variable.java

```
1   package util.nominaltest2.model.expression.singleExpressions;
2
3   import org.simpleframework.xml.Attribute;
4
5   import common.types.argument.Argument;
6
7   import util.nominaltest2.model.expression.Expression;
8   import util.nominaltest2.model.expression.SingleExpression;
9
10  public class Variable extends SingleExpression implements Cloneable
        {
11
12      @Attribute
13      protected String name = "";
14
15      // @Attribute(required=false)
16      SingleExpression parentExpr;
17
18      public static final String typeName = "Parameter Expression";
19
20      @Override
21      public String getTypeName() {
22          return typeName;
23      }
24
25      public Variable(@Attribute(name = "name") String name) {
26          this.name = name;
27      }
28
29      /**
30       * Sets the referenced {@link Expression}
31       * @param parentExpr
32       */
33      public void setParentExpr(SingleExpression parentExpr) {
34          this.parentExpr = parentExpr;
35          this.addObserver(parentExpr);
```

```
36      }
37
38      @Override
39      public Variable clone() {
40          Variable expr = new Variable(
41                  new String(name));
42          if (parentExpr != null) {
43              expr.setParentExpr((SingleExpression) parentExpr.clone());
44          }
45          return expr;
46      }
47
48      @Override
49      public SingleExpression getValue() {
50          if (parentExpr != null) {
51              return parentExpr.getValue();
52          }
53          return this;
54      }
55
56      @Override
57      public String toString() {
58          return (name == null || name.equals("") ? "<unnamed>" : name)
                ;
59      }
60
61      @Override
62      public boolean evaluateAgainstArgument(Argument arg) {
63          if (parentExpr != null) {
64              return parentExpr.evaluateAgainstArgument(arg);
65          }
66          return false;
67      }
68
69      @Override
70      public boolean isBounded() {
71          return parentExpr==null?false:parentExpr.isBounded();
72      }
73
74      public String getName() {
75          return name;
76      }
77
78      public void setName(String name) {
79          this.name = name;
80          setChanged();
81          notifyObservers();
82      }
83
84      @Override
85      public String getStringValue() {
86          if (parentExpr != null) {
87              return parentExpr.getStringValue();
88          }
89          return null;
```

```
90     }
91 }
```

expression/Expression.java

```
1  package util.nominaltest2.model.expression;
2
3  import java.util.Observable;
4
5  import common.types.argument.Argument;
6
7  public abstract class Expression extends Observable {
8      @Override
9      public abstract Expression clone();
10
11     public abstract String getTypeName();
12
13     public abstract String getStringValue();
14
15     /**
16      * Will evaluate the {@link Argument} given towards the expected
            .
17      *
18      * @param arg
19      * @return <b>True</b> in case we received what is expected.
20      */
21     public abstract boolean evaluateAgainstArgument(Argument arg);
22
23     /**
24      * Check is the Expression is bounded, meaning it has an exact
            value.
25      * @return <b>True</b> if it is bounded.
26      */
27     public abstract boolean isBounded();
28
29 }
```

expression/SingleExpression.java

```
1  package util.nominaltest2.model.expression;
2
3  import java.util.Observable;
4  import java.util.Observer;
5
6  import util.nominaltest2.model.expression.singleExpressions.
       ArgumentSingleExpression;
```

```
7
8  public abstract class SingleExpression extends Expression
       implements Observer {
9
10     /**
11      * Will return the referenced Expression, or it self if it is an
            instance of
12      * {@link ArgumentSingleExpression}
13      *
14      * @return
15      */
16     public abstract SingleExpression getValue();
17
18
19
20     @Override
21     public void update(Observable o, Object arg) {
22         setChanged();
23         notifyObservers(arg);
24     }
25 }
```

guard/booleanguards/BooleanGuard.java

```
1  package util.nominaltest2.model.guard.booleanguards;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  import util.nominaltest2.model.TreeItem;
7  import util.nominaltest2.model.errors.ErrorValue;
8  import util.nominaltest2.model.expression.singleExpressions.
       Variable;
9  import util.nominaltest2.model.guard.Guard;
10 import util.nominaltest2.model.template.BasicTemplate;
11
12 public abstract class BooleanGuard extends Guard {
13     @Override
14     public ArrayList<TreeItem> getChildren() {
15         return new ArrayList<TreeItem>();
16     }
17
18     @Override
19     protected void doAddChild(TreeItem item) throws Exception {
20         noChildrenException();
21
22     }
23
24     @Override
25     protected void doDeleteChild(TreeItem child) throws Exception {
26         noChildrenException();
```

```
27
28      }
29
30      private void noChildrenException() throws Exception {
31          throw new Exception("A BooleanGuard doesn't have any children
                ");
32      }
33
34      @Override
35      public boolean isVariableInUse(
36              Variable variable) {
37          return false;
38      }
39
40      @Override
41      public void validate(List<ErrorValue> errorList, BasicTemplate
            basicTemplate) {
42          // Nothing to validate
43      }
44
45      @Override
46      public void setupObserverObservableHierarchy() {
47          // TODO Auto-generated method stub
48
49      }
50
51  }
```

guard/booleanguards /FalseBooleanGuard.java

```
1   package util.nominaltest2.model.guard.booleanguards;
2
3   import java.util.ArrayList;
4
5   import util.nominaltest2.model.expression.singleExpressions.
        Variable;
6   import util.nominaltest2.model.guard.Guard;
7
8   public class FalseBooleanGuard extends BooleanGuard {
9       public static final String typeName = "False";
10
11      @Override
12      public String getTypeName() {
13          return typeName;
14      }
15
16      @Override
17      public boolean evaluate() {
18          return false;
```

```
19      }
20
21      @Override
22      public Guard clone() {
23          return new FalseBooleanGuard();
24      }
25
26      @Override
27      public void updateExpression(ArrayList<Variable> variables) {
28      }
29
30  }
```

guard/booleanguards /TrueBooleanGuard.java

```
1   package util.nominaltest2.model.guard.booleanguards;
2
3   import java.util.ArrayList;
4
5   import util.nominaltest2.model.expression.singleExpressions.
        Variable;
6   import util.nominaltest2.model.guard.Guard;
7
8   public class TrueBooleanGuard extends BooleanGuard {
9       public static final String typeName = "True";
10
11      @Override
12      public String getTypeName() {
13          return typeName;
14      }
15
16      @Override
17      public boolean evaluate() {
18          return true;
19      }
20
21      @Override
22      public Guard clone() {
23          return new TrueBooleanGuard();
24      }
25
26      @Override
27      public void updateExpression(ArrayList<Variable> variables) {
28       }
29  }
```

guard/logicalguards /comparelogicalguards /ANDCompareLogicalGuard.java

```java
package util.nominaltest2.model.guard.logicalguards.comparelogicalguards;

import java.util.logging.Logger;

import util.nominaltest2.model.guard.Guard;

public class ANDCompareLogicalGuard extends CompareLogicalGuard {
    private final Logger logger = Logger.getLogger(getClass().getName());

    public static final String typeName = "AND";

    @Override
    public String getTypeName() {
        return typeName;
    }

    @Override
    public boolean evaluate() throws Exception {
        boolean finalRes = true;
        for (Guard guard : guards) {
            finalRes = finalRes && guard.evaluate();
            if (!finalRes)
                return false;
        }
        return finalRes; // true
    }

    @Override
    public Guard clone() {
        ANDCompareLogicalGuard clonedSelf = new ANDCompareLogicalGuard();
        try {
            for (Guard guard : guards) {
                clonedSelf.addChild(guard.clone());
            }
        } catch (Exception e) {
            logger.severe(e.getMessage());
        }
        return clonedSelf;
    }

}
```

guard/logicalguards /comparelogicalguards /CompareLogicalGuard.java

```
1  package util.nominaltest2.model.guard.logicalguards.
       comparelogicalguards;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  import org.simpleframework.xml.Element;
7  import org.simpleframework.xml.ElementListUnion;
8  import org.simpleframework.xml.ElementList;
9  import org.simpleframework.xml.ElementUnion;
10
11 import util.nominaltest2.model.TreeItem;
12 import util.nominaltest2.model.errors.ErrorValue;
13 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
14 import util.nominaltest2.model.guard.Guard;
15 import util.nominaltest2.model.guard.RootGuard;
16 import util.nominaltest2.model.guard.booleanguards.
       FalseBooleanGuard;
17 import util.nominaltest2.model.guard.booleanguards.TrueBooleanGuard
       ;
18 import util.nominaltest2.model.guard.logicalguards.LogicalGuard;
19 import util.nominaltest2.model.guard.logicalguards.NOTLogicalGuard;
20 import util.nominaltest2.model.guard.relationalguards.
       EQRelationalGuard;
21 import util.nominaltest2.model.guard.relationalguards.
       GTERelationalGuard;
22 import util.nominaltest2.model.guard.relationalguards.
       GTRelationalGuard;
23 import util.nominaltest2.model.guard.relationalguards.
       LTERelationalGuard;
24 import util.nominaltest2.model.guard.relationalguards.
       LTRelationalGuard;
25 import util.nominaltest2.model.guard.relationalguards.
       NEQRelationalGuard;
26 import util.nominaltest2.model.template.BasicTemplate;
27
28 public abstract class CompareLogicalGuard extends LogicalGuard {
29
30     @ElementListUnion({
31             @ElementList(entry = "True", type = TrueBooleanGuard.class
                   , inline = true, required = false),
32             @ElementList(entry = "False", type = FalseBooleanGuard.
                   class, inline = true, required = false),
33             @ElementList(entry = "EQ", type = EQRelationalGuard.class,
                    inline = true, required = false),
34             @ElementList(entry = "GT", type = GTRelationalGuard.class,
                    inline = true, required = false),
```

```
35          @ElementList(entry = "GTE", type = GTERelationalGuard.
                class, inline = true, required = false),
36          @ElementList(entry = "LT", type = LTRelationalGuard.class,
                 inline = true, required = false),
37          @ElementList(entry = "LTE", type = LTERelationalGuard.
                class, inline = true, required = false),
38          @ElementList(entry = "NEQ", type = NEQRelationalGuard.
                class, inline = true, required = false),
39          @ElementList(entry = "AND", type = ANDCompareLogicalGuard.
                class, inline = true, required = false),
40          @ElementList(entry = "OR", type = ORCompareLogicalGuard.
                class, inline = true, required = false),
41          @ElementList(entry = "NOT", type = NOTLogicalGuard.class,
                inline = true, required = false),
42          @ElementList(entry = "RootGuard", type = RootGuard.class,
                inline = true, required = false) })
43      ArrayList<Guard> guards = new ArrayList<Guard>();
44
45      @Override
46      public ArrayList<TreeItem> getChildren() {
47
48          return new ArrayList<TreeItem>(guards);
49      }
50
51      @Override
52      protected void doAddChild(TreeItem item) throws Exception {
53          if (isGuard(item)) {
54              guards.add((Guard) item);
55          }
56      }
57
58      public void setupObserverObservableHierarchy() {
59          for (Guard g : guards) {
60              g.addObserver(this);
61              g.setupObserverObservableHierarchy();
62          }
63
64      }
65
66      @Override
67      protected void doDeleteChild(TreeItem child) throws Exception {
68          guards.remove(child);
69
70      }
71
72      @Override
73      public void updateExpression(ArrayList<Variable> variables) {
74          for (Guard guard : guards) {
75              guard.updateExpression(variables);
76          }
77      }
78
79      @Override
80      public boolean isVariableInUse(
81              Variable variable) {
```

```
82          for (Guard guard : guards) {
83              if (guard.isVariableInUse(variable)) {
84                  return true;
85              }
86          }
87          return false;
88      }
89
90      @Override
91      public void validate(List<ErrorValue> errorList, BasicTemplate
            basicTemplate) {
92          for (Guard g : guards) {
93              g.validate(errorList, basicTemplate);
94          }
95      }
96  }
```

guard/logicalguards /comparelogicalguards /ORCompareLogicalGuard.java

```
1  package util.nominaltest2.model.guard.logicalguards.
       comparelogicalguards;
2
3  import java.util.logging.Logger;
4
5  import util.nominaltest2.model.guard.Guard;
6
7  public class ORCompareLogicalGuard extends CompareLogicalGuard {
8      private final Logger logger = Logger.getLogger(getClass().
           getName());
9
10     public static final String typeName = "OR";
11
12     @Override
13     public String getTypeName() {
14         return typeName;
15     }
16
17     @Override
18     public boolean evaluate() throws Exception {
19         boolean finalRes = false;
20         for (Guard guard : guards) {
21             finalRes = finalRes || guard.evaluate();
22             if (finalRes)
23                 return true;
24         }
25         return finalRes; // false
26     }
```

```
27
28      @Override
29      public Guard clone() {
30          ORCompareLogicalGuard clonedSelf = new ORCompareLogicalGuard
                ();
31          try {
32              for (Guard guard : guards) {
33                  clonedSelf.addChild(guard.clone());
34              }
35          } catch (Exception e) {
36              logger.severe(e.getMessage());
37          }
38          return clonedSelf;
39      }
40  }
```

guard/logicalguards/LogicalGuard.java

```
1  package util.nominaltest2.model.guard.logicalguards;
2
3  import util.nominaltest2.model.guard.Guard;
4
5  public abstract class LogicalGuard extends Guard {
6
7  }
```

guard/logicalguards /NOTLogicalGuard.java

```
1  package util.nominaltest2.model.guard.logicalguards;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.logging.Logger;
6
7  import org.simpleframework.xml.Element;
8  import org.simpleframework.xml.ElementUnion;
9
10 import util.nominaltest2.model.TreeItem;
11 import util.nominaltest2.model.errors.ErrorValue;
12 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
13 import util.nominaltest2.model.guard.Guard;
14 import util.nominaltest2.model.guard.RootGuard;
15 import util.nominaltest2.model.guard.booleanguards.
       FalseBooleanGuard;
```

```
16 import util.nominaltest2.model.guard.booleanguards.TrueBooleanGuard
       ;
17 import util.nominaltest2.model.guard.logicalguards.
       comparelogicalguards.ANDCompareLogicalGuard;
18 import util.nominaltest2.model.guard.logicalguards.
       comparelogicalguards.ORCompareLogicalGuard;
19 import util.nominaltest2.model.guard.relationalguards.
       EQRelationalGuard;
20 import util.nominaltest2.model.guard.relationalguards.
       GTERelationalGuard;
21 import util.nominaltest2.model.guard.relationalguards.
       GTRelationalGuard;
22 import util.nominaltest2.model.guard.relationalguards.
       LTERelationalGuard;
23 import util.nominaltest2.model.guard.relationalguards.
       LTRelationalGuard;
24 import util.nominaltest2.model.guard.relationalguards.
       NEQRelationalGuard;
25 import util.nominaltest2.model.template.BasicTemplate;
26
27 public class NOTLogicalGuard extends LogicalGuard {
28     private final Logger logger = Logger.getLogger(getClass().
           getName());
29
30     public static final String typeName = "NOT";
31
32     @Override
33     public String getTypeName() {
34         return typeName;
35     }
36
37     @ElementUnion({
38         @Element(name = "True", type = TrueBooleanGuard.class,
               required = false),
39         @Element(name = "False", type = FalseBooleanGuard.class,
               required = false),
40         @Element(name = "EQ", type = EQRelationalGuard.class,
               required = false),
41         @Element(name = "GT", type = GTRelationalGuard.class,
               required = false),
42         @Element(name = "GTE", type = GTERelationalGuard.class,
               required = false),
43         @Element(name = "LT", type = LTRelationalGuard.class,
               required = false),
44         @Element(name = "LTE", type = LTERelationalGuard.class,
               required = false),
45         @Element(name = "NEQ", type = NEQRelationalGuard.class,
               required = false),
46         @Element(name = "AND", type = ANDCompareLogicalGuard.class,
               required = false),
47         @Element(name = "OR", type = ORCompareLogicalGuard.class,
               required = false),
48         @Element(name = "NOT", type = NOTLogicalGuard.class, required
               = false),
```

```
49          @Element(name = "RootGuard", type = RootGuard.class, required
                = false) })
50      Guard guard;
51
52      @Override
53      public boolean evaluate() throws Exception {
54          return !guard.evaluate();
55      }
56
57      @Override
58      public ArrayList<TreeItem> getChildren() {
59          ArrayList<TreeItem> rList = new ArrayList<TreeItem>();
60          if (guard != null) {
61              rList.add(guard);
62          }
63          return rList;
64      }
65
66      @Override
67      protected void doAddChild(TreeItem item) throws Exception {
68          if (isGuard(item)) {
69              if (guard == null) {
70                  guard = (Guard) item;
71                  return;
72              }
73              throw new Exception("NOTLogicalGuard already has child");
74          }
75
76      }
77
78      @Override
79      protected void doDeleteChild(TreeItem child) throws Exception {
80          if (guard == child) {
81              guard = null;
82          }
83
84      }
85
86      @Override
87      public Guard clone() {
88          NOTLogicalGuard clonedSelf = new NOTLogicalGuard();
89          if (guard != null) {
90              try {
91                  clonedSelf.addChild(guard.clone());
92              } catch (Exception e) {
93                  logger.severe(e.getMessage());
94              }
95          }
96
97          return clonedSelf;
98      }
99
100     @Override
101     public void updateExpression(ArrayList<Variable> variables) {
102         guard.updateExpression(variables);
```

```
103     }
104
105     @Override
106     public boolean isVariableInUse(
107             Variable variable) {
108         if (guard != null && guard.isVariableInUse(variable)) {
109             return true;
110         }
111         return false;
112     }
113
114     @Override
115     public void validate(List<ErrorValue> errorList, BasicTemplate
            basicTemplate) {
116         if (guard != null) {
117             guard.validate(errorList, basicTemplate);
118         } else {
119             errorList.add(new ErrorValue(
120                     "No child has been added to NOT-guard",
                            basicTemplate));
121         }
122     }
123
124     @Override
125     public void setupObserverObservableHierarchy() {
126         if(guard!=null)
127         {
128             guard.addObserver(this);
129             guard.setupObserverObservableHierarchy();
130         }
131
132     }
133
134 }
```

guard/relationalguards /EQRelationalGuard.java

```
1  package util.nominaltest2.model.guard.relationalguards;
2
3  import util.nominaltest2.model.expression.SingleExpression;
4  import util.nominaltest2.model.expression.singleExpressions.
       ArgumentSingleExpression;
5  import util.nominaltest2.model.guard.Guard;
6
7  public class EQRelationalGuard extends RelationalGuard {
8
9      public static final String typeName = "Equal";
10
11     @Override
```

```
12     public String getTypeName() {
13         return typeName;
14     }
15 
16     @Override
17     public boolean evaluate() throws Exception {
18         ArgumentSingleExpression[] boundedExprs =
               getBoundedExpressions();
19         return boundedExprs[0].getLongValue() == boundedExprs[1].
               getLongValue();
20     }
21 
22     @Override
23     public Guard clone() {
24         EQRelationalGuard clonedSelf = new EQRelationalGuard();
25         clonedSelf.setExpressions(new SingleExpression[] { expr1,
               expr2 });
26 
27         return clonedSelf;
28     }
29 
30 }
```

guard/relationalguards /GTERelationalGuard.java

```
1  package util.nominaltest2.model.guard.relationalguards;
2 
3  import util.nominaltest2.model.expression.SingleExpression;
4  import util.nominaltest2.model.expression.singleExpressions.
       ArgumentSingleExpression;
5  import util.nominaltest2.model.guard.Guard;
6 
7  public class GTERelationalGuard extends RelationalGuard {
8      public static final String typeName = "Greater Than or Equal";
9 
10     @Override
11     public String getTypeName() {
12         return typeName;
13     }
14 
15     @Override
16     public boolean evaluate() throws Exception {
17         ArgumentSingleExpression[] boundedExprs =
               getBoundedExpressions();
18         return boundedExprs[0].getLongValue() >= boundedExprs[1].
               getLongValue();
19     }
20 
21     @Override
```

```
22     public Guard clone() {
23         GTERelationalGuard clonedSelf = new GTERelationalGuard();
24         SingleExpression[] clonedExpr = cloneExpressions();
25         clonedSelf.setExpressions(clonedExpr);
26         return clonedSelf;
27     }
28 }
```

guard/relationalguards /GTRelationalGuard.java

```
1  package util.nominaltest2.model.guard.relationalguards;
2
3  import util.nominaltest2.model.expression.SingleExpression;
4  import util.nominaltest2.model.expression.singleExpressions.
       ArgumentSingleExpression;
5  import util.nominaltest2.model.guard.Guard;
6
7  public class GTRelationalGuard extends RelationalGuard {
8
9      public static final String typeName = "Greater Than";
10
11     @Override
12     public String getTypeName() {
13         return typeName;
14     }
15
16     @Override
17     public boolean evaluate() throws Exception {
18         ArgumentSingleExpression[] boundedExprs =
               getBoundedExpressions();
19         return boundedExprs[0].getLongValue() > boundedExprs[1].
               getLongValue();
20     }
21
22     @Override
23     public Guard clone() {
24         GTRelationalGuard clonedSelf = new GTRelationalGuard();
25         SingleExpression[] clonedExpr = cloneExpressions();
26         clonedSelf.setExpressions(clonedExpr);
27         return clonedSelf;
28     }
29
30 }
```

guard/relationalguards /LTERelationalGuard.java

```
1  package util.nominaltest2.model.guard.relationalguards;
2
3  import util.nominaltest2.model.expression.SingleExpression;
4  import util.nominaltest2.model.expression.singleExpressions.
       ArgumentSingleExpression;
5  import util.nominaltest2.model.guard.Guard;
6
7  public class LTERelationalGuard extends RelationalGuard {
8      public static final String typeName = "Less Than or Equal";
9
10     @Override
11     public String getTypeName() {
12         return typeName;
13     }
14
15     @Override
16     public boolean evaluate() throws Exception {
17         ArgumentSingleExpression[] boundedExprs =
               getBoundedExpressions();
18         return boundedExprs[0].getLongValue() <= boundedExprs[1].
               getLongValue();
19     }
20
21     @Override
22     public Guard clone() {
23         LTERelationalGuard clonedSelf = new LTERelationalGuard();
24         SingleExpression[] clonedExpr = cloneExpressions();
25         clonedSelf.setExpressions(clonedExpr);
26         return clonedSelf;
27     }
28 }
```

guard/relationalguards /LTRelationalGuard.java

```
1  package util.nominaltest2.model.guard.relationalguards;
2
3  import util.nominaltest2.model.expression.SingleExpression;
4  import util.nominaltest2.model.expression.singleExpressions.
       ArgumentSingleExpression;
5  import util.nominaltest2.model.guard.Guard;
6
7  public class LTRelationalGuard extends RelationalGuard {
8      public static final String typeName = "Less Than";
9
```

```
10      @Override
11      public String getTypeName() {
12          return typeName;
13      }
14
15      @Override
16      public boolean evaluate() throws Exception {
17          ArgumentSingleExpression[] boundedExprs =
                getBoundedExpressions();
18          return boundedExprs[0].getLongValue() < boundedExprs[1].
                getLongValue();
19      }
20
21      @Override
22      public Guard clone() {
23          LTRelationalGuard clonedSelf = new LTRelationalGuard();
24          SingleExpression[] clonedExpr = cloneExpressions();
25          clonedSelf.setExpressions(clonedExpr);
26          return clonedSelf;
27      }
28  }
```

guard/relationalguards /NEQRelationalGuard.java

```
1   package util.nominaltest2.model.guard.relationalguards;
2
3   import util.nominaltest2.model.expression.SingleExpression;
4   import util.nominaltest2.model.expression.singleExpressions.
        ArgumentSingleExpression;
5   import util.nominaltest2.model.guard.Guard;
6
7   public class NEQRelationalGuard extends RelationalGuard {
8       public static final String typeName = "Not Equal";
9
10      @Override
11      public String getTypeName() {
12          return typeName;
13      }
14
15      @Override
16      public boolean evaluate() throws Exception {
17          ArgumentSingleExpression[] boundedExprs =
                getBoundedExpressions();
18          return boundedExprs[0].getLongValue() != boundedExprs[1].
                getLongValue();
19      }
20
21      @Override
22      public Guard clone() {
```

```
23          NEQRelationalGuard clonedSelf = new NEQRelationalGuard();
24          SingleExpression[] clonedExpr = cloneExpressions();
25          clonedSelf.setExpressions(clonedExpr);
26          return clonedSelf;
27      }
28  }
```

guard/relationalguards /RelationalGuard.java

```
1  package util.nominaltest2.model.guard.relationalguards;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  import org.simpleframework.xml.Element;
7  import org.simpleframework.xml.ElementUnion;
8  import org.simpleframework.xml.Path;
9
10 import util.nominaltest2.model.TreeItem;
11 import util.nominaltest2.model.errors.ErrorValue;
12 import util.nominaltest2.model.expression.SingleExpression;
13 import util.nominaltest2.model.expression.singleExpressions.
       ArgumentSingleExpression;
14 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
15 import util.nominaltest2.model.expression.singleExpressions.
       argument.EnumArgumentSingleExpression;
16 import util.nominaltest2.model.expression.singleExpressions.
       argument.IntArgumentSingleExpression;
17 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.MinusArithmeticSingleExpression;
18 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.MultiplicationArithmeticSingleExpression;
19 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.PlusArithmeticSingleExpression;
20 import util.nominaltest2.model.guard.Guard;
21 import util.nominaltest2.model.template.BasicTemplate;
22
23 public abstract class RelationalGuard extends Guard {
24
25     @Path("Expression1")
26     @ElementUnion( {
27             @Element(name = "EnumArgumentExpression", type =
                    EnumArgumentSingleExpression.class, required = false),
28             @Element(name = "IntArgumentExpression", type =
                    IntArgumentSingleExpression.class, required = false),
29             @Element(name = "Variable", type = Variable.class,
                    required = false),
```

```
30          @Element(name = "MinusArithmeticSingleExpression", type =
                MinusArithmeticSingleExpression.class, required =
                false),
31          @Element(name = "MultiplicationArithmeticSingleExprssion",
                 type = MultiplicationArithmeticSingleExpression.class
                , required = false),
32          @Element(name = "PlusArithmeticSingleExpression", type =
                PlusArithmeticSingleExpression.class, required = false
                ) })
33      SingleExpression expr1;
34
35      @Path("Expression2")
36      @ElementUnion( {
37          @Element(name = "EnumArgumentExpression", type =
                EnumArgumentSingleExpression.class, required = false),
38          @Element(name = "IntArgumentExpression", type =
                IntArgumentSingleExpression.class, required = false),
39          @Element(name = "Variable", type = Variable.class,
                required = false),
40          @Element(name = "MinusArithmeticSingleExpression", type =
                MinusArithmeticSingleExpression.class, required =
                false),
41          @Element(name = "MultiplicationArithmeticSingleExprssion",
                 type = MultiplicationArithmeticSingleExpression.class
                , required = false),
42          @Element(name = "PlusArithmeticSingleExpression", type =
                PlusArithmeticSingleExpression.class, required = false
                ) })
43      SingleExpression expr2;
44
45      public SingleExpression[] getExpressions() {
46          return new SingleExpression[] { expr1, expr2 };
47      }
48
49      protected ArgumentSingleExpression[] getBoundedExpressions()
50              throws Exception {
51          ArgumentSingleExpression[] boundedExprs = new
                ArgumentSingleExpression[2];
52          boundedExprs[0] = getBoundedExpression(expr1);
53          boundedExprs[1] = getBoundedExpression(expr2);
54          return boundedExprs;
55      }
56
57      private ArgumentSingleExpression getBoundedExpression(
            SingleExpression e)
58              throws Exception {
59          if (e == null) {
60              throw new Exception("Can't validate when an expression is
                    null");
61          } else if (e.getValue() instanceof ArgumentSingleExpression)
                {
62              return (ArgumentSingleExpression) e.getValue();
63          } else {
64              throw new Exception(
65                      "Can't validate when an expression isn't bounded: "
```

```
66                      + e.getValue());
67          }
68      }
69
70      @Override
71      public ArrayList<TreeItem> getChildren() {
72          return new ArrayList<TreeItem>();
73      }
74
75      @Override
76      protected void doAddChild(TreeItem item) throws Exception {
77          noTreeItemsException();
78      }
79
80      @Override
81      protected void doDeleteChild(TreeItem child) throws Exception {
82          noTreeItemsException();
83      }
84
85      private void noTreeItemsException() throws Exception {
86          throw new Exception(
87                  "RelationalGuard doesn't have TreeItems as children");
88      }
89
90      public void setLeftExpression(SingleExpression expression) {
91          expr1 = expression;
92      }
93
94      public void setRightExpression(SingleExpression expression) {
95          expr2 = expression;
96      }
97
98      protected SingleExpression[] cloneExpressions() {
99          SingleExpression[] clonedExprs = new SingleExpression[2];
100
101         if (expr1 != null) {
102             clonedExprs[0] = (SingleExpression) expr1.clone();
103         }
104         if (expr2 != null) {
105             clonedExprs[1] = (SingleExpression) expr2.clone();
106         }
107         return clonedExprs;
108     }
109
110     public void setExpressions(SingleExpression[] exprs) {
111         if (exprs.length == 2) {
112             expr1 = exprs[0];
113             expr2 = exprs[1];
114         }
115     }
116
117     @Override
118     public void updateExpression(ArrayList<Variable> variables) {
119         if (expr1 != null) {
```

```
120             SingleExpression newExpr = updateExpression(expr1,
                    variables);
121             if (newExpr != null)
122                 expr1 = newExpr;
123         }
124         if (expr2 != null) {
125             SingleExpression newExpr = updateExpression(expr2,
                    variables);
126             if (newExpr != null)
127                 expr2 = newExpr;
128         }
129     }
130
131     /**
132      * Update the given {@link SingleExpression} according to the
                variables
133      * given, in case it was referencing a variable.
134      *
135      * @param expression
136      * @param variables
137      * @return
138      */
139     private SingleExpression updateExpression(SingleExpression
            expression,
140             ArrayList<Variable> variables) {
141         if (expression != null && expression instanceof Variable) {
142             for (SingleExpression expr : variables) {
143                 if (expr instanceof Variable) {
144                     Variable variable = (Variable) expr;
145                     if (((Variable) expression).getName().equals(
146                             variable.getName())) {
147                         return expr;
148                     }
149                 }
150             }
151         }
152         return null;
153     }
154
155     @Override
156     public boolean isVariableInUse(Variable variable) {
157         if ((expr1 != null && expr1 == variable)
158                 || (expr2 != null && expr2 == variable)) {
159             return true;
160         }
161         return false;
162     }
163
164     @Override
165     public void validate(List<ErrorValue> errorList, BasicTemplate
            basicTemplate) {
166         if (expr1 != null && expr2 != null) {
167
168         } else {
169             errorList
```

```
170                 .add(new ErrorValue(
171                         "One or both expressions haven't been set in
                            RelationalGuard",
172                         basicTemplate));
173         }
174
175     }
176
177     @Override
178     public void setupObserverObservableHierarchy() {
179
180     }
181
182 }
```

guard/Guard.java

```
1  package util.nominaltest2.model.guard;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  import util.nominaltest2.model.TreeItem;
7  import util.nominaltest2.model.errors.ErrorValue;
8  import util.nominaltest2.model.expression.singleExpressions.
       Variable;
9  import util.nominaltest2.model.results.TestResult.testResultEnum;
10 import util.nominaltest2.model.template.BasicTemplate;
11
12 public abstract class Guard extends TreeItem {
13
14     public abstract String getTypeName();
15
16     /**
17      * Evaluate the guard, given a boolean as result
18      *
19      * @return <b>True</b> in case the guard has no problems, and
            one can
20      *         proceed.
21      * @throws Exception
22      */
23     public abstract boolean evaluate() throws Exception;
24
25     @Override
26     public testResultEnum getEnumResult() {
27        return null;
28     }
29
30     /**
31      * Checks it the given {@link TreeItem} is a instance of a Guard
            .
```

```
32      *
33      * @param item
34      * @return <b>True</b> in case the given item is a guard.
35      * @throws Exception
36      */
37     protected boolean isGuard(TreeItem item) throws Exception {
38        if (item instanceof Guard) {
39           return true;
40        }
41        throw new Exception("Can't add an item that is not a Guard");
42     }
43
44     @Override
45     public String toString() {
46        return getTypeName();
47     }
48
49     @Override
50     public abstract Guard clone();
51
52     /**
53      * Check if the given variable is being used
54      *
55      * @param paramSingleExpression
56      * @return
57      */
58     public abstract boolean isVariableInUse(Variable variable);
59
60     /**
61      * Updates the variables used by the guard to the given
            variables according
62      * to the names, is needed when a cloning has taken place.
63      *
64      * @param variables
65      */
66     public abstract void updateExpression(ArrayList<Variable>
            variables);
67
68     /**
69      * Validates that everything is in order for the Guard.
70      *
71      * @param errorList
72      *             a list of all errors that has occurred in the
            system.
73      * @param basicTemplate
74      *             the {@link BasicTemplate} that contains the Guard.
75      */
76     public abstract void validate(List<ErrorValue> errorList,
77           BasicTemplate basicTemplate);
78
79     public abstract void setupObserverObservableHierarchy();
80
81 }
```

guard/RootGuard.java

```
1  package util.nominaltest2.model.guard;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.logging.Logger;
6
7  import org.simpleframework.xml.Element;
8  import org.simpleframework.xml.ElementUnion;
9
10 import util.nominaltest2.model.TreeItem;
11 import util.nominaltest2.model.errors.ErrorValue;
12 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
13 import util.nominaltest2.model.guard.booleanguards.
       FalseBooleanGuard;
14 import util.nominaltest2.model.guard.booleanguards.TrueBooleanGuard
       ;
15 import util.nominaltest2.model.guard.logicalguards.NOTLogicalGuard;
16 import util.nominaltest2.model.guard.logicalguards.
       comparelogicalguards.ANDCompareLogicalGuard;
17 import util.nominaltest2.model.guard.logicalguards.
       comparelogicalguards.ORCompareLogicalGuard;
18 import util.nominaltest2.model.guard.relationalguards.
       EQRelationalGuard;
19 import util.nominaltest2.model.guard.relationalguards.
       GTERelationalGuard;
20 import util.nominaltest2.model.guard.relationalguards.
       GTRelationalGuard;
21 import util.nominaltest2.model.guard.relationalguards.
       LTERelationalGuard;
22 import util.nominaltest2.model.guard.relationalguards.
       LTRelationalGuard;
23 import util.nominaltest2.model.guard.relationalguards.
       NEQRelationalGuard;
24 import util.nominaltest2.model.template.BasicTemplate;
25
26 public class RootGuard extends Guard {
27     private final Logger logger = Logger.getLogger(getClass().
          getName());
28
29     @ElementUnion({
30             @Element(name = "True", type = TrueBooleanGuard.class,
                   required = false),
31             @Element(name = "False", type = FalseBooleanGuard.class,
                   required = false),
32             @Element(name = "EQ", type = EQRelationalGuard.class,
                   required = false),
33             @Element(name = "GT", type = GTRelationalGuard.class,
                   required = false),
34             @Element(name = "GTE", type = GTERelationalGuard.class,
                   required = false),
```

```
35          @Element(name = "LT", type = LTRelationalGuard.class,
                required = false),
36          @Element(name = "LTE", type = LTERelationalGuard.class,
                required = false),
37          @Element(name = "NEQ", type = NEQRelationalGuard.class,
                required = false),
38          @Element(name = "AND", type = ANDCompareLogicalGuard.class
                , required = false),
39          @Element(name = "OR", type = ORCompareLogicalGuard.class,
                required = false),
40          @Element(name = "NOT", type = NOTLogicalGuard.class,
                required = false),
41          @Element(name = "RootGuard", type = RootGuard.class,
                required = false) })
42      Guard guard;
43      public static final String typeName = "ROOT";
44
45      public RootGuard() {
46      }
47
48      @Override
49      public boolean evaluate() throws Exception {
50          logger.info("called evaluate() on RootGuard");
51          if (guard != null) {
52              return guard.evaluate();
53          } else {
54              return true;
55          }
56      }
57
58      @Override
59      protected void doAddChild(TreeItem item) throws Exception {
60          if (guard == null && item instanceof Guard
61                  && !(item instanceof RootGuard)) {
62              guard = (Guard) item;
63          } else {
64              throw new Exception("Can't add an extra guard to a root
                    guard");
65          }
66      }
67
68      @Override
69      protected void doDeleteChild(TreeItem child) throws Exception {
70          if (child == guard) {
71              guard = null;
72          } else {
73              throw new Exception("Can't delete unknown child from a
                    root guard");
74          }
75      }
76
77      @Override
78      public ArrayList<TreeItem> getChildren() {
79          ArrayList<TreeItem> guards = new ArrayList<TreeItem>();
80          if (guard != null) {
```

```
81              guards.add(guard);
82          }
83          return guards;
84      }
85
86      @Override
87      public String getTypeName() {
88          return typeName;
89      }
90
91      @Override
92      public RootGuard clone() {
93          RootGuard clonedSelf = new RootGuard();
94          if (guard != null) {
95              try {
96                  clonedSelf.addChild(guard.clone());
97              } catch (Exception e) {
98                  logger.severe(e.getMessage());
99              }
100         }
101         return clonedSelf;
102     }
103
104     @Override
105     public void updateExpression(ArrayList<Variable> variables) {
106         if (guard != null) {
107             guard.updateExpression(variables);
108         }
109     }
110
111     @Override
112     public boolean isVariableInUse(
113             Variable variable) {
114         if (guard != null && guard.isVariableInUse(variable)) {
115             return true;
116         }
117         return false;
118     }
119
120     @Override
121     public void validate(List<ErrorValue> errorList, BasicTemplate
            basicTemplate) {
122         if (guard != null)
123             guard.validate(errorList, basicTemplate);
124     }
125
126     public void setupObserverObservableHierarchy() {
127         if(guard!=null)
128         {
129             guard.addObserver(this);
130             guard.setupObserverObservableHierarchy();
131         }
132
133
134     }
```

```
135
136 }
```

results/PacketResult.java

```
1  package util.nominaltest2.model.results;
2
3  import java.util.ArrayList;
4
5  import org.simpleframework.xml.Attribute;
6  import org.simpleframework.xml.ElementList;
7
8  import util.nominaltest2.model.TestManager;
9  import util.nominaltest2.model.results.TestResult.testResultEnum;
10 import util.nominaltest2.model.validators.PacketValidator;
11
12 import common.dtusat2.packet.Packet;
13
14 public class PacketResult {
15
16    PacketValidator packetValidator;
17
18    @ElementList
19    ArrayList<ValidatorResult> validatorResults = new ArrayList<
          ValidatorResult>();
20
21    @Attribute(required = false)
22    testResultEnum packetResult;
23    Packet receivedPacket;
24
25    @Attribute(required =false)
26    String expectedPacketName;
27
28    @Attribute(required = false)
29    String receivedPacketName;
30
31
32    public void updateForSaving() {
33       if (packetValidator != null && packetValidator.getTypeName()
             != null) {
34          expectedPacketName = packetValidator.getTypeName();
35       }
36
37       if (receivedPacket != null) {
38          receivedPacketName = TestManager.getPacketTypeName(
                receivedPacket);
39       }
40
41       for (ValidatorResult vr : validatorResults) {
42          vr.updateForSaving();
43       }
```

```
44      }
45
46      public PacketResult(PacketValidator packetValidator) {
47          this.packetValidator = packetValidator;
48      }
49
50      public PacketValidator getPacketValidator() {
51          return packetValidator;
52      }
53
54      public void addValidatorResult(ValidatorResult result) {
55          validatorResults.add(result);
56      }
57
58      public ArrayList<ValidatorResult> getValidatorResults() {
59          return validatorResults;
60      }
61
62      /**
63       * Updates the result according to the {@link ValidatorResult}s
              currently
64       * stores on the given {@link PacketResult}. In case one result
              is
65       * <b>Fail</b>, we set the result to <b>Fail</b> as well and
              stop looking for
66       * more results.
67       */
68      public void updatePacketResult() {
69          if (packetResult == null) {
70              boolean tempRes = true;
71              for (ValidatorResult vres : validatorResults) {
72                  tempRes = vres.getResult();
73                  if (!tempRes) {
74                      packetResult = testResultEnum.FAIL;
75                      break;
76                  }
77              }
78              if (tempRes) {
79                  packetResult = testResultEnum.PASS;
80              }
81          }
82      }
83
84      /**
85       * Sets that there are no SimpleValidators on this packetresult.
              Thus =>
86       * true
87       */
88      public void noValidators() {
89          packetResult = testResultEnum.PASS;
90      }
91
92      public void setGuardFai() {
93          packetResult = testResultEnum.GUARD;
94      }
```

```
95
96      public testResultEnum getResult() {
97          return packetResult;
98      }
99
100     public void setReceivedPacket(Packet receivedPacket) {
101         this.receivedPacket = receivedPacket;
102     }
103
104     public Packet getReceivedPacket() {
105         return receivedPacket;
106     }
107
108     public void setWrongPacket() {
109         packetResult = testResultEnum.FAIL;
110     }
111
112 }
```

results/TestResult.java

```
1  package util.nominaltest2.model.results;
2
3  import java.util.ArrayList;
4
5  import org.simpleframework.xml.Attribute;
6  import org.simpleframework.xml.ElementList;
7
8  import util.nominaltest2.model.template.BasicTemplate;
9  import util.nominaltest2.model.validators.ExpressionValidator;
10 import util.nominaltest2.model.validators.PacketValidator;
11
12 public class TestResult implements Comparable<TestResult> {
13
14     private BasicTemplate basicTemplate;
15
16     @Attribute(required = false)
17     private String basicTemplateName;
18
19     @Attribute(required = false)
20     private testResultEnum res;
21
22     @ElementList
23     private ArrayList<PacketResult> packetResult = new ArrayList<
           PacketResult>();
24
25     public void updateForSaving() {
26         if (basicTemplate != null) {
27             basicTemplateName = basicTemplate.getNameOrIDString();
28         }
29
```

```
30          for (PacketResult pr : packetResult) {
31              pr.updateForSaving();
32          }
33      }
34
35      public TestResult(BasicTemplate basicTemplate, testResultEnum
            res) {
36          this.basicTemplate = basicTemplate;
37          this.res = res;
38      }
39
40      /**
41       * Get the packet result for the given packet validator. Return
             null in case
42       * there is no packet result.
43       *
44       * @param packetValidator
45       * @return
46       */
47      public PacketResult getPacketResult(PacketValidator
            packetValidator) {
48          for (PacketResult pr : packetResult) {
49              if (pr.getPacketValidator() == packetValidator) {
50                  return pr;
51              }
52          }
53          return null;
54      }
55
56      /**
57       * Return the result for the given {@link ExpressionValidator}.
58       *
59       * @param validator
60       *            the {@link ExpressionValidator} whose result one
             want.
61       * @return <b>null</b> in case the given {@link
             ExpressionValidator} is not
62       *         present on this {@link TestResult}, otherwise the
63       *         {@link ValidatorResult}
64       */
65      public ValidatorResult getValidatorResult(ExpressionValidator
            validator) {
66          for (PacketResult pr : packetResult) {
67              for (ValidatorResult vr : pr.getValidatorResults()) {
68                  if (vr.getValidator() == validator) {
69                      return vr;
70                  }
71              }
72          }
73          return null;
74      }
75
76      public void addPacketResult(PacketResult pResult) {
77          packetResult.add(pResult);
78      }
```

```
79
80      public void addPacketResults(ArrayList<PacketResult> pResults) {
81          packetResult.addAll(pResults);
82      }
83
84      public ArrayList<PacketResult> getPacketResults() {
85          return packetResult;
86      }
87
88      /**
89       * The backbone for all results generated in the system. The
             current result one can get are:
90       * <br />
91       * <center>Pass > None > Fail > Guard > Inconc</center>
92       * <br />
93       * In which <b>Pass</b> is the best result while <b>Inconc</b> (
             inconclusive) is the worst.
94       */
95      public enum testResultEnum {
96          PASS, NONE, FAIL, GUARD, INCONC
97      }
98
99      /**
100      * Set the result to be <b>Inconc</b>, as an indication of a
             timeout.
101      */
102     public synchronized void setTimedOut() {
103         res = testResultEnum.INCONC;
104     }
105
106     public synchronized void setResult(boolean pass) {
107         if (pass) {
108             res = testResultEnum.PASS;
109         } else {
110             res = testResultEnum.FAIL;
111         }
112     }
113
114     public synchronized testResultEnum getEnumResult() {
115         return res;
116     }
117
118     public BasicTemplate getTestCase() {
119         return basicTemplate;
120     }
121
122     @Override
123     public String toString() {
124         return res.toString();
125     }
126
127     @Override
128     public int compareTo(TestResult other) {
129         return res.ordinal() - other.getEnumResult().ordinal();
130     }
```

```
131
132 }
```

results/ValidatorResult.java

```
1  package util.nominaltest2.model.results;
2
3  import org.simpleframework.xml.Attribute;
4  import org.simpleframework.xml.Element;
5  import org.simpleframework.xml.ElementUnion;
6
7  import common.types.argument.Argument;
8  import common.types.argument.EnumArgument;
9  import common.types.argument.IntArgument;
10
11 import util.nominaltest2.model.expression.Expression;
12 import util.nominaltest2.model.expression.SingleExpression;
13 import util.nominaltest2.model.expression.singleExpressions.
       ArgumentSingleExpression;
14 import util.nominaltest2.model.template.DTUSatParameterArgument;
15 import util.nominaltest2.model.test.ApplyTest;
16 import util.nominaltest2.model.test.ConcurrentCollectionTest;
17 import util.nominaltest2.model.test.DelayTest;
18 import util.nominaltest2.model.test.SequentialCollectionTest;
19 import util.nominaltest2.model.validators.ExpressionValidator;
20
21 public class ValidatorResult {
22
23     private ExpressionValidator validator;
24
25     @Attribute(name = "ExpectedArgument", required = false)
26     private String expectedArgument;
27
28     @Attribute(required = false)
29     private boolean result;
30
31     private DTUSatParameterArgument dtuParamArg;
32
33     @ElementUnion( {
34             @Element(name = "Received_Integer", type = IntArgument.
                    class, required = false),
35             @Element(name = "Received_Enum", type = EnumArgument.class
                    , required = false) })
36     private Argument receivedArgument;
37
38     @Element(name = "Expected", required = false)
39     private String expectedExpression;
40
41     public void updateForSaving() {
42         if (validator != null && validator.getExpression() != null) {
```

```
43            expectedExpression = validator.getExpression().
                  getStringValue();
44            expectedArgument = validator.getParamName();
45        }
46
47        if (dtuParamArg != null && dtuParamArg.getArgument() != null)
              {
48            receivedArgument = dtuParamArg.getArgument();
49        }
50    }
51
52    public ValidatorResult(ExpressionValidator validator,
53            DTUSatParameterArgument dtuParamArg, boolean result) {
54        this.validator = validator;
55        this.result = result;
56        this.dtuParamArg = dtuParamArg;
57    }
58
59    public DTUSatParameterArgument getParamArg() {
60        return dtuParamArg;
61    }
62
63    public boolean getResult() {
64        return result;
65    }
66
67    public ExpressionValidator getValidator() {
68        return validator;
69    }
70 }
```

sutAdapter/dummy /FlashModuleSimulator.java

```
1  package util.nominaltest2.model.sutAdapter.dummy;
2
3  import java.util.logging.Logger;
4
5  import common.dtusat2.packet.ConfirmationPacket;
6  import common.types.TypeMaps;
7  import common.types.argument.Argument;
8  import common.types.argument.EnumArgument;
9  import common.types.argument.IntArgument;
10 import common.types.command.CommandData;
11 import common.types.confirmation.ConfirmationType;
12
13 public class FlashModuleSimulator {
14     private final Logger logger = Logger.getLogger(getClass().
           getName());
15
```

```
16      private long totalBlocks = 32;
17      private long erasedBlocks = 0;
18      private long curSeqNo;
19
20      public synchronized ConfirmationPacket generateReplyTo(
            CommandData data,
21              long curSeqNo) {
22          this.curSeqNo = curSeqNo;
23          String recvCmdName = data.getCommandType().getName();
24          logger.info("FlashSim received command: " + recvCmdName);
25
26          if (recvCmdName.equals("CMD_FLA_ERASE_BLOCKS")) {
27              logger.info("FlashSimulator received CMD_FLA_ERASE_BLOCKS.
                    totalBlocks="
28                      + totalBlocks + ", currentlyErasedBlocks=" +
                            erasedBlocks);
29              Argument[] args = data.getArguments();
30              if (args.length == 1) {
31                  if (args[0].isInt()) {
32                      IntArgument arg = (IntArgument) args[0];
33                      long toBeErased = arg.getValue();
34
35                      if (toBeErased < 1 || toBeErased > 62) {
36                          return generateEnumReply("CONF_FLA_FLASH_ERROR",
37                                  "FLASHFS2_UNEXPECTED_BLOCK_SIZE_OR_COUNT");
38                      }
39
40                      long actuallyErasedBlocks;
41                      if (erasedBlocks + toBeErased <= totalBlocks) {
42                          erasedBlocks += toBeErased;
43                          actuallyErasedBlocks = toBeErased;
44                      } else {
45                          toBeErased=totalBlocks-erasedBlocks;
46                          erasedBlocks=totalBlocks;
47                          actuallyErasedBlocks=toBeErased;
48                      }
49                      logger.info("FlashSimulator received done
                            CMD_FLA_ERASE_BLOCKS. totalBlocks="
50                              + totalBlocks
51                              + ", currentlyErasedBlocks="
52                              + erasedBlocks);
53
54                      return generateIntReply("CONF_FLA_ERASE_OK",
55                              actuallyErasedBlocks);
56                  }
57              }
58          }
59          logger.info("FlashSimulator failed erasing.");
60          return generateIntReply("CONF_FAILED", 1);
61      }
62
63      /**
64       * Generates a reply-packet containing an Enum-argument
65       *
66       * @param type
```

```
67      * @param enumArg
68      * @return
69      */
70     private ConfirmationPacket generateEnumReply(String type, String
           enumArg) {
71         ConfirmationType confType = TypeMaps.getConfirmationTypeMap()
               .getType(
72                 type);
73         Argument[] results = new Argument[1];
74         results[0] = new EnumArgument(enumArg);
75         return new ConfirmationPacket(curSeqNo, confType, results);
76     }
77
78     /**
79      * Generates a reply-packet containing an Integer-argument
80      *
81      * @param type
82      * @param enumArg
83      * @return
84      */
85     private ConfirmationPacket generateIntReply(String type, long
           intArg) {
86         ConfirmationType confType = TypeMaps.getConfirmationTypeMap()
               .getType(
87                 type);
88         logger.info("In generateIntReply for type: " + type
89                 + ", fetched confType: " + confType.getName());
90         Argument[] results = new Argument[1];
91         results[0] = new IntArgument(intArg);
92         return new ConfirmationPacket(curSeqNo, confType, results);
93     }
94 }
```

sutAdapter/dummy /ReceiveMonitorDummy.java

```
1  package util.nominaltest2.model.sutAdapter.dummy;
2
3  import java.util.ArrayList;
4  import java.util.logging.Logger;
5
6  import util.nominaltest2.model.sutAdapter.AbstractReceiveMonitor;
7  import common.dtusat2.packet.ConfirmationPacket;
8  import common.dtusat2.packet.Packet;
9
10 public class ReceiveMonitorDummy extends AbstractReceiveMonitor {
11     private final Logger logger = Logger.getLogger(getClass().
           getName());
12
13     @Override
```

```
14      public synchronized Packet receivePacket(long seqNo) throws
            InterruptedException {
15          Packet receivedPacket;
16          logger.info("Have a Thread waiting for " + seqNo);
17          while (!queue.containsKey(seqNo) || isQueueEmpty(seqNo)) {
18                logger.info("Queue did not contain " + seqNo);
19                this.wait();
20
21          }
22          ArrayList<Packet> packets =  queue.get(seqNo);
23          receivedPacket = packets.get(0);
24          queue.get(seqNo).remove(0);
25          if (packets.size() == 0) {
26             queue.remove(seqNo);
27          }
28          return receivedPacket;
29      }
30
31      @Override
32      public synchronized void gotMessage(Packet packet) {
33          logger.info("Added " + packet.getSeqNo() + " to the queue");
34          if (packet instanceof ConfirmationPacket) {
35             updateQueue(((ConfirmationPacket) packet).getCmdSeqNo(),
                   packet);
36          } else {
37             updateQueue(packet.getSeqNo(), packet);
38          }
39          if (queue.size() > 0) {
40             logger.info("NotifyAll was executed");
41             this.notifyAll();
42          }
43      }
44
45      private synchronized void updateQueue(long seqNo,Packet packet)
            {
46          if (queue.containsKey(seqNo)) {
47             queue.get(seqNo).add(packet);
48          } else {
49             ArrayList<Packet> packets = new ArrayList<Packet>();
50             packets.add(packet);
51             queue.put(seqNo, packets);
52          }
53
54      }
55
56  }
```

sutAdapter/dummy /ReplierThreadDummy.java

```
1  package util.nominaltest2.model.sutAdapter.dummy;
2
3  import java.util.Random;
4  import java.util.logging.Logger;
5
6  import common.dtusat2.packet.ConfirmationPacket;
7  import common.dtusat2.packet.Packet.PacketPrio;
8  import common.types.argument.Argument;
9  import common.types.command.CommandData;
10 import common.types.confirmation.ConfirmationType;
11
12 public class ReplierThreadDummy extends Thread {
13
14    private final Logger logger = Logger.getLogger(getClass().
          getName());
15
16    long curSeqNo;
17    ConfirmationType confType;
18    Argument[] results;
19    ReceiveMonitorDummy receiveMonitor;
20    CommandData data;
21    long execTime;
22    PacketPrio priority;
23
24    public ReplierThreadDummy(ReceiveMonitorDummy receiveMonitor,
          long seqNo,
25           CommandData data, long execTime, PacketPrio priority) {
26       this.receiveMonitor = receiveMonitor;
27       this.curSeqNo = seqNo;
28       this.data = data;
29       this.execTime = execTime;
30       this.priority = priority;
31    }
32
33    @Override
34    public void run() {
35       try {
36          Random r = new Random();
37          Thread.sleep(1000 + r.nextInt(2000));
38
39          String recvModName = data.getDestination().name();
40          ConfirmationPacket conf = SUTAdapterDummy.
                getFlashSimulator()
41                .generateReplyTo(data, curSeqNo);
42          receiveMonitor.gotMessage(conf);
43       } catch (Exception e) {
44          logger.severe(e.getMessage());
45       }
46    }
47 }
```

sutAdapter/dummy /SUTAdapterDummy.java

```
1  package util.nominaltest2.model.sutAdapter.dummy;
2
3  import java.util.ArrayList;
4  import java.util.logging.Logger;
5
6  import util.nominaltest2.model.sutAdapter.AbstractSUTAdapter;
7
8  import common.dtusat2.ModuleID;
9  import common.dtusat2.packet.Packet;
10 import common.dtusat2.packet.Packet.PacketPrio;
11 import common.types.argument.Argument;
12 import common.types.command.CommandData;
13 import common.types.command.CommandType;
14 import common.types.exceptions.IllegalParameterException;
15 import common.types.exceptions.UnknownCommandTypeException;
16
17 public class SUTAdapterDummy extends AbstractSUTAdapter {
18    private final Logger logger = Logger.getLogger(getClass().
          getName());
19
20    long seqNoCounter = 10;
21    private static FlashModuleSimulator flashSim = new
          FlashModuleSimulator();
22
23    public SUTAdapterDummy() {
24       receiveMonitor = new ReceiveMonitorDummy();
25    }
26
27    public static FlashModuleSimulator getFlashSimulator() {
28       return flashSim;
29    }
30
31    @Override
32    public void createConnection() {
33       this.receiveMonitor = new ReceiveMonitorDummy();
34
35    }
36
37    @Override
38    public void closeConnection() {
39
40    }
41
42    @Override
43    public Packet receivePacket(long seqNo) throws
          InterruptedException {
44       return receiveMonitor.receivePacket(seqNo);
45    }
46
47    @Override
```

```
48      public int getPort() {
49          return 0;
50      }
51
52      @Override
53      public String getHost() {
54          return "SUT Adapter is not connecting to an external source";
55      }
56
57
58      public long sendCommand(CommandData data, long execTime,
            PacketPrio priority) {
59          logger.info("sendMessage in SUTAdapterDummy. data=" + data);
60          ReplierThreadDummy replier = new ReplierThreadDummy(
61                  (ReceiveMonitorDummy) receiveMonitor, ++seqNoCounter,
                        data,
62                  execTime, priority);
63          replier.start();
64          return seqNoCounter;
65      }
66
67      @Override
68      public long sendCommand(CommandType cmdType, ModuleID module,
69              ArrayList<Argument> arguments, long execTime, PacketPrio
                    priority) {
70          Argument[] args = arguments.toArray(new Argument[0]);
71          return sendCommand(new CommandData(cmdType, module, args),
                execTime, priority);
72      }
73
74      @Override
75      public void changeConnection(String host, int port) {
76
77      }
78
79      @Override
80      public long sendCommand(CommandType type, ModuleID moduleID,
81              Argument[] arguments, long execTime, PacketPrio priority)
                    {
82          CommandData data = new CommandData(type, moduleID, arguments)
                ;
83          return sendCommand(data, execTime, priority);
84      }
85  }
```

sutAdapter/AbstractReceiveMonitor.java

```
1  package util.nominaltest2.model.sutAdapter;
2
3  import java.util.ArrayList;
4  import java.util.HashMap;
```

```
5
6  import antlr.collections.List;
7
8  import common.dtusat2.packet.Packet;
9
10 public abstract class AbstractReceiveMonitor {
11
12     protected HashMap<Long, ArrayList<Packet>> queue = new HashMap<
           Long, ArrayList<Packet>>();
13
14     public abstract Packet receivePacket(long seq) throws
           InterruptedException;
15
16     public abstract void gotMessage(Packet packet);
17
18     /**
19      * Returns whether there are any {@link Packet}s for the given
            sequence
20      * number.
21      *
22      * @param seqNo
23      * @return
24      */
25     protected boolean isQueueEmpty(long seqNo) {
26        if (queue.containsKey(seqNo)) {
27           if (queue.get(seqNo).size() == 0) {
28              return true;
29           } else {
30              return false;
31           }
32        } else {
33           return true;
34        }
35     }
36 }
```

sutAdapter/AbstractSUTAdapter.java

```
1  package util.nominaltest2.model.sutAdapter;
2
3  public abstract class AbstractSUTAdapter implements SAI {
4
5     protected AbstractReceiveMonitor receiveMonitor;
6
7  }
```

sutAdapter/Connection.java

```
1  package util.nominaltest2.model.sutAdapter;
2
3  import java.io.IOException;
4  import java.io.ObjectInputStream;
5  import java.io.ObjectOutputStream;
6  import java.net.ConnectException;
7  import java.net.Socket;
8  import java.util.logging.Logger;
9
10 import common.types.TypeMaps;
11
12 public class Connection {
13     private final Logger logger = Logger.getLogger(getClass().
           getName());
14
15     // Standard connection info, change if needed.
16     protected int port = 5002;
17     protected String host = "tif.tif.dtusat.dtu.dk";
18
19     private Socket con;
20
21     private ObjectOutputStream out;
22     private ObjectInputStream in;
23
24     /**
25      * Initialise the connection with the Ground Station using a TCP
            connection
26      */
27     public void openConnection() {
28        try {
29           TypeMaps.loadTypes();
30           Socket con = null;
31
32           while (con == null)
33              try {
34                 con = new Socket(host, port);
35                 logger.info("Connected to " + host + ":" + port);
36              } catch (ConnectException e) {
37                 logger.info("Protocol not ready");
38                 Thread.sleep(15 * 1000);
39              }
40
41           // con.setTcpNoDelay(true);
42
43           // Order important to prevent deadlock!
44           out = new ObjectOutputStream(con.getOutputStream());
45           in = new ObjectInputStream(con.getInputStream());
46
47        } catch (Exception e) {
48           logger.severe(e.getMessage());
49        }
50     }
```

```
51
52      public void closeConnection() {
53          try {
54              out.close();
55              in.close();
56              con.close();
57          } catch (IOException e) {
58              logger.severe(e.getMessage());
59          }
60      }
61
62      public ObjectOutputStream getOutputStream() {
63          return out;
64      }
65
66      public ObjectInputStream getInputStream() {
67          return in;
68      }
69
70      public String getHost() {
71          return host;
72      }
73
74      public int getPort() {
75          return port;
76      }
77
78      /**
79       * Sets the new host and port to be used. It will not generate a
             new
80       * connection, which can be done by callin createConnection()
81       *
82       * @param host
83       * @param port
84       */
85      public void updateConnection(String host, int port) {
86          this.host = host;
87          this.port = port;
88      }
89
90  }
```

sutAdapter/ReceiveMonitor.java

```
1  package util.nominaltest2.model.sutAdapter;
2
3  import java.util.ArrayList;
4  import java.util.logging.Logger;
5
6  import common.dtusat2.packet.ConfirmationPacket;
7  import common.dtusat2.packet.Packet;
```

```
8  import common.dtusat2.packet.SignalPacket;
9  import common.dtusat2.packet.TeledataPacket;
10
11 /**
12  * A monitor used for managing the receival of packages, making
        sure only one
13  * Thread is allowed at once.
14  *
15  * @author andreas
16  */
17 public class ReceiveMonitor extends AbstractReceiveMonitor {
18     private final Logger logger = Logger.getLogger(getClass().
          getName());
19
20     /**
21      * Will return any packages arriving with the given sequence
            number.
22      *
23      * @param seqNo
24      *             the sequence number of the send message.
25      */
26     @Override
27     public synchronized Packet receivePacket(long seqNo)
28             throws InterruptedException {
29         Packet receivedPacket;
30         logger.info("Have a Thread waiting for " + seqNo);
31         while (!queue.containsKey(seqNo) || isQueueEmpty(seqNo)) {
32             logger.info("Queue did not contain " + seqNo);
33             this.wait();
34         }
35         ArrayList<Packet> packets = queue.get(seqNo);
36         receivedPacket = packets.get(0);
37         queue.get(seqNo).remove(0);
38         if (packets.size() == 0) {
39             queue.remove(seqNo);
40         }
41         return receivedPacket;
42     }
43
44     /**
45      * Will insert a Packet into the pool.
46      *
47      * @param packet
48      */
49     @Override
50     public synchronized void gotMessage(Packet packet) {
51         logger.info("Added " + packet.getSeqNo() + " to the queue");
52         if (packet instanceof SignalPacket) {
53             logger.info("Received signal packet. Doing nothing");
54         } else {
55             long seqNo = -1;
56             if (packet instanceof TeledataPacket) {
57                 seqNo = ((TeledataPacket) packet).getCmdSeqNo();
58             } else if (packet instanceof ConfirmationPacket) {
59                 seqNo = ((ConfirmationPacket) packet).getCmdSeqNo();
```

```
60          }
61
62          logger.info("Received reply for seq. no " + seqNo);
63
64          if (queue.containsKey(seqNo)) {
65              queue.get(seqNo).add(packet);
66          } else {
67              ArrayList<Packet> packets = new ArrayList<Packet>();
68              packets.add(packet);
69              queue.put(seqNo, packets);
70          }
71          if (queue.size() > 0) {
72              logger.info("NotifyAll was executed");
73              this.notifyAll();
74          }
75      }
76  }
77 }
```

sutAdapter/ReceiveThread.java

```
1  package util.nominaltest2.model.sutAdapter;
2
3  import java.io.ObjectInputStream;
4  import java.util.logging.Logger;
5
6  import common.dtusat2.packet.Packet;
7  import common.dtusat2.packet.Packet.PacketType;
8
9  public class ReceiveThread extends Thread {
10     private final Logger logger = Logger.getLogger(getClass().
           getName());
11
12     ReceiveMonitor receiveMonitor;
13     ObjectInputStream in;
14
15     public ReceiveThread(ReceiveMonitor receiveMonitor,
           ObjectInputStream in) {
16         this.receiveMonitor = receiveMonitor;
17         this.in = in;
18     }
19
20     public void setInoutStream(ObjectInputStream in) {
21         this.in = in;
22     }
23
24     @Override
25     public void run() {
26         logger.info("receiveThread run");
27         while (true) {
28             try {
```

```
29              logger.info("trying to read input");
30              byte[] binpacket = (byte[]) in.readObject();
31              int len = binpacket.length;
32              // Decode packet
33              Packet packet = Packet.create(binpacket);
34
35              long seqNo = Packet.findSeqNo(packet.toBytes());
36              logger.info("Tester recvd packet with number " + seqNo
37                    + " and seq on packet " + packet.getSeqNo());
38
39              receiveMonitor.gotMessage(packet);
40            } catch (Exception e) {
41              logger.severe(e.getMessage());
42              break;
43            }
44        }
45    }
46
47 }
```

sutAdapter/SAI.java

```
1  package util.nominaltest2.model.sutAdapter;
2
3  import java.util.ArrayList;
4
5  import util.protocolrunner.ProtocolRunner;
6
7  import common.dtusat2.ModuleID;
8  import common.dtusat2.packet.Packet;
9  import common.dtusat2.packet.Packet.PacketPrio;
10 import common.types.argument.Argument;
11 import common.types.command.CommandType;
12 import common.types.parameter.Parameter;
13
14 public interface SAI {
15
16    /**
17     * Establish a connection to the {@link ProtocolRunner}, located
          on the Host
18     * and Port set on the SUT Adapter.
19     */
20    public void createConnection();
21
22    /**
23     * Closes the current connection, and cleans up after it.
24     */
25    public void closeConnection();
26
27    // public long sendMessage(String cmdData, long execTime,
          PacketPrio
```

```
28      // priority)
29      /**
30       * Will establish a new connection with the given Host and Port
31       *
32       * @param host
33       *            the string that connection should connect to.
34       * @param port
35       *            the port that should be used for the connection.
36       */
37      public void changeConnection(String host, int port);
38
39      public int getPort();
40
41      public String getHost();
42
43      /**
44       * Receive the packet that has arrived, as a response to the
           command with
45       * the given sequence number.
46       *
47       * @param seqNo
48       *            the sequence number for the command which want to
           retrieve a
49       *            response.
50       * @return the packet which arrived first.
51       * @throws InterruptedException
52       */
53      public Packet receivePacket(long seqNo) throws
            InterruptedException;
54
55      /**
56       * Sends a command to the satellite with the given arguments.
57       *
58       * @param cmdType
59       * @param module
60       *            the module which should handle the command.
61       * @param arguments
62       *            the arguments that the {@link Parameter}s on the
           command
63       *            should use. They should be lay out in a 1:1
           fashion
64       *            according to the {@link Parameter}s on the command
           .
65       * @param execTime
66       * @param priority
67       * @return
68       */
69      public long sendCommand(CommandType cmdType, ModuleID module,
70            ArrayList<Argument> arguments, long execTime, PacketPrio
                priority);
71
72      /**
73       * Sends a command to the satellite with the given arguments.
74       *
75       * @param cmdType
```

```
76      * @param module
77      *              the module which should handle the command.
78      * @param arguments
79      *              the arguments that the {@link Parameter}s on the
            command
80      *              should use. They should be lay out in a 1:1
            fashion
81      *              according to the {@link Parameter}s on the command
            .
82      * @param execTime
83      * @param priority
84      * @return
85      */
86     public long sendCommand(CommandType type, ModuleID moduleID,
87             Argument[] arguments, long execTime, PacketPrio priority);
88
89 }
```

sutAdapter/SendMonitor.java

```
1  package util.nominaltest2.model.sutAdapter;
2
3  import java.io.IOException;
4  import java.io.ObjectOutputStream;
5  import java.util.logging.Logger;
6
7  import common.dtusat2.ModuleID;
8  import common.dtusat2.packet.CommandPacket;
9  import common.dtusat2.packet.Packet;
10 import common.dtusat2.packet.SignalPacket;
11 import common.dtusat2.packet.Packet.PacketPrio;
12 import common.types.argument.Argument;
13 import common.types.command.CommandData;
14 import common.types.command.CommandType;
15 import common.types.exceptions.IllegalParameterException;
16 import common.types.exceptions.UnknownCommandTypeException;
17
18 /**
19  * Will handle the sending of {@link CommandType}, making sure they
        get unique
20  * sequence number, hence the monitor name.
21  *
22  * @author andreas
23  */
24 public class SendMonitor {
25     private final Logger logger = Logger.getLogger(getClass().
          getName());
26
27     ObjectOutputStream out = null;
28     long seqNoCounter = 0;
29
```

```
30      private synchronized long nextSeqNo() {
31          return seqNoCounter++;
32      }
33
34      public SendMonitor(ObjectOutputStream out) {
35          this.out = out;
36      }
37
38      public void setOutputStream(ObjectOutputStream out) {
39          this.out = out;
40      }
41
42      private void sendPacket(Packet pkt) throws IOException {
43
44          if (out != null) {
45              // Be sure to send one packet at a time
46              synchronized (out) {
47                  out.writeObject(pkt.toBytes());
48                  out.flush();
49              }
50
51          } else {
52              logger.severe("No connection");
53          }
54      }
55
56      /**
57       * Sends the given command.
58       *
59       * @param data
60       * @param execTime
61       * @param priority
62       * @return
63       */
64      public long sendMessage(CommandData data, long execTime,
            PacketPrio priority) {
65          long seqNo = nextSeqNo();
66          try {
67              CommandPacket cmd = new CommandPacket(execTime, seqNo,
                    priority,
68                      data);
69
70              sendPacket(cmd);
71          } catch (IllegalArgumentException e) {
72              logger.severe(e.getMessage());
73          } catch (IOException e) {
74              logger.severe(e.getMessage());
75          }
76          return seqNo;
77      }
78
79      public long sendSignal(String name) {
80          long seqNo = nextSeqNo();
81          try {
82              SignalPacket sig = new SignalPacket(name);
```

```
83            sig.setDestination(ModuleID.COM);
84            sig.setPktSeqNo(seqNo);
85
86            sendPacket(sig);
87        } catch (Exception e) {
88            logger.severe(e.getMessage());
89        }
90        return seqNo;
91    }
92 }
```

sutAdapter/SUTAdapter.java

```
1  package util.nominaltest2.model.sutAdapter;
2
3  import java.util.ArrayList;
4
5  import common.dtusat2.ModuleID;
6  import common.dtusat2.packet.Packet;
7  import common.dtusat2.packet.Packet.PacketPrio;
8  import common.types.argument.Argument;
9  import common.types.command.CommandData;
10 import common.types.command.CommandType;
11
12 /**
13  * Will be responsible for managing the communication between the
        System Under
14  * Test (SUT) and the Test Enviroment
15  *
16  * @author andreas
17  *
18  */
19 public class SUTAdapter extends AbstractSUTAdapter {
20
21     private Connection con;
22
23     // Monitors that guarantee only one thread send or receives at a
           time.
24     private ReceiveMonitor receiveMonitor;
25     private SendMonitor sendMonitor;
26
27     private ReceiveThread receiveThread;
28
29     public SUTAdapter() {
30         con = new Connection();
31     }
32
33     /**
34      * Sets up the SUT Adapter, so it can send and receive messages
           from the
35      * SUT.
```

```
36      */
37     @Override
38     public void createConnection() {
39        con.openConnection();
40
41        receiveMonitor = new ReceiveMonitor();
42        sendMonitor = new SendMonitor(con.getOutputStream());
43        receiveThread = new ReceiveThread(receiveMonitor, con.
              getInputStream());
44        receiveThread.start();
45
46        sendMonitor.sendSignal("SIG_REQUEST_PACKETS");
47     }
48
49     public Connection getCon() {
50        return con;
51     }
52
53     @Override
54     public void changeConnection(String host, int port) {
55        closeConnection();
56
57        con.updateConnection(host, port);
58
59        createConnection();
60     }
61
62     @Override
63     public String getHost() {
64        return con.getHost();
65     }
66
67     @Override
68     public int getPort() {
69        return con.getPort();
70     }
71
72     @Override
73     public void closeConnection() {
74        sendMonitor.sendSignal("SIG_PACKETS_STOP_AND_RESET");
75        receiveThread.interrupt();
76        con.closeConnection();
77     }
78
79
80     @Override
81     public Packet receivePacket(long seqNo) throws
           InterruptedException {
82        return receiveMonitor.receivePacket(seqNo);
83     }
84
85     @Override
86     public long sendCommand(CommandType type, ModuleID moduleID,
87           Argument[] arguments, long execTime, PacketPrio priority)
                {
```

```
88          return sendMonitor.sendMessage(new CommandData(type, moduleID
                ,
89                  arguments), execTime, priority);
90      }
91
92      @Override
93      public long sendCommand(CommandType cmdType, ModuleID module,
94              ArrayList<Argument> arguments, long execTime, PacketPrio
                    priority) {
95          return sendMonitor.sendMessage(new CommandData(cmdType,
                module,
96                  arguments.toArray(new Argument[arguments.size()])),
                        execTime,
97                  priority);
98      }
99
100 }
```

template/BasicTemplate.java

```
1  package util.nominaltest2.model.template;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.concurrent.Executors;
6  import java.util.concurrent.ScheduledExecutorService;
7  import java.util.concurrent.TimeUnit;
8  import java.util.logging.Logger;
9
10 import org.simpleframework.xml.Attribute;
11 import org.simpleframework.xml.Element;
12 import org.simpleframework.xml.ElementList;
13 import org.simpleframework.xml.Path;
14
15 import util.nominaltest2.model.Scenario;
16 import util.nominaltest2.model.TestManager;
17 import util.nominaltest2.model.TreeItem;
18 import util.nominaltest2.model.errors.ErrorValue;
19 import util.nominaltest2.model.exceptions.FailedTestException;
20 import util.nominaltest2.model.expression.Expression;
21 import util.nominaltest2.model.expression.SingleExpression;
22 import util.nominaltest2.model.expression.singleExpressions.
       ArgumentSingleExpression;
23 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
24 import util.nominaltest2.model.results.PacketResult;
25 import util.nominaltest2.model.results.TestResult;
26 import util.nominaltest2.model.results.TestResult.testResultEnum;
27 import util.nominaltest2.model.validators.PacketValidator;
28
29 import common.dtusat2.ModuleID;
```

```
30 import common.dtusat2.packet.ConfirmationPacket;
31 import common.dtusat2.packet.Packet;
32 import common.dtusat2.packet.TeledataPacket;
33 import common.dtusat2.packet.Packet.PacketPrio;
34 import common.types.TypeMaps;
35 import common.types.argument.Argument;
36 import common.types.command.CommandType;
37 import common.types.parameter.Parameter;
38
39 public class BasicTemplate extends ParameterTemplate {
40
41     private final Logger logger = Logger.getLogger(getClass().
         getName());
42     private ScheduledExecutorService scheduleExecutor = Executors
43             .newSingleThreadScheduledExecutor();
44     private long timeout = Integer.MAX_VALUE;
45
46     boolean timedOut = false;
47
48     Thread thread;
49     Object synchronizer = new Object();
50
51     private static String thisName = "Basic Template ";
52     boolean sent = false;
53     @Attribute
54     String cmdName;
55     @Attribute
56     ModuleID moduleID;
57
58     @ElementList(name = "DTUSatParameterExpressions", required =
         false)
59     ArrayList<DTUSatParameterExpression> dtuSatParamExprs = new
         ArrayList<DTUSatParameterExpression>();
60
61     TestResult testResult = new TestResult(this, testResultEnum.NONE
         );
62
63     @Path("PacketValidators")
64     @ElementList(inline = true)
65     ArrayList<PacketValidator> packetValidators = new ArrayList<
         PacketValidator>();
66     ArrayList<PacketValidator> currentValidators = new ArrayList<
         PacketValidator>();
67
68     private long seqNo;
69
70     public BasicTemplate() {
71        id = TestManager.getNextBasicTemplateID();
72     }
73
74     public BasicTemplate(String cmdName, ModuleID moduleID,
75             ArrayList<Variable> variables) {
76        super(variables);
77        id = TestManager.getNextBasicTemplateID();
78        this.cmdName = cmdName;
```

```
79          this.moduleID = moduleID;
80
81          setUpDTUSatParameters();
82
83      }
84
85      public BasicTemplate(String cmdName, ModuleID moduleID,
86              ArrayList<Variable> variables,
87              ArrayList<DTUSatParameterExpression> dtuSatParamExprs) {
88          this(cmdName, moduleID, variables);
89          this.dtuSatParamExprs = dtuSatParamExprs;
90      }
91
92      /**
93       * Constructor used for cloning this BasicTemplate (by keeping
             the ID
94       * intact)
95       *
96       * @param string
97       * @param moduleID
98       * @param paramExprsCloned
99       * @param dtuSatParamExprsCloned
100      * @param id
101      */
102     private BasicTemplate(String cmdName, ModuleID moduleID,
103             ArrayList<Variable> variables,
104             ArrayList<DTUSatParameterExpression> dtuSatParamExprs, int
                    id) {
105         super(variables);
106         this.cmdName = cmdName;
107         this.moduleID = moduleID;
108         setUpDTUSatParameters();
109         this.dtuSatParamExprs = dtuSatParamExprs;
110         this.id = id;
111     }
112
113     public ArrayList<DTUSatParameterExpression> getdtuSatParamExprs
            () {
114         if (dtuSatParamExprs != null)
115             return dtuSatParamExprs;
116         else
117             return new ArrayList<DTUSatParameterExpression>();
118     }
119
120     /**
121      * Will initialise the exact number of {@link
             DTUSatParameterExpression},
122      * holding only the {@link Parameter}, needed for the given
123      * {@link CommandType} to be send to the DTUSat. This means one
             needs to
124      * assign {@link Expression} to these {@link Parameter} in order
             to run the
125      * program.
126      *
127      */
```

```
128     public void setUpDTUSatParameters() {
129         ArrayList<Parameter> parameters = TypeMaps.getCommandTypeMap
                ().getType(
130                 cmdName).getParameters();
131         dtuSatParamExprs = new ArrayList<DTUSatParameterExpression>()
                ;
132         if (parameters != null) {
133             for (Parameter param : parameters) {
134                 dtuSatParamExprs.add(new DTUSatParameterExpression(
                        param
135                         .getName()));
136             }
137         }
138     }
139 
140     /**
141      * Update the a {@link DTUSatParameterExpression}s expression
                with the given
142      * input. This will be matched on the name of the {@link
                Parameter}. In case
143      * no {@link DTUSatParameterExpression} exist with the given
                name, the
144      * methods will return false.
145      *
146      * @param paramName
147      *             the name of the {@link Parameter} which should be
                set.
148      * @param expr
149      *             the new expression to be used.
150      * @return <b>true</b> if it was able to set the new expression,
                otherwise
151      *         <b>false</b>.
152      */
153     public boolean setExpressionOnParameter(String paramName,
154             SingleExpression expr) {
155         for (DTUSatParameterExpression dtuSatParamExpr :
                dtuSatParamExprs) {
156             if (dtuSatParamExpr.getParameterName().equals(paramName))
                    {
157                 dtuSatParamExpr.setExpression(expr);
158                 return true;
159             }
160         }
161         return false;
162     }
163 
164     @Override
165     public BasicTemplate clone() {
166         ArrayList<Variable> variablesCloned = cloneVariables();
167 
168         // Fixing references from our expression that the parameters
                for the
169         // command are using.
170         ArrayList<DTUSatParameterExpression> dtuSatParamExprsCloned =
                cloneDTUSatParameters();
```

```
171         for (DTUSatParameterExpression dtuSatParamExpr :
                dtuSatParamExprsCloned) {
172             SingleExpression expr = dtuSatParamExpr.getExpression();
173             if (expr instanceof Variable) {
174                 String dtuSatParamExprName = ((Variable) expr).getName
                        ();
175                 for (Variable variable : variablesCloned) {
176                     if (dtuSatParamExprName.equals(variable.getName())
                            {
177                         dtuSatParamExpr.setExpression(variable);
178                     }
179                 }
180             }
181         }
182 
183         BasicTemplate clonedSelf = new BasicTemplate(new String(
                cmdName),
184                 moduleID, variablesCloned, dtuSatParamExprsCloned, id);
185         clonedSelf.setTimeout(new Long(timeout));
186 
187         // Clone packetvalidators
188         for (PacketValidator origPv : packetValidators) {
189             PacketValidator clonedPv = origPv.clone();
190             clonedPv.updateReferences(variablesCloned);
191             try {
192                 clonedSelf.addChild(clonedPv);
193             } catch (Exception e) {
194                 logger.severe(e.getMessage());
195             }
196         }
197 
198         return clonedSelf;
199     }
200 
201     /**
202      * Clones all {@link DTUSatParameterExpression} aswell as the
203      * {@link Expression}s found under them.
204      *
205      * @return A list containing the cloned {@link
                DTUSatParameterExpression}
206      */
207     private ArrayList<DTUSatParameterExpression>
            cloneDTUSatParameters() {
208         ArrayList<DTUSatParameterExpression> clonedExprs = new
                ArrayList<DTUSatParameterExpression>();
209         for (DTUSatParameterExpression expr : dtuSatParamExprs) {
210             clonedExprs.add(expr.clone());
211         }
212         return clonedExprs;
213     }
214 
215     @Override
216     public void send() {
217         scheduleExecutor = Executors.newSingleThreadScheduledExecutor
                ();
```

```
218         if (validateSelf(new ArrayList<ErrorValue>())) {
219
220             timedOut = false;
221             if (!Thread.currentThread().isInterrupted()) {
222                 try {
223                     setThread(Thread.currentThread());
224                     startTimeOutCount();
225                     logger.info("send called in BasicTemplate");
226                     CommandType cmdType = TypeMaps.getCommandTypeMap().
                            getType(
227                               cmdName);
228
229                     seqNo = TestManager.getSA().sendCommand(cmdType,
                            moduleID,
230                               convertExpressionsToDTUSatArguments(), 0,
231                               PacketPrio.NORMAL);
232
233                     sent = true;
234                     setThread(null);
235                 } catch (InterruptedException e) {
236                     logger
237                             .info("BasicTemplate experience an interrupt
                                    exception");
238                     // Thread.currentThread().interrupt();
239                     timedOut = true;
240                     return;
241                 } catch (Exception e) {
242                     logger.severe("Excepton in send() on BasicTemplate "
243                             + e.getMessage() + ", " + e.getStackTrace());
244                 }
245             } else {
246                 logger.severe("Couldn't validate before sending!");
247
248             }
249         }
250     }
251
252     /**
253      * Will set the current thread we are working on. Has been made
            thread safe,
254      * as more thread may change the value at a time.
255      *
256      * @param thread
257      */
258     private void setThread(Thread thread) {
259         synchronized (synchronizer) {
260             this.thread = thread;
261         }
262     }
263
264     /**
265      * Will convert the {@link DTUSatParameterExpression} into {
            @link Argument}s
266      * what we are able to send to the DTUSat with the given command
            . This is
```

```
267      * achieved by getting the {@link Expression}s DTUSat {@link
             Argument}
268      * stored upon it.
269      *
270      * @return A list containing the {@link Argument} to be send.
271      * @throws Exception
272      *             throws an exception when a value is not set.
273      */
274     private ArrayList<Argument> convertExpressionsToDTUSatArguments
             ()
275             throws Exception {
276         // if (validateParamsAndExprs(expressions)) {
277         ArrayList<Argument> dtuSatArgs = new ArrayList<Argument>();
278 
279         for (DTUSatParameterExpression paramExpr : dtuSatParamExprs)
                 {
280             if (paramExpr.getExpression() != null) {
281                 SingleExpression exprValue = paramExpr.getExpression()
282                         .getValue();
283                 if (exprValue != null
284                         && exprValue instanceof ArgumentSingleExpression)
                                 {
285                     dtuSatArgs.add(((ArgumentSingleExpression) exprValue
                             )
286                             .getDTUSatArgument());
287                 } else {
288                     throw new Exception(
289                             "Encountered an unbounded expression in a
                                 BasicTemplate when trying to send: "
290                                     + exprValue.toString());
291                 }
292             } else {
293                 throw new Exception("DTUSatParameterExpression:
                         expression "
294                         + paramExpr.getParameterName() + " not set");
295             }
296         }
297         return dtuSatArgs;
298         // }
299         // throw new Exception(
300         // "Encountered an unbounded expression in a BasicTemplate
                 when trying to send");
301     }
302 
303     @Override
304     protected boolean validateVariablesAndExprs(List<
             SingleExpression> expressions) {
305         if (super.validateVariablesAndExprs(expressions)) {
306             for (SingleExpression expression : expressions) {
307                 if (!(expression.getValue() instanceof
                         ArgumentSingleExpression)) {
308                     return false;
309                 }
310             }
311             return true;
```

```
312         }
313         return false;
314     }
315 
316     /**
317      * Check if the given ArrayList of {@link PacketResult} contain
           one which
318      * have <b>true</b> as its result.
319      *
320      * @param pResults
321      * @return
322      */
323     private boolean checkIfSuccesful(ArrayList<PacketResult>
          pResults) {
324         for (PacketResult packetResult : pResults) {
325             if (packetResult.getResult() == testResultEnum.PASS) {
326                 return true;
327             }
328         }
329         return false;
330     }
331 
332     /**
333      * Checks whether a received packet have an appropriate
           validator waiting
334      * for it. In case we are missing one, the method will fail. If
           a validator
335      * is waiting, lit will validate the given input.
336      *
337      * @param packet
338      *             must be a packet of type {@link ConfirmationPacket
           } or
339      *             {@link TeledataPacket}
340      * @return <b>true</b> in case one or more validators can
           validate the input
341      *         correctly.
342      * @throws Exception
343      */
344     private ArrayList<PacketResult> checkPacket(Packet packet)
          throws Exception {
345         ArrayList<PacketResult> packetResults = new ArrayList<
              PacketResult>();
346         ArrayList<PacketValidator> newValidators = new ArrayList<
              PacketValidator>();
347 
348         logger.info("In checkPacket: " + currentValidators.size()
349                 + " validators on hold");
350 
351         for (PacketValidator val : currentValidators) {
352             if (val.evaluteGuard()) {
353                 logger.info("Guard evaluated to true");
354                 PacketResult pResult = val.validate(packet);
355                 pResult.updatePacketResult();
356                 packetResults.add(pResult);
357                 if (pResult.getResult() == testResultEnum.PASS) {
```

```
358                 if (val.getChildren().size() == 0) {
359                     currentValidators = new ArrayList<PacketValidator
                            >();
360                     return packetResults;
361                 } else {
362                     newValidators.addAll(val.getPacketValidators());
363                 }
364             }
365         } else {
366             logger.info("Guard evaluated to false");
367             PacketResult pResult = val.setGuardFail();
368             packetResults.add(pResult);
369         }
370     }
371 
372     currentValidators = newValidators;
373     return packetResults;
374 }
375 
376 @Override
377 public Void call() throws Exception {
378     setThread(Thread.currentThread());
379     if (!Thread.currentThread().isInterrupted() && !timedOut) {
380         if (sent) {
381             currentValidators.addAll(packetValidators);
382             try {
383                 listenForAnswer();
384             } catch (FailedTestException e) {
385                 logger.severe(e.getDescription());
386             }
387             setThread(null);
388         } else {
389             testResult = new TestResult(this, testResultEnum.INCONC
                    );
390             logger
391                     .severe("In BasicTemplate: Trying to receive, but
                            I haven't sent!");
392             scheduleExecutor.shutdownNow();
393         }
394     }
395     scheduleExecutor.shutdownNow();
396     return null;
397 }
398 
399 /**
400  * Wait for a package to arrive for the TestCase, and check if
        we have any
401  * validators for it, which means we should continue on the
        validator tree.
402  *
403  * @throws Exception
404  */
405 private void listenForAnswer() throws Exception {
406     if (!Thread.currentThread().isInterrupted()) {
407         boolean succes = true;
```

```
408             Packet packet;
409             try {
410                 packet = TestManager.getSA().receivePacket(seqNo);
411             } catch (InterruptedException e) {
412                 Thread.currentThread().interrupt();
413                 scheduleExecutor.shutdownNow();
414                 return;
415             }
416 
417             ArrayList<PacketResult> packetResults = checkPacket(packet
                    );
418             testResult.addPacketResults(packetResults);
419             succes = checkIfSuccesful(packetResults);
420             if (succes && currentValidators.size() == 0) {
421                 testResult.setResult(true);
422             } else if (succes) {
423                 listenForAnswer();
424             } else {
425                 testResult.setResult(false);
426             }
427 
428             if (!Thread.currentThread().isInterrupted()) {
429                 notifyAboutChangedResults();
430             }
431         } else {
432             scheduleExecutor.shutdownNow();
433             return;
434         }
435         scheduleExecutor.shutdownNow();
436     }
437 
438     /**
439      * Notify observers that a change has happened and that they
                should react.
440      */
441     private void notifyAboutChangedResults() {
442         setChanged();
443         notifyObservers();
444     }
445 
446     @Override
447     public testResultEnum getEnumResult() {
448         return testResult.getEnumResult();
449     }
450 
451     @Override
452     public ArrayList<TreeItem> getChildren() {
453         return new ArrayList<TreeItem>(packetValidators);
454     }
455 
456     @Override
457     protected void doAddChild(TreeItem item) throws Exception {
458         if (item instanceof PacketValidator) {
459             packetValidators.add((PacketValidator) item);
460         }
```

```
461
462     }
463
464     @Override
465     protected void doDeleteChild(TreeItem child) throws Exception {
466         packetValidators.remove(child);
467     }
468
469     public String getCommandName() {
470         return cmdName;
471     }
472
473     /**
474      * Sets the name for the {@link CommandType} that should be send
             to the
475      * satellite. Furthermore it will set up the structure for the
             given
476      * command, by calling setUpDTUSatParameters as well as setting
             up the
477      * modules which is allowed by the {@link CommandType}.
478      *
479      * @param cmdName
480      *             the name for the new {@link CommandType} to be
             send.
481      */
482     public void setCommandName(String cmdName) {
483         this.cmdName = cmdName;
484         CommandType cmdType = TypeMaps.getCommandTypeMap().getType(
                 cmdName);
485         setUpDTUSatParameters();
486         if (cmdType.getLegalDestinations() == null
487                 || cmdType.getLegalDestinations().contains(moduleID)) {
488             return;
489         }
490         if (cmdType.getLegalDestinations().size() > 0) {
491             moduleID = cmdType.getLegalDestinations().get(0);
492         } else {
493             moduleID = null;
494         }
495         setChanged();
496         notifyObservers();
497
498     }
499
500     public ModuleID getModuleID() {
501         return moduleID;
502     }
503
504     public void setModuleID(ModuleID moduleID) {
505         this.moduleID = moduleID;
506         setChanged();
507         notifyObservers();
508     }
509
510     @Override
```

```
511     public String toString() {
512        return thisName + super.toString();
513     }
514
515     @Override
516     public String getNameOrIDString() {
517        return thisName + (getName().equals("") ? id : getName());
518     }
519
520     public void setTimeout(long timeout) {
521        this.timeout = timeout;
522     }
523
524     public long getTimeout() {
525        return timeout;
526     }
527
528     /**
529      * Causes a schedule task to be started, which will run after
            the timeout
530      * window has expired.
531      *
532      * Is using a {@link ScheduledExecutorService} to perform this
            task.
533      */
534     private void startTimeOutCount() {
535        logger.info("Start timeout count with timeout set to : " +
              timeout);
536        scheduleExecutor.schedule(new Runnable() {
537           @Override
538           public void run() {
539
540              logger.info("Timeout expired. Interrupting test");
541              synchronized (synchronizer) {
542                 if (thread != null && thread.isAlive()) {
543                    thread.interrupt();
544                    testResult.setTimedOut();
545                 }
546              }
547           }
548        }, timeout, TimeUnit.MILLISECONDS);
549     }
550
551     protected boolean validateSelf(List<ErrorValue> errorList) {
552        if (super.validateSelf(errorList)) {
553           CommandType cmdType = validateCmdName(errorList);
554           if (cmdType != null) {
555              validateModuleID(errorList, cmdType);
556           }
557
558           for (DTUSatParameterExpression dtuExpr : dtuSatParamExprs)
                 {
559              dtuExpr.validate(errorList, this);
560           }
561
```

```
562             if (packetValidators == null || packetValidators.size() ==
                    0) {
563                 errorList.add(new ErrorValue(
564                         "No PacketValidator has been added to
                            BasicTemplate",
565                         this));
566             } else {
567                 for (PacketValidator pv : packetValidators) {
568                     pv.validate(errorList, this);
569                 }
570             }
571             return errorList.size() == 0;
572         }
573         return false;
574     }
575 
576     @Override
577     public void validateRecursively(List<ErrorValue> errorList) {
578         validateSelf(errorList);
579     }
580 
581     /**
582      * Checks that the cmdName actually exists in the xml-files
583      *
584      * @param errorList
585      */
586     private CommandType validateCmdName(List<ErrorValue> errorList)
            {
587         if (cmdName != null) {
588             CommandType cmdType = TypeMaps.getCommandTypeMap().getType
                    (cmdName);
589             if (cmdType != null) {
590                 return cmdType;
591             } else {
592                 errorList.add(new ErrorValue("The command: " + cmdName
593                         + " doesn't exist", this));
594             }
595         } else {
596             errorList.add(new ErrorValue("Missing command in
                        BasicTemplate",
597                     this));
598         }
599         return null;
600     }
601 
602     private void validateModuleID(List<ErrorValue> errorList,
603             CommandType cmdType) {
604         if (moduleID != null) {
605             ArrayList<ModuleID> validModules = cmdType.
                        getLegalDestinations();
606             if (validModules == null || validModules.size() == 0) {
607                 // All modules valid
608             } else {
609                 boolean moduleIdMatches = false;
610                 for (ModuleID moduleID : validModules) {
```

```
611                 if (moduleID.name().toString().equals(moduleID.
                        toString())) {
612                     moduleIdMatches = true;
613                     break;
614                 }
615             }
616             if (!moduleIdMatches) {
617                 errorList.add(new ErrorValue("The moduleID: " +
                        moduleID
618                     + " doesn't exist", this));
619             }
620         }
621     } else {
622         errorList.add(new ErrorValue("Missing moduleID in
                BasicTemplate",
623             this));
624     }
625 }
626
627 @Override
628 public boolean isVariableInUse(Variable variable) {
629     for (DTUSatParameterExpression paramExpr : dtuSatParamExprs)
            {
630         if (paramExpr.getExpression() == variable) {
631             return true;
632         }
633     }
634
635     for (PacketValidator pv : packetValidators) {
636         if (pv.isVariableInUse(variable)) {
637             return true;
638         }
639     }
640
641     return false;
642 }
643
644 @Override
645 public void setupObserverObservableHierarchy() {
646     super.setupObserverObservableHierarchy();
647     for (PacketValidator pv : packetValidators) {
648         pv.addObserver(this);
649         pv.setupObserverObservableHierarchy();
650     }
651
652 }
653
654 @Override
655 public void interruptFuture() {
656 }
657
658 @Override
659 public ArrayList<TestResult> getTestResults() {
660     ArrayList<TestResult> results = new ArrayList<TestResult>();
661     if (testResult != null) {
```

```
662             results.add(testResult);
663         }
664         return results;
665     }
666 
667 }
```

template/DefinitionTemplate.java

```
1  package util.nominaltest2.model.template;
2  
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.logging.Logger;
6  
7  import org.simpleframework.xml.Element;
8  import org.simpleframework.xml.ElementUnion;
9  
10 import util.nominaltest2.model.TestManager;
11 import util.nominaltest2.model.TreeItem;
12 import util.nominaltest2.model.errors.ErrorValue;
13 import util.nominaltest2.model.expression.SingleExpression;
14 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
15 import util.nominaltest2.model.results.TestResult;
16 import util.nominaltest2.model.results.TestResult.testResultEnum;
17 import util.nominaltest2.model.test.ApplyTest;
18 import util.nominaltest2.model.test.ConcurrentCollectionTest;
19 import util.nominaltest2.model.test.DelayTest;
20 import util.nominaltest2.model.test.ForTest;
21 import util.nominaltest2.model.test.SequentialCollectionTest;
22 import util.nominaltest2.model.test.Test;
23 
24 public class DefinitionTemplate extends ParameterTemplate {
25     private final Logger logger = Logger.getLogger(getClass().
          getName());
26 
27     @ElementUnion({
28             @Element(name = "ApplyTest", type = ApplyTest.class,
                  required = false),
29             @Element(name = "ConcurrentCollectionTest", type =
                  ConcurrentCollectionTest.class, required = false),
30             @Element(name = "SequentialCollectionTest", type =
                  SequentialCollectionTest.class, required = false),
31             @Element(name = "DelayTest", type = DelayTest.class,
                  required = false),
32             @Element(name = "ForTest", type = ForTest.class, required
                  = false) })
33     Test test;
34     private static String thisName = "Definition Template ";
35 
```

```
36    public DefinitionTemplate(ArrayList<Variable> variables, Test
          test) {
37       super(variables);
38       id = TestManager.getNextDefinitionTemplateID();
39       try {
40          addChild(test);
41       } catch (Exception e) {
42          logger.severe(e.getMessage());
43       }
44    }
45
46    public DefinitionTemplate() {
47       id = TestManager.getNextDefinitionTemplateID();
48    }
49
50    public DefinitionTemplate(ArrayList<Variable> variables, Test
          test, int id) {
51       super(variables);
52       this.id = id;
53       try {
54          addChild(test);
55       } catch (Exception e) {
56          logger.severe(e.getMessage());
57       }
58    }
59
60    private void setId(int id) {
61       this.id = id;
62    }
63
64    @Override
65    public Void call() throws Exception {
66       if (!Thread.currentThread().isInterrupted()) {
67          if (test != null) {
68             test.call();
69          }
70       }
71       return null;
72    }
73
74    @Override
75    public ParameterTemplate clone() {
76       if (test != null) {
77          return new DefinitionTemplate(variables, test.clone(), id)
                ;
78       } else {
79          DefinitionTemplate clonedSelf = new DefinitionTemplate();
80          for (Variable variable : variables) {
81             clonedSelf.addVariable(variable);
82          }
83          clonedSelf.setId(id);
84          return clonedSelf;
85       }
86    }
87
```

```
88      @Override
89      public void send() {
90          if (!Thread.currentThread().isInterrupted()) {
91              test.send();
92          }
93      }
94
95      @Override
96      public testResultEnum getEnumResult() {
97          if (test != null) {
98              return test.getEnumResult();
99          } else {
100             return testResultEnum.INCONC;
101         }
102     }
103
104     @Override
105     public ArrayList<TreeItem> getChildren() {
106         ArrayList<TreeItem> arrList = new ArrayList<TreeItem>();
107         if (test != null) {
108             arrList.add(test);
109         }
110         return arrList;
111     }
112
113     @Override
114     protected void doAddChild(TreeItem item) throws Exception {
115         if (item instanceof Test && test == null) {
116             test = (Test) item;
117         } else {
118             throw new Exception(
119                     "Can't add that type or an extra child to
                            DefinitionTemplate");
120         }
121
122     }
123
124     @Override
125     protected void doDeleteChild(TreeItem child) throws Exception {
126         if (test == child) {
127             test = null;
128         }
129
130     }
131
132     @Override
133     public String toString() {
134         return thisName + super.toString();
135     }
136
137     @Override
138     public String getNameOrIDString() {
139         return thisName + (getName().equals("") ? id : getName());
140     }
141
```

```
142     @Override
143     protected boolean validateSelf(List<ErrorValue> errorList) {
144        if (super.validateSelf(errorList)) {
145           if (test == null) {
146              errorList.add(new ErrorValue(
147                    "No test has been added to the DefinitionTemplate
                         ",
148                    this));
149              return false;
150           }
151           return true;
152        } else {
153           return false;
154        }
155     }
156
157     @Override
158     public void validateRecursively(List<ErrorValue> errorList) {
159        validateSelf(errorList);
160        if (test != null)
161           test.validateRecursively(errorList);
162
163     }
164
165     @Override
166     public boolean isVariableInUse(Variable variable) {
167        if (test != null) {
168           ArrayList<ApplyTest> applyTests = test.
                getDecendentApplyTest();
169           for (ApplyTest applyTest : applyTests) {
170              for (SingleExpression paramExpr : applyTest.
                   getExpressions()) {
171                 if (paramExpr instanceof Variable) {
172                    if (paramExpr == variable) {
173                       return true;
174                    }
175                 }
176
177              }
178           }
179        }
180        return false;
181     }
182
183     @Override
184     public void interruptFuture() {
185        logger.info("Interrupting Future in DefinitionTemplate");
186        if (test != null) {
187           test.interruptFuture();
188        }
189     }
190
191     @Override
192     public ArrayList<TestResult> getTestResults() {
193        ArrayList<TestResult> results = new ArrayList<TestResult>();
```

```
194         if (test != null) {
195             results.addAll(test.getTestResults());
196         }
197         return results;
198     }
199 
200 }
```

template/DTUSatParameterArgument.java

```
1  package util.nominaltest2.model.template;
2  
3  import org.simpleframework.xml.Attribute;
4  
5  import common.types.argument.Argument;
6  import common.types.parameter.Parameter;
7  
8  /**
9   * Encapsulates a DTUSat {@link Parameter} with its corresponding
10  * {@link Argument}
11  *
12  */
13 public class DTUSatParameterArgument {
14 
15     @Attribute(required = false)
16     String paramName;
17 
18     @Attribute(required = false)
19     Argument argument;
20 
21     public DTUSatParameterArgument(String paramName) {
22         this.paramName = paramName;
23     }
24 
25     public String getParameterName() {
26         return paramName;
27     }
28 
29     public void setArgument(Argument argument) {
30         this.argument = argument;
31     }
32 
33     public Argument getArgument() {
34         return argument;
35     }
36 
37 }
```

template/DTUSatParameterExpression.java

```
1  package util.nominaltest2.model.template;
2
3  import java.util.List;
4  import java.util.logging.Logger;
5
6  import org.simpleframework.xml.Attribute;
7  import org.simpleframework.xml.Element;
8  import org.simpleframework.xml.ElementUnion;
9
10 import common.types.argument.Argument;
11 import common.types.parameter.Parameter;
12
13 import util.nominaltest2.model.errors.ErrorValue;
14 import util.nominaltest2.model.expression.Expression;
15 import util.nominaltest2.model.expression.SingleExpression;
16 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
17 import util.nominaltest2.model.expression.singleExpressions.
       argument.EnumArgumentSingleExpression;
18 import util.nominaltest2.model.expression.singleExpressions.
       argument.IntArgumentSingleExpression;
19 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.MinusArithmeticSingleExpression;
20 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.MultiplicationArithmeticSingleExpression;
21 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.PlusArithmeticSingleExpression;
22
23 /**
24  * Encapsulates a given {@link Parameter} with the {@link
        Expression} that
25  * should be send along it, as it should contain the {@link
        Argument}.
26  *
27  */
28 public class DTUSatParameterExpression {
29     private final Logger logger = Logger.getLogger(getClass().
          getName());
30
31     @Attribute
32     String paramName;
33
34     @ElementUnion( {
35             @Element(name = "EnumArgumentExpression", type =
                EnumArgumentSingleExpression.class, required = false),
36             @Element(name = "IntArgumentExpression", type =
                IntArgumentSingleExpression.class, required = false),
37             @Element(name = "Variable", type = Variable.class,
                required = false),
38             @Element(name = "MinusArithmeticSingleExpression", type =
                MinusArithmeticSingleExpression.class, required =
                false),
```

```
39          @Element(name = "MultiplicationArithmeticSingleExprssion",
                type = MultiplicationArithmeticSingleExpression.class
                , required = false),
40          @Element(name = "PlusArithmeticSingleExpression", type =
                PlusArithmeticSingleExpression.class, required = false
                ) })
41      SingleExpression expr;
42
43      public DTUSatParameterExpression(
44              @Attribute(name = "paramName") String paramName) {
45          this.paramName = paramName;
46      }
47
48      public DTUSatParameterExpression(String paramName,
            SingleExpression expr) {
49          this(paramName);
50          this.expr = expr;
51      }
52
53      public String getParameterName() {
54          return paramName;
55      }
56
57      public void setExpression(SingleExpression expr) {
58          this.expr = expr;
59      }
60
61      public SingleExpression getExpression() {
62          return expr;
63      }
64
65      @Override
66      public DTUSatParameterExpression clone() {
67          return new DTUSatParameterExpression(new String(paramName),
68                  (SingleExpression) expr.clone());
69      }
70
71      @Override
72      public String toString() {
73          return paramName;
74      }
75
76      /**
77       * Check whether the {@link Expression} has been set.
78       *
79       * @param errorList
80       *            the list containing all errors discovered.
81       * @param basicTemplate
82       *            the {@link BasicTemplate} in which the
83       *            {@link DTUSatParameterExpression} is contained.
84       */
85      public void validate(List<ErrorValue> errorList, BasicTemplate
            basicTemplate) {
86          logger.info("valiging in DTUSatParameterExpression: " +
                paramName
```

```
87              + ", expr=" + expr);
88          if (expr == null) {
89              errorList.add(new ErrorValue("The parameter " + paramName
90                      + " has not been mapped to an expression in
                            BasicTemplate",
91                      basicTemplate));
92          }
93
94      }
95  }
```

template/ParameterTemplate.java

```
1  package util.nominaltest2.model.template;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.logging.Logger;
6
7  import org.simpleframework.xml.ElementList;
8  import org.simpleframework.xml.Path;
9
10 import util.nominaltest2.model.errors.ErrorValue;
11 import util.nominaltest2.model.expression.Expression;
12 import util.nominaltest2.model.expression.SingleExpression;
13 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
14
15 public abstract class ParameterTemplate extends Template {
16     private final Logger logger = Logger.getLogger(getClass().
           getName());
17     @Path("Variables")
18     @ElementList(required = false, inline = true)
19     protected ArrayList<Variable> variables = new ArrayList<Variable
           >();
20
21     /**
22      * Determines if the {@link ParameterTemplate} can be referenced
            .
23      */
24     private boolean isRoot = false;
25     protected int id;
26
27     public ParameterTemplate() {
28     }
29
30     public ParameterTemplate(ArrayList<Variable> variables) {
31         this();
32         for (Variable variable : variables) {
33             addVariable(variable);
34         }
```

```
35      }
36
37      /**
38       * Set whether the {@link ParameterTemplate} should be able to
             be
39       * referenced.
40       *
41       * @param isRoot
42       */
43      public void setRoot(boolean isRoot) {
44          this.isRoot = isRoot;
45      }
46
47      public boolean isRoot() {
48          return isRoot;
49      }
50
51      @Override
52      public ArrayList<Variable> getVariables() {
53          return variables;
54      }
55
56      public void addVariable(Variable pe) {
57          variables.add(pe);
58          pe.addObserver(this);
59          setChanged();
60          notifyObservers();
61      }
62
63      /**
64       * Clones the {@link Variable}s found on the {@link Template}.
65       *
66       * @return a list containing the cloned {@link Variable}s.
67       */
68      protected ArrayList<Variable> cloneVariables() {
69          ArrayList<Variable> clonedExprs = new ArrayList<Variable>(
                 variables
70                   .size());
71          for (Variable variable : variables) {
72              clonedExprs.add(variable.clone());
73          }
74          return clonedExprs;
75      }
76
77      @Override
78      public boolean mapApplyToTemplate(List<SingleExpression>
             applyExprs) {
79          logger.finer("In ParameterTemplate.mapApplyToTemplate:
                 variables.size()= "
80                          + variables.size()
81                          + ", applyExprs.size()="
82                          + applyExprs.size()
83                          + " and parameter tempalte is "
84                          + this);
85          ArrayList<Variable> variables = getVariables();
```

```
86          if (variables.size() == applyExprs.size()) {
87              for (int i = 0; i < variables.size(); i++) {
88                  variables.get(i).setParentExpr(applyExprs.get(i));
89              }
90              return true;
91          } else {
92              logger.severe("Error in mapApplyToTemplate: variables.size
                    ()= "
93                      + variables.size() + ", applyExprs.size()="
94                      + applyExprs.size());
95              return false;
96          }
97      }
98
99      /**
100      * Check if we have given as many {@link Expression}s as there
             are
101      * {@link Variable}s.
102      *
103      * @param expressions
104      *            the list of {@link Expression} to be checked.
105      * @return <b>true</b> if there was an equal amount of {@link
             Expression}s
106      *         and {@link Variable}s
107      */
108     protected boolean validateVariablesAndExprs(
109             List<SingleExpression> expressions) {
110         return expressions.size() == variables.size();
111     }
112
113     public void removeVariable(Variable p) {
114         variables.remove(p);
115
116     }
117
118     @Override
119     public String toString() {
120         String rStr = super.toString().equals("") ? "" + id : super.
                toString();
121         if (variables.size() == 0)
122             return rStr;
123         rStr += " (";
124         for (Variable e : variables) {
125             rStr += (e.getName().equals("") ? "_" : e.getName()) + ",
                    ";
126         }
127         rStr = rStr.substring(0, rStr.length() - 2) + ")";
128         return rStr;
129     }
130
131     public abstract String getNameOrIDString();
132
133     protected boolean validateSelf(List<ErrorValue> errorList) {
134         for (int i = 0; i < variables.size(); i++) {
135             for (int j = i + 1; j < variables.size(); j++) {
```

```
136             if (variables.get(i).getName().equals(
137                     variables.get(j).getName())) {
138                 errorList.add(new ErrorValue(
139                         "Contains two or more variables with same name
                            ",
140                         this));
141                 return false;
142             }
143         }
144     }
145
146     return true;
147 }
148
149 @Override
150 public void validateRecursively(List<ErrorValue> errorList) {
151
152 }
153
154 @Override
155 public void setupObserverObservableHierarchy() {
156     for (Variable pe : variables) {
157         pe.addObserver(this);
158     }
159
160 }
161
162 /**
163  * Check whether the Variable is being used. Is used to check it
        a variable
164  * can be deleted.
165  *
166  * @param variable
167  *            the variable to be checked.
168  * @return <b>true</b> if the variable is in use.
169  */
170 public abstract boolean isVariableInUse(Variable variable);
171 }
```

template/ReferenceTemplate.java

```
1  package util.nominaltest2.model.template;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.concurrent.Future;
6  import java.util.logging.Logger;
7
8  import org.simpleframework.xml.Element;
9  import org.simpleframework.xml.ElementUnion;
10
```

```
11 import util.nominaltest2.model.TestManager;
12 import util.nominaltest2.model.TreeItem;
13 import util.nominaltest2.model.errors.ErrorValue;
14 import util.nominaltest2.model.expression.SingleExpression;
15 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
16 import util.nominaltest2.model.results.TestResult;
17 import util.nominaltest2.model.results.TestResult.testResultEnum;
18
19 public class ReferenceTemplate extends Template {
20    private final Logger logger = Logger.getLogger(getClass().
          getName());
21
22    Future<Void> futureRes;
23
24    @ElementUnion( {
25          @Element(name = "BasicTemplate", type = BasicTemplate.
                class, required = false),
26          @Element(name = "DefinitionTemplate", type =
                DefinitionTemplate.class, required = false) })
27    ParameterTemplate templateOrig;
28    ParameterTemplate templateCloned;
29
30    public ReferenceTemplate(ParameterTemplate templateOrig) {
31       try {
32          addChild(templateOrig);
33       } catch (Exception e) {
34          logger.severe(e.getMessage());
35       }
36    }
37
38    public ReferenceTemplate() {
39
40    }
41
42    @Override
43    public Void call() throws Exception {
44       if (!Thread.currentThread().isInterrupted()) {
45          if (templateCloned != null) {
46             try {
47                futureRes = TestManager.getExecutor()
48                      .submit(templateCloned);
49                futureRes.get();
50             } catch (InterruptedException e) {
51                logger.info("In ReferenceTemplate :
                      InterruptedException occurred");
52                Thread.currentThread().interrupt();
53                return null;
54             }
55          } else {
56             logger.severe("templateCloned is null in call() on
                   ApplyTest");
57          }
58       }
59       return null;
```

```
60      }
61
62      @Override
63      public testResultEnum getEnumResult() {
64          if (templateCloned != null) {
65              return templateCloned.getEnumResult();
66          } else {
67              return testResultEnum.NONE;
68          }
69      }
70
71      @Override
72      public ArrayList<TreeItem> getChildren() {
73          ArrayList<TreeItem> rList = new ArrayList<TreeItem>();
74          if (templateCloned != null) {
75              rList.add(templateCloned);
76          } else if (templateOrig != null) {
77              rList.add(templateOrig);
78          }
79          return rList;
80      }
81
82      public ParameterTemplate getOriginalParameterTemplate() {
83          return templateOrig;
84      }
85
86      @Override
87      public ArrayList<Variable> getVariables() {
88          if (templateOrig != null) {
89              return templateOrig.getVariables();
90          }
91          return new ArrayList<Variable>();
92      }
93
94      @Override
95      public void send() {
96          if (!Thread.currentThread().isInterrupted()) {
97
98              if (validateSelf(new ArrayList<ErrorValue>())
99                      && templateOrig instanceof ParameterTemplate) {
100                 templateCloned = (ParameterTemplate) templateOrig.clone
                        ();
101                 templateCloned.addObserver(this);
102                 // if (mapApplyToTemplate(expressions)) {
103                 templateCloned.send();
104                 // }
105             }
106         }
107     }
108
109     @Override
110     public boolean mapApplyToTemplate(List<SingleExpression>
            applyExprs) {
111         if (templateOrig != null && templateOrig instanceof
                ParameterTemplate) {
```

```
112             if (templateCloned != null) {
113                 templateCloned.mapApplyToTemplate(applyExprs);
114             }
115             return templateOrig.mapApplyToTemplate(applyExprs);
116         }
117         return false;
118     }
119
120     @Override
121     protected void doAddChild(TreeItem item) throws Exception {
122         if (item instanceof ParameterTemplate) {
123             templateOrig = (ParameterTemplate) item;
124         } else {
125             throw new Exception("Can't add that type to
                    ReferenceTemplate");
126         }
127
128     }
129
130     @Override
131     protected void doDeleteChild(TreeItem child) {
132         if (templateOrig == child) {
133             templateOrig = null;
134             templateCloned = null;
135         }
136
137     }
138
139     @Override
140     public String toString() {
141         return "Reference: "
142                 + (templateOrig == null ? "<no reference set>" :
                        templateOrig
143                         .getNameOrIDString());
144     }
145
146     @Override
147     public Template clone() {
148         ReferenceTemplate clonedSelf = new ReferenceTemplate(
149                 (ParameterTemplate) templateOrig.clone());
150         return clonedSelf;
151     }
152
153     @Override
154     protected boolean validateSelf(List<ErrorValue> errorList) {
155         if (templateOrig == null) {
156             errorList.add(new ErrorValue(
157                     "No reference has been set in ReferenceTemplate",
                            this));
158             return false;
159         }
160         return true;
161     }
162
163     @Override
```

```
164     public void validateRecursively(List<ErrorValue> errorList) {
165         validateSelf(errorList);
166         // Don't validate any further: Assume templates are validated
                separately
167         // through rootTemplates
168 
169     }
170 
171     @Override
172     public void setupObserverObservableHierarchy() {
173 
174     }
175 
176     public void interruptFuture() {
177         logger.info("Interrupting Futures in ReferenceTemplate");
178         if (futureRes != null) {
179             futureRes.cancel(true);
180         }
181         if (templateCloned != null) {
182             templateCloned.interruptFuture();
183         }
184     }
185 
186     @Override
187     public ArrayList<TestResult> getTestResults() {
188         ArrayList<TestResult> results = new ArrayList<TestResult>();
189         if (templateCloned != null) {
190             results.addAll(templateCloned.getTestResults());
191         }
192         return results;
193     }
194 }
```

template/Template.java

```
1  package util.nominaltest2.model.template;
2 
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.concurrent.Callable;
6  import java.util.concurrent.FutureTask;
7 
8  import util.nominaltest2.model.TreeItem;
9  import util.nominaltest2.model.errors.ErrorValue;
10 import util.nominaltest2.model.expression.Expression;
11 import util.nominaltest2.model.expression.SingleExpression;
12 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
13 import util.nominaltest2.model.results.TestResult;
14 import util.nominaltest2.model.results.TestResult.testResultEnum;
15 
```

```
16 public abstract class Template extends TreeItem implements Callable
       <Void> {
17
18     /**
19      * Will map the given {@link Expression} down onto the {@link
            Variable}s
20      * defined on the template in a 1:1 fashion, making the given
21      * {@link Expression}s the parent of the {@link Variable}s.
22      *
23      * @param applyExprs
24      *             the {@link Expression} that needs to be mapped
            down on the
25      *             {@link Variable}s
26      * @return
27      */
28     public abstract boolean mapApplyToTemplate(List<SingleExpression
           > applyExprs);
29
30     /**
31      * Will send that content defined to be send on the current
            objet.
32      */
33     public abstract void send();
34
35     public abstract ArrayList<Variable> getVariables();
36
37     @Override
38     public abstract Template clone();
39
40     /**
41      * Validate that there are no illegal setting in the test system
            , that can
42      * caused an exception. This will be used when the test scenario
            is being
43      * executed.
44      *
45      * @param errorList
46      *             A list that should be filled up when errors are
            spotted.
47      * @return <b>false</b> in case an error is located.
48      */
49     protected abstract boolean validateSelf(List<ErrorValue>
           errorList);
50
51     /**
52      * Is used to validate at any time whether there are errors in
            the system.
53      *
54      * @param errorList
55      *             A list that should be filled up when errors are
            spotted.
56      */
57     public abstract void validateRecursively(List<ErrorValue>
           errorList);
58
```

```
59      /**
60       * Sets up the Observer-Observable hierarchy, insuring that we
             will notify
61       * all the way through the hierarchy when making changes to the
             lower
62       * levels.
63       */
64      public abstract void setupObserverObservableHierarchy();
65
66      /**
67       * In case a {@link FutureTask} is currently running, this will
             be
68       * interrupted if deemed possible. Otherwise we would call
             interruptFuture()
69       * on the children aswell.
70       */
71      public abstract void interruptFuture();
72
73
74      public abstract ArrayList<TestResult> getTestResults();
75  }
```

test/ApplyTest.java

```
1   package util.nominaltest2.model.test;
2
3   import java.util.ArrayList;
4   import java.util.List;
5   import java.util.logging.Logger;
6
7   import org.simpleframework.xml.Element;
8   import org.simpleframework.xml.ElementListUnion;
9   import org.simpleframework.xml.ElementUnion;
10  import org.simpleframework.xml.ElementList;
11  import org.simpleframework.xml.Path;
12
13  import util.nominaltest2.model.TreeItem;
14  import util.nominaltest2.model.errors.ErrorValue;
15  import util.nominaltest2.model.expression.SingleExpression;
16  import util.nominaltest2.model.expression.singleExpressions.
        Variable;
17  import util.nominaltest2.model.expression.singleExpressions.
        argument.EnumArgumentSingleExpression;
18  import util.nominaltest2.model.expression.singleExpressions.
        argument.IntArgumentSingleExpression;
19  import util.nominaltest2.model.expression.singleExpressions.
        arithmetic.MinusArithmeticSingleExpression;
20  import util.nominaltest2.model.expression.singleExpressions.
        arithmetic.MultiplicationArithmeticSingleExpression;
21  import util.nominaltest2.model.expression.singleExpressions.
        arithmetic.PlusArithmeticSingleExpression;
```

```
22 import util.nominaltest2.model.results.TestResult;
23 import util.nominaltest2.model.results.TestResult.testResultEnum;
24 import util.nominaltest2.model.template.BasicTemplate;
25 import util.nominaltest2.model.template.DefinitionTemplate;
26 import util.nominaltest2.model.template.ParameterTemplate;
27 import util.nominaltest2.model.template.ReferenceTemplate;
28 import util.nominaltest2.model.template.Template;
29
30 public class ApplyTest extends Test {
31     private final Logger logger = Logger.getLogger(getClass().
          getName());
32
33     @Path("ApplyExpressions")
34     @ElementListUnion(value = {
35             @ElementList(entry = "EnumArgumentExpression", type =
                  EnumArgumentSingleExpression.class, inline = true,
                  required = false),
36             @ElementList(entry = "IntArgumentExpression", type =
                  IntArgumentSingleExpression.class, inline = true,
                  required = false),
37             @ElementList(entry = "Variable", type = Variable.class,
                  inline = true, required = false),
38             @ElementList(entry = "MinusArithmeticSingleExpression",
                  type = MinusArithmeticSingleExpression.class, inline =
                   true, required = false),
39             @ElementList(entry = "
                  MultiplicationArithmeticSingleExprssion", type =
                  MultiplicationArithmeticSingleExpression.class, inline
                   = true, required = false),
40             @ElementList(entry = "PlusArithmeticSingleExpression",
                  type = PlusArithmeticSingleExpression.class, inline =
                  true, required = false) })
41     private ArrayList<SingleExpression> applyExprs = new ArrayList<
          SingleExpression>();
42
43     @ElementUnion( {
44             @Element(name = "ReferenceTemplate", type =
                  ReferenceTemplate.class, required = false),
45             @Element(name = "BasicTemplate", type = BasicTemplate.
                  class, required = false),
46             @Element(name = "DefinitionTemplate", type =
                  DefinitionTemplate.class, required = false) })
47     private Template template;
48
49     public ApplyTest(Template template, ArrayList<SingleExpression>
          applyExprs)
50             throws Exception {
51         if (template instanceof ParameterTemplate
52                 && ((ParameterTemplate) template).isRoot()) {
53             throw new Exception(
54                     "Trying to use a template without referencing it");
55         }
56         this.applyExprs = applyExprs;
57         this.addChild(template);
58     }
```

```
59
60      public ApplyTest() {
61
62      }
63
64      @Override
65      public testResultEnum getEnumResult() {
66          if (template != null) {
67              return template.getEnumResult();
68          } else {
69              return testResultEnum.NONE;
70          }
71      }
72
73      @Override
74      public Void call() throws Exception {
75          if (!Thread.currentThread().isInterrupted()) {
76              if (template != null) {
77                  try {
78                      return template.call();
79                  } catch (InterruptedException e) {
80                      logger.info("In ApplyTest: InterruptedException
                            occured");
81                      Thread.currentThread().interrupt();
82                      return null;
83                  }
84              }
85          }
86          return null;
87      }
88
89      /**
90       * Will apply the given {@link SingleExpression} on the given
             index. In case
91       * the index is out of bound, no action will be performed.
92       *
93       * @param index
94       * @param singleExpression
95       */
96      public void setSingleExpressionOnIndex(int index,
97              SingleExpression singleExpression) {
98          if (index < applyExprs.size() && index >= 0) {
99              applyExprs.set(index, singleExpression);
100         }
101     }
102
103     @Override
104     public void send() {
105         if (!Thread.currentThread().isInterrupted()
106                 && actuallyMapExpressionsToTemplateAndValidate(new
                        ArrayList<ErrorValue>())) {
107             template.send();
108         } else {
109             logger.severe("Couldn't validate before sending!");
110         }
```

```
111     }
112
113     /**
114      * Returns true if there are no null applyExpressions
115      *
116      * @return
117      */
118     private boolean noNullApplyExpressions() {
119        for (SingleExpression expr : applyExprs) {
120           if (expr == null) {
121              return false;
122           }
123        }
124        return true;
125     }
126
127     @Override
128     public ArrayList<TreeItem> getChildren() {
129        ArrayList<TreeItem> rList = new ArrayList<TreeItem>();
130        if (template != null) {
131           rList.add(template);
132        }
133        return rList;
134     }
135
136     @Override
137     protected void doAddChild(TreeItem item) throws Exception {
138        logger.info("Adding child to ApplyTest");
139        if (item instanceof Template && template == null) {
140           template = (Template) item;
141           updateExpressionsFromChild();
142        } else {
143           throw new Exception(
144                 "Can't add that type or an extra child to ApplyTest"
                      );
145        }
146     }
147
148     @Override
149     protected void doDeleteChild(TreeItem child) {
150        if (template == child) {
151           template = null;
152           updateExpressionsFromChild();
153        }
154
155     }
156
157     /**
158      * Will get the {@link SingleExpression} on the given index.
159      *
160      * @param index
161      *            the place to look up
162      * @return {@link SingleExpression} located on the index
163      */
164     public SingleExpression getExpression(int index) {
```

```
165         if (applyExprs != null && applyExprs.size() > index) {
166             return applyExprs.get(index);
167         }
168         return null;
169     }
170 
171     public List<SingleExpression> getExpressions() {
172         return applyExprs;
173     }
174 
175     /**
176      * Will update our array over the {@link SingleExpression}
               located in the
177      * {@link ApplyTest} to correspond to a new template. This
               method will not
178      * delete all {@link SingleExpression} already set, but shorten
               the list to
179      * the necessary length keeping old {@link SingleExpression}
               intact, so they
180      * wont need to be redone. Or increase it in size with indexes
               set to
181      * <code>null</code>.
182      */
183     public void updateExpressionsFromChild() {
184         logger.info("updateExpressionsFromChild called in ApplyTest")
                ;
185         if (template != null) {
186             int newSize = template.getVariables().size();
187             logger.info("newSize=" + newSize + ", applyExprs.size()="
188                     + applyExprs.size());
189             if (newSize > applyExprs.size()) {
190                 logger.info("expanding");
191                 for (int i = applyExprs.size(); i < newSize; i++) {
192                     applyExprs.add(null);
193                 }
194             } else {
195                 logger.info("not expanding");
196                 // Working with ArrayList instead of List to make
                        Simple XML
197                 // work
198                 applyExprs = new ArrayList<SingleExpression>(applyExprs
199                         .subList(0, newSize));
200             }
201         } else {
202             applyExprs = new ArrayList<SingleExpression>();
203         }
204         logger.info("After: applyExprs.size()=" + applyExprs.size());
205     }
206 
207     @Override
208     public String toString() {
209         return "Apply Test";
210     }
211 
212     /**
```

```
213      * Changes the original {@link SingleExpression} with the new
214      * {@link SingleExpression} if the original is located in the
           array of
215      * {@link SingleExpression}.
216      *
217      * @param originalExpr
218      * @param newExpr
219      */
220     public void changeExprWithExpr(SingleExpression originalExpr,
221             SingleExpression newExpr) {
222         logger.info("apply exprs has " + applyExprs.size() + " in it
                ");
223         for (int i = 0; i < applyExprs.size(); i++) {
224             if (applyExprs.get(i) instanceof Variable) {
225                 ((Variable)applyExprs.get(i)).getName();
226                 if (((Variable)applyExprs.get(i)).getName().equals(((
                        Variable)originalExpr).getName())) {
227                     applyExprs.set(i, newExpr);
228                     logger.info("Expreesion is now " + applyExprs.get(i)
                            );
229                 }
230             }
231         }
232     }
233
234     @Override
235     public Test clone() {
236         try {
237             Template clonedTemplate = template.clone();
238             ApplyTest clonedTest = new ApplyTest(clonedTemplate,
                    applyExprs);
239             return clonedTest;
240         } catch (Exception e) {
241             logger.severe(e.getMessage());
242             return null;
243         }
244     }
245
246     @Override
247     protected boolean validateSelf(List<ErrorValue> errorList) {
248         return actuallyMapExpressionsToTemplateAndValidate(errorList)
                ;
249     }
250
251     /**
252      * Maps the applyExpressions from this ApplyTest to the
               underlying Template.
253      * Does validation at the same time
254      *
255      * @param errorList
256      * @return
257      */
258     private boolean actuallyMapExpressionsToTemplateAndValidate(
259             List<ErrorValue> errorList) {
260         logger.info("Validate called. Expressions=" + applyExprs);
```

```
261         updateExpressionsFromChild();
262         if (template != null) {
263             if (noNullApplyExpressions()) {
264                 if (template instanceof ReferenceTemplate
265                         && template.getChildren().size() == 0) {
266                     errorList.add(new ErrorValue(
267                             "No template set in ReferenceTemplate",
                                template));
268                 } else if (template.mapApplyToTemplate(applyExprs)) {
269                     return true; // all good
270                 } else
271                     errorList
272                             .add(new ErrorValue(
273                                     "Not all expressions have been mapped in
                                        ApplyTest",
274                                     this));
275             } else {
276                 errorList.add(new ErrorValue(
277                         "Some ApplyExpressions have not been set in
                            ApplyTest",
278                         this));
279             }
280         } else {
281             errorList.add(new ErrorValue(
282                     "Template has not been set in ApplyTest", this));
283         }
284         return false;
285     }
286 
287     @Override
288     public void validateRecursively(List<ErrorValue> errorList) {
289         validateSelf(errorList);
290 
291         if (template != null) {
292             template.validateRecursively(errorList);
293         }
294 
295     }
296 
297     @Override
298     public void setupObserverObservableHierarchy() {
299         if (template != null) {
300             template.addObserver(this);
301         }
302 
303     }
304 
305     @Override
306     public ArrayList<ApplyTest> getDecendentApplyTest() {
307         ArrayList<ApplyTest> applyTests = new ArrayList<ApplyTest>();
308         applyTests.add(this);
309         return applyTests;
310     }
311 
312     @Override
```

```
313     public void interruptFuture() {
314         logger.info("Interrupting future on ApplyTest");
315         if (template != null) {
316             template.interruptFuture();
317         }
318     }
319 
320     @Override
321     public ArrayList<TestResult> getTestResults() {
322         ArrayList<TestResult> results = new ArrayList<TestResult>();
323         if (template != null) {
324             results.addAll(template.getTestResults());
325         }
326         return results;
327     }
328 }
```

test/CollectionTest.java

```
1  package util.nominaltest2.model.test;
2  
3  import java.util.ArrayList;
4  import java.util.Collections;
5  import java.util.List;
6  
7  import org.simpleframework.xml.ElementList;
8  import org.simpleframework.xml.ElementListUnion;
9  import org.simpleframework.xml.Path;
10 
11 import util.nominaltest2.model.TreeItem;
12 import util.nominaltest2.model.errors.ErrorValue;
13 import util.nominaltest2.model.expression.SingleExpression;
14 import util.nominaltest2.model.results.TestResult;
15 import util.nominaltest2.model.results.TestResult.testResultEnum;
16 
17 public abstract class CollectionTest extends Test {
18 
19     protected List<SingleExpression> expressions;
20 
21     @Path("Tests")
22     @ElementListUnion({
23             @ElementList(entry = "ApplyTest", type = ApplyTest.class,
                    inline = true, required = false),
24             @ElementList(entry = "ConcurrentCollectionTest", type =
                    ConcurrentCollectionTest.class, inline = true,
                    required = false),
25             @ElementList(entry = "SequentialCollectionTest", type =
                    SequentialCollectionTest.class, inline = true,
                    required = false),
26             @ElementList(entry = "DelayTest", type = DelayTest.class,
                    inline = true, required = false),
```

```
27          @ElementList(entry = "ForTest", type = ForTest.class,
                inline = true, required = false) })
28      protected ArrayList<Test> tests = new ArrayList<Test>();
29
30      @Override
31      public ArrayList<TestResult> getTestResults() {
32          ArrayList<TestResult> testResults = new ArrayList<TestResult
                >();
33          for (Test t : tests) {
34              testResults.addAll(t.getTestResults());
35          }
36          return testResults;
37      }
38
39
40      @Override
41      public testResultEnum getEnumResult() {
42          ArrayList<testResultEnum> testResults = new ArrayList<
                testResultEnum>();
43          for (Test t : tests) {
44              testResults.add(t.getEnumResult());
45          }
46          if (testResults.size() > 0) {
47              return Collections.max(testResults);
48          }
49          return testResultEnum.NONE;
50
51      }
52
53      /**
54       * Will send the test either by sending a list of {@link
              SingleExpression}
55       * if this is initialised, otherwise with a normal send on the
              given
56       * {@link Test}.
57       *
58       * @param test
59       */
60      protected void sendTest(Test test) {
61          if (!Thread.currentThread().isInterrupted()) {
62              test.send();
63          }
64      }
65
66      @Override
67      public ArrayList<TreeItem> getChildren() {
68          return new ArrayList<TreeItem>(tests);
69      }
70
71      @Override
72      protected void doAddChild(TreeItem item) throws Exception {
73          if (item instanceof Test) {
74              tests.add((Test) item);
75          } else
76
```

```
77          {
78              throw new Exception("Can't add that type to a Collection:
                    " + item);
79          }
80      }
81
82      @Override
83      protected void doDeleteChild(TreeItem child) {
84          tests.remove(child);
85      }
86
87      @Override
88      protected boolean validateSelf(List<ErrorValue> errorList) {
89          return true;
90      }
91
92      @Override
93      public void validateRecursively(List<ErrorValue> errorList) {
94          for (Test t : tests) {
95              t.validateRecursively(errorList);
96          }
97      }
98
99      @Override
100     public void setupObserverObservableHierarchy() {
101         for (Test t : tests) {
102             t.addObserver(this);
103             t.setupObserverObservableHierarchy();
104         }
105
106     }
107
108     @Override
109     public ArrayList<ApplyTest> getDecendentApplyTest() {
110         ArrayList<ApplyTest> applyTests = new ArrayList<ApplyTest>();
111         for (Test test : tests) {
112             applyTests.addAll(test.getDecendentApplyTest());
113         }
114         return applyTests;
115     }
116
117 }
```

test/ConcurrentCollectionTest.java

```
1 package util.nominaltest2.model.test;
2
3 import java.util.List;
4 import java.util.concurrent.ExecutionException;
5 import java.util.concurrent.Future;
6 import java.util.logging.Logger;
```

```
7
8  import util.nominaltest2.model.TestManager;
9
10 public class ConcurrentCollectionTest extends CollectionTest {
11    private final Logger logger = Logger.getLogger(getClass().
          getName());
12
13    private List<Future<Void>> futureResults;
14
15    @Override
16    public Void call() {
17       if (!Thread.currentThread().isInterrupted()) {
18          try {
19             futureResults = TestManager.getExecutor()
20                   .invokeAll(tests);
21
22             for (Future<Void> futList : futureResults) {
23                try {
24
25                   futList.get();
26                   setChanged();
27                   notifyObservers();
28                } catch (ExecutionException e) {
29                   logger.severe(e.getMessage());
30                }
31             }
32          } catch (InterruptedException e1) {
33             logger.info("In ConcurrentCollectionTest :
                   InterruptedException occured");
34             Thread.currentThread().interrupt();
35          }
36       }
37       return null;
38    }
39
40    @Override
41    public void send() {
42       if (!Thread.currentThread().isInterrupted()) {
43          if (!tests.isEmpty()) {
44             for (Test t : tests) {
45                sendTest(t);
46             }
47          }
48          logger.info("Done calling send() on all children.
                Returning");
49       }
50    }
51
52    @Override
53    public String toString() {
54       return "Concurrent Collection";
55    }
56
57    @Override
58    public Test clone() {
```

```
59          ConcurrentCollectionTest clonedSelf = new
                ConcurrentCollectionTest();
60          for (Test test : tests) {
61             try {
62                clonedSelf.addChild(test.clone());
63             } catch (Exception e) {
64                logger.severe(e.getMessage());
65             }
66          }
67          return clonedSelf;
68       }
69
70       @Override
71       public void interruptFuture() {
72          if (futureResults != null) {
73             for (Future<Void> future : futureResults) {
74                future.cancel(true);
75             }
76          }
77          for (Test test : tests) {
78             test.interruptFuture();
79          }
80       }
81
82    }
```

test/DelayTest.java

```
1  package util.nominaltest2.model.test;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.concurrent.Future;
6  import java.util.logging.Logger;
7
8  import org.simpleframework.xml.Attribute;
9  import org.simpleframework.xml.Element;
10 import org.simpleframework.xml.ElementUnion;
11
12 import util.nominaltest2.model.TestManager;
13 import util.nominaltest2.model.TreeItem;
14 import util.nominaltest2.model.errors.ErrorValue;
15 import util.nominaltest2.model.results.TestResult;
16 import util.nominaltest2.model.results.TestResult.testResultEnum;
17
18 public class DelayTest extends Test {
19    private final Logger logger = Logger.getLogger(getClass().
          getName());
20
21    @ElementUnion({
```

```
22          @Element(name = "ApplyTest", type = ApplyTest.class,
                required = false),
23          @Element(name = "ConcurrentCollectionTest", type =
                ConcurrentCollectionTest.class, required = false),
24          @Element(name = "SequentialCollectionTest", type =
                SequentialCollectionTest.class, required = false),
25          @Element(name = "DelayTest", type = DelayTest.class,
                required = false),
26          @Element(name = "ForTest", type = ForTest.class, required
                = false) })
27      private Test test;
28
29      @Attribute
30      private int wait = 0;
31
32      @Override
33      public void send() {
34          if (validateSelf(new ArrayList<ErrorValue>())) {
35              if (wait > 0) {
36                  try {
37                      Thread.sleep(wait);
38                  } catch (InterruptedException e) {
39                      logger.info("DelayTest got interrupted, cancels
                            execution");
40                      Thread.currentThread().interrupt();
41                      return;
42                  }
43              }
44              test.send();
45          } else {
46              logger.severe("Couldn't validate before sending!");
47          }
48      }
49
50      public void setWait(int wait) {
51          this.wait = wait;
52      }
53
54      public int getWait() {
55          return wait;
56      }
57
58      @Override
59      protected void doAddChild(TreeItem item) throws Exception {
60          if (test == null && item instanceof Test) {
61              test = (Test) item;
62          } else {
63              throw new Exception("Can't add an extra test or of type "
                    + item
64                      + " to a DelayTest");
65          }
66      }
67
68      @Override
69      protected void doDeleteChild(TreeItem child) throws Exception {
```

```
70          if (child == test) {
71              test = null;
72          } else {
73              throw new Exception("Can't delete " + child
74                      + " from DelayTest as it is not placed here.");
75          }
76      }
77
78      @Override
79      public ArrayList<TreeItem> getChildren() {
80          ArrayList<TreeItem> children = new ArrayList<TreeItem>();
81          if (test != null) {
82              children.add(test);
83          }
84          return children;
85      }
86
87      @Override
88      public testResultEnum getEnumResult() {
89          if (test != null) {
90              return test.getEnumResult();
91          } else {
92              return testResultEnum.NONE;
93          }
94      }
95
96      @Override
97      public Void call() throws Exception {
98          Future<Void> futureRes;
99          if (test != null) {
100             futureRes = TestManager.getExecutor().submit(test);
101             try {
102                 futureRes.get();
103             } catch (Exception e) {
104                 logger.severe(e.getMessage());
105             }
106         }
107         return null;
108     }
109
110     @Override
111     public String toString() {
112         return "Delay Test";
113     }
114
115     @Override
116     public Test clone() {
117         DelayTest clonedSelf = new DelayTest();
118         if (test != null) {
119             Test clonedTest = test.clone();
120             try {
121                 clonedSelf.addChild(clonedTest);
122             } catch (Exception e) {
123                 logger.severe(e.getMessage());
124             }
```

```
125         }
126         clonedSelf.setWait(wait);
127         return clonedSelf;
128     }
129 
130     @Override
131     protected boolean validateSelf(List<ErrorValue> errorList) {
132         if (test != null) {
133             return true;
134         } else {
135             errorList.add(new ErrorValue("No test has been set in
                    DelayTest",
136                     this));
137         }
138         return false;
139     }
140 
141     @Override
142     public void validateRecursively(List<ErrorValue> errorList) {
143         if (validateSelf(errorList))
144             test.validateRecursively(errorList);
145     }
146 
147     @Override
148     public void setupObserverObservableHierarchy() {
149         if (test != null) {
150             test.addObserver(this);
151             test.setupObserverObservableHierarchy();
152         }
153     }
154 
155     @Override
156     public ArrayList<ApplyTest> getDecendentApplyTest() {
157         ArrayList<ApplyTest> applyTests = new ArrayList<ApplyTest>();
158         applyTests.addAll(test.getDecendentApplyTest());
159         return applyTests;
160     }
161 
162     @Override
163     public void interruptFuture() {
164         if (test != null) {
165             test.interruptFuture();
166         }
167     }
168 
169     @Override
170     public ArrayList<TestResult> getTestResults() {
171         ArrayList<TestResult> results = new ArrayList<TestResult>();
172         if (test != null) {
173             results.addAll(test.getTestResults());
174         }
175 
176         return results;
177     }
178 
```

```
179 }
```

test/ForTest.java

```
1  package util.nominaltest2.model.test;
2
3  import java.util.ArrayList;
4  import java.util.Collections;
5  import java.util.List;
6  import java.util.logging.Logger;
7
8  import org.simpleframework.xml.Attribute;
9  import org.simpleframework.xml.Element;
10 import org.simpleframework.xml.ElementUnion;
11 import org.simpleframework.xml.Path;
12
13 import util.nominaltest2.model.TreeItem;
14 import util.nominaltest2.model.errors.ErrorValue;
15 import util.nominaltest2.model.expression.SingleExpression;
16 import util.nominaltest2.model.expression.singleExpressions.
       ArgumentSingleExpression;
17 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
18 import util.nominaltest2.model.expression.singleExpressions.
       argument.EnumArgumentSingleExpression;
19 import util.nominaltest2.model.expression.singleExpressions.
       argument.IntArgumentSingleExpression;
20 import util.nominaltest2.model.results.TestResult;
21 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.MinusArithmeticSingleExpression;
22 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.MultiplicationArithmeticSingleExpression;
23 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.PlusArithmeticSingleExpression;
24 import util.nominaltest2.model.results.TestResult.testResultEnum;
25
26 public class ForTest extends Test {
27     private final Logger logger = Logger.getLogger(getClass().
          getName());
28
29     @Path("StartValue")
30     @ElementUnion( {
31             @Element(name = "EnumArgumentExpression", type =
                    EnumArgumentSingleExpression.class, required = false),
32             @Element(name = "IntArgumentExpression", type =
                    IntArgumentSingleExpression.class, required = false),
33             @Element(name = "Variable", type = Variable.class,
                    required = false),
34             @Element(name = "MinusArithmeticSingleExpression", type =
                    MinusArithmeticSingleExpression.class, required =
                    false),
```

```
35          @Element(name = "MultiplicationArithmeticSingleExprssion",
                type = MultiplicationArithmeticSingleExpression.class
                , required = false),
36          @Element(name = "PlusArithmeticSingleExpression", type =
                PlusArithmeticSingleExpression.class, required = false
                ) })
37      SingleExpression startValue;
38
39      @Path("ToValue")
40      @ElementUnion( {
41          @Element(name = "EnumArgumentExpression", type =
                EnumArgumentSingleExpression.class, required = false),
42          @Element(name = "IntArgumentExpression", type =
                IntArgumentSingleExpression.class, required = false),
43          @Element(name = "Variable", type = Variable.class,
                required = false),
44          @Element(name = "MinusArithmeticSingleExpression", type =
                MinusArithmeticSingleExpression.class, required =
                false),
45          @Element(name = "MultiplicationArithmeticSingleExprssion",
                type = MultiplicationArithmeticSingleExpression.class
                , required = false),
46          @Element(name = "PlusArithmeticSingleExpression", type =
                PlusArithmeticSingleExpression.class, required = false
                ) })
47      SingleExpression toValue;
48
49      @Path("StepValue")
50      @ElementUnion( {
51          @Element(name = "EnumArgumentExpression", type =
                EnumArgumentSingleExpression.class, required = false),
52          @Element(name = "IntArgumentExpression", type =
                IntArgumentSingleExpression.class, required = false),
53          @Element(name = "Variable", type = Variable.class,
                required = false),
54          @Element(name = "MinusArithmeticSingleExpression", type =
                MinusArithmeticSingleExpression.class, required =
                false),
55          @Element(name = "MultiplicationArithmeticSingleExprssion",
                type = MultiplicationArithmeticSingleExpression.class
                , required = false),
56          @Element(name = "PlusArithmeticSingleExpression", type =
                PlusArithmeticSingleExpression.class, required = false
                ) })
57      SingleExpression stepValue;
58      long startValLong, toValLong, stepValLong;
59
60      @Attribute
61      boolean upwards = true;
62
63      @Element
64      ApplyTest applyTest;
65      ApplyTest currentClonedApplyTest;
66
```

```
67      ArrayList<ApplyTest> resultApplyTests = new ArrayList<ApplyTest
            >();
68
69      @Element
70      Variable variable = new Variable("For Loop Parameter");
71
72      @Override
73      public Test clone() {
74          ForTest clonedSelf = new ForTest();
75          if (startValue != null) {
76              clonedSelf.setStartValue((SingleExpression) startValue.
                    clone());
77          }
78          if (toValue != null) {
79              clonedSelf.setToValue((SingleExpression) toValue.clone());
80          }
81          if (stepValue != null) {
82              clonedSelf.setStepValue((SingleExpression) stepValue.clone
                    ());
83          }
84          clonedSelf.setStepDirection(getStepDirection());
85          try {
86              clonedSelf.addChild(applyTest);
87          } catch (Exception e) {
88              logger.severe(e.getMessage());
89          }
90
91          return clonedSelf;
92      }
93
94      @Override
95      public ArrayList<ApplyTest> getDecendentApplyTest() {
96          ArrayList<ApplyTest> applyTests = new ArrayList<ApplyTest>();
97          if (applyTest != null) {
98              applyTests.add(applyTest);
99          }
100         return applyTests;
101     }
102
103     @Override
104     public void send() {
105         logger.info("Starting for-test");
106         if (validateSelf(new ArrayList<ErrorValue>())) {
107             resultApplyTests = new ArrayList<ApplyTest>();
108             setChanged();
109             notifyObservers();
110             if (checkIfValid() && !Thread.currentThread().
                    isInterrupted()) {
111                 currentClonedApplyTest = cloneApplyTestWithExpr(
                        startValue);
112                 currentClonedApplyTest.send();
113             }
114         } else {
115             logger.severe("Couldn't validate before sending!");
116         }
```

```
117     }
118
119     @Override
120     public Void call() throws Exception {
121        if (checkIfValid() && !Thread.currentThread().isInterrupted()
               ) {
122           if (currentClonedApplyTest != null) {
123              currentClonedApplyTest.call();
124
125              if (upwards) {
126                 for (long i = startValLong
127                       + stepValLong; i <= toValLong; i +=
                             stepValLong) {
128                    sendCloneAndCall(new IntArgumentSingleExpression(
                          i));
129                 }
130              } else {
131                 for (long i = startValLong
132                       - stepValLong; i >= toValLong; i -=
                             stepValLong) {
133                    sendCloneAndCall(new IntArgumentSingleExpression(
                          i));
134                 }
135              }
136           }
137        }
138
139        return null;
140     }
141
142     /**
143      * Clone the {@link ApplyTest} assigned, aswell as sending and
            receiving
144      * results for the cloned {@link ApplyTest}. All in all it will
            do a
145      * complete run.
146      *
147      * @param expr
148      *            the instantiation of the Variable found on the {
            @link ForTest}
149      * @throws Exception
150      */
151     private void sendCloneAndCall(SingleExpression expr) throws
           Exception {
152        if (!Thread.currentThread().isInterrupted()) {
153           currentClonedApplyTest = cloneApplyTestWithExpr(expr);
154           currentClonedApplyTest.send();
155           currentClonedApplyTest.call();
156        }
157     }
158
159     /**
160      * Clone the {@link ApplyTest} associated with the {@link
            ForTest}.
161      *
```

```
162      * @param expr
163      *            the expression to replace the {@link ForTest}
           variable, to
164      *            ensure that it is updated in the cloned {@link
           Test}.
165      * @return the cloned {@link ApplyTest}
166      */
167     private ApplyTest cloneApplyTestWithExpr(SingleExpression expr)
          {
168         ApplyTest clonedApplyTest = (ApplyTest) applyTest.clone();
169         clonedApplyTest.changeExprWithExpr(variable, expr);
170         resultApplyTests.add(clonedApplyTest);
171         clonedApplyTest.addObserver(this);
172         setChanged();
173         notifyObservers();
174         return clonedApplyTest;
175     }
176 
177     /**
178      * Used to check that all values are set, and we don't cause an
           infinite
179      * loop to happen.
180      *
181      * @return <b>True</b> if everything is okay, and the {@link
           ForTest} is
182      *         ready to be executed.
183      */
184     private boolean checkIfValid() {
185         if (startValue != null
186                 && startValue.getValue() instanceof
                      IntArgumentSingleExpression
187                 && toValue != null
188                 && toValue.getValue() instanceof
                      IntArgumentSingleExpression
189                 && stepValue != null
190                 && stepValue.getValue() instanceof
                      IntArgumentSingleExpression) {
191             ArgumentSingleExpression startVal = (
                    ArgumentSingleExpression) startValue
192                     .getValue();
193             startValLong = startVal.getLongValue();
194             ArgumentSingleExpression toVal = (ArgumentSingleExpression
                    ) toValue.getValue();
195             toValLong=toVal.getLongValue();
196             ArgumentSingleExpression stepVal = (
                    ArgumentSingleExpression) stepValue.getValue();
197             stepValLong=stepVal.getLongValue();
198 
199             if (upwards && startVal.getLongValue() > toVal.
                    getLongValue()
200                     && stepVal.getLongValue() >= 0) {
201                 return false;
202             } else if (upwards
203                     && startVal.getLongValue() < toVal.getLongValue()
204                     && stepVal.getLongValue() <= 0) {
```

```
205             return false;
206         } else {
207             return true;
208         }
209     }
210     return false;
211 }
212
213 @Override
214 protected boolean validateSelf(List<ErrorValue> errorList) {
215     if (applyTest != null) {
216         if (startValue == null || !startValue.isBounded()) {
217
218             errorList.add(new ErrorValue("Start value not set on
                    ForTest ",
219                     this));
220             return false;
221         }
222         if (toValue == null || !toValue.isBounded()) {
223             errorList.add(new ErrorValue(
224                     "The end value not set on ForTest ", this));
225             return false;
226         }
227         if (stepValue == null || !stepValue.isBounded()) {
228             errorList.add(new ErrorValue("Step value not set on
                    ForTest ",
229                     this));
230             return false;
231         }
232
233         if (stepValue.getValue() instanceof
                IntArgumentSingleExpression) {
234             if (((IntArgumentSingleExpression) stepValue.getValue()
                        )
235                     .getLongValue() == 0) {
236                 errorList.add(new ErrorValue(
237                         "The step cannot be set to be 0", this));
238             }
239         } else if (!checkIfValid()) {
240             errorList
241                     .add(new ErrorValue(
242                             "The input values causes an infinite loop
                                    in a ForTest",
243                             this));
244         }
245     } else {
246         errorList.add(new ErrorValue(
247                 "ApplyTest has not been set in ForTest", this));
248     }
249     return errorList.size() == 0;
250 }
251
252 @Override
253 public void validateRecursively(List<ErrorValue> errorList) {
254     logger.info("Validate called. ForTest =" + this);
```

```
255         if (validateSelf(errorList)) {
256             applyTest.validateRecursively(errorList);
257 
258         }
259     }
260 
261     @Override
262     protected void doAddChild(TreeItem item) throws Exception {
263         if (applyTest == null && item instanceof ApplyTest) {
264             applyTest = (ApplyTest) item;
265         } else {
266             throw new Exception("Can't add that type " + item
267                     + " or more children to the given for-test");
268         }
269 
270     }
271 
272     @Override
273     protected void doDeleteChild(TreeItem child) throws Exception {
274         if (applyTest == child) {
275             applyTest = null;
276         } else {
277             throw new Exception("Can't delete " + child
278                     + " from the given for-test as it is not contained
                            here");
279         }
280     }
281 
282     @Override
283     public ArrayList<TreeItem> getChildren() {
284         ArrayList<TreeItem> children = new ArrayList<TreeItem>();
285         if (applyTest != null) {
286             children.add(applyTest);
287         }
288         return children;
289     }
290 
291     /**
292      * Return the cloned {@link ApplyTest}s generated during a run.
293      *
294      * @return a list containing all the {@link ApplyTest} generated
                during a
295      *          run.
296      */
297     public ArrayList<TreeItem> getResultApplyTests() {
298         if(resultApplyTests==null||resultApplyTests.size()==0)
299         {
300             return getChildren();
301         }
302         ArrayList<TreeItem> items = new ArrayList<TreeItem>();
303         items.addAll(resultApplyTests);
304         return items;
305     }
306 
307     @Override
```

```
308     public testResultEnum getEnumResult() {
309        if (resultApplyTests.size() == 0) {
310           return testResultEnum.NONE;
311        } else {
312           ArrayList<testResultEnum> results = new ArrayList<
                testResultEnum>();
313           for (ApplyTest applyTest : resultApplyTests) {
314              results.add(applyTest.getEnumResult());
315           }
316           return Collections.max(results);
317        }
318     }
319
320     public SingleExpression getStartValue() {
321        return startValue;
322     }
323
324     public void setStartValue(SingleExpression startValue) {
325        this.startValue = startValue;
326     }
327
328     public SingleExpression getToValue() {
329        return toValue;
330     }
331
332     public void setToValue(SingleExpression toValue) {
333        this.toValue = toValue;
334     }
335
336     public SingleExpression getStepValue() {
337        return stepValue;
338     }
339
340     public void setStepValue(SingleExpression stepValue) {
341        this.stepValue = stepValue;
342     }
343
344     public Variable getForLoopVariable() {
345        return variable;
346     }
347
348     @Override
349     public void setupObserverObservableHierarchy() {
350
351     }
352
353     /**
354      * Gets the step direction
355      *
356      * @return <b>true</b> mean we will go up to the declared end-
                value
357      */
358     public boolean getStepDirection() {
359        return upwards;
360     }
```

```
361
362     /**
363      * Set the step direction.
364      *
365      * @param stepDirection
366      *            <b>true</b> mean we will go up to the declared end
           -value
367      */
368     public void setStepDirection(boolean stepDirection) {
369         this.upwards = stepDirection;
370     }
371
372     @Override
373     public String toString() {
374         return "For Test";
375     }
376
377     @Override
378     public void interruptFuture() {
379         if (currentClonedApplyTest != null) {
380             currentClonedApplyTest.interruptFuture();
381         }
382     }
383
384     @Override
385     public ArrayList<TestResult> getTestResults() {
386         ArrayList<TestResult> results = new ArrayList<TestResult>();
387         for (ApplyTest at : resultApplyTests) {
388             results.addAll(at.getTestResults());
389         }
390         return results;
391     }
392
393 }
```

test/RootTest.java

```
1 package util.nominaltest2.model.test;
2
3 public class RootTest extends SequentialCollectionTest {
4     @Override
5     public String toString() {
6         return "Root Test";
7     }
8
9 }
```

test/SequentialCollectionTest.java

```
1  package util.nominaltest2.model.test;
2
3  import java.util.concurrent.ExecutionException;
4  import java.util.List;
5  import java.util.concurrent.Future;
6  import java.util.logging.Logger;
7
8  import util.nominaltest2.model.TestManager;
9  import util.nominaltest2.model.errors.ErrorValue;
10
11 public class SequentialCollectionTest extends CollectionTest {
12    private final Logger logger = Logger.getLogger(getClass().
          getName());
13
14    Future<Void> futureRes;
15
16    @Override
17    public void send() {
18       if (!Thread.currentThread().isInterrupted()) {
19          logger.info("Sending first test");
20          if (tests.size() > 0) {
21             sendTest(tests.get(0));
22          } else {
23             logger.info("No tests to send");
24          }
25       }
26    }
27
28    @Override
29    public Void call() {
30       if (tests.size() > 0) {
31          if (!Thread.currentThread().isInterrupted()) {
32             futureRes = TestManager.getExecutor().submit(tests.get
                   (0));
33             try {
34                futureRes.get();
35             } catch (InterruptedException e) {
36                logger.info("Sequential experienced interrupt");
37                Thread.currentThread().interrupt();
38             } catch (ExecutionException e) {
39                logger.severe(e.getMessage());
40             }
41          }
42
43          for (int i = 1; i < tests.size(); i++) {
44             if (!Thread.currentThread().isInterrupted()) {
45                Test t = tests.get(i);
46                sendTest(t);
47                futureRes = TestManager.getExecutor().submit(t);
48                try {
49                   futureRes.get();
50                } catch (InterruptedException e) {
```

```
51                     logger.info("Sequential experienced interrupt");
52                     Thread.currentThread().interrupt();
53                 } catch (ExecutionException e) {
54                     logger.severe(e.getMessage());
55                 }
56             }
57         }
58 
59     }
60     return null;
61 }
62 
63 @Override
64 public String toString() {
65     return "Sequential Collection";
66 }
67 
68 @Override
69 public Test clone() {
70     SequentialCollectionTest clonedSelf = new
           SequentialCollectionTest();
71     for (Test test : tests) {
72         try {
73             clonedSelf.addChild(test.clone());
74         } catch (Exception e) {
75             logger.severe(e.getMessage());
76         }
77     }
78 
79     return clonedSelf;
80 }
81 
82 @Override
83 public void interruptFuture() {
84     if (futureRes != null) {
85         futureRes.cancel(true);
86     }
87     for (Test test : tests) {
88         test.interruptFuture();
89     }
90 }
91 }
```

test/Test.java

```
1 package util.nominaltest2.model.test;
2 
3 import java.util.ArrayList;
4 import java.util.List;
5 import java.util.concurrent.Callable;
6 
```

```
7  import util.nominaltest2.model.TreeItem;
8  import util.nominaltest2.model.errors.ErrorValue;
9  import util.nominaltest2.model.results.TestResult;
10
11 public abstract class Test extends TreeItem implements Callable<
       Void> {
12
13     /**
14      * Will send any test or template currently under the given test
15      */
16     public abstract void send();
17
18     /**
19      * Will clone the test, giving rise to a copy of the given test.
            Is used to
20      * ensure we wont correct any information in the original test,
            thereby
21      * allowing multiple use of it at the same time.
22      *
23      * @return {@link Test} a clone of the original test.
24      */
25     @Override
26     public abstract Test clone();
27
28     /**
29      * Will retrieve all apply test under the given test. If the
            given test is
30      * an {@link ApplyTest} it will return itself.
31      *
32      * This can be used to check whether a parameter is being used
            further down
33      * the tree.
34      *
35      * @return a list containing all descendant {@link ApplyTest}
36      */
37     public abstract ArrayList<ApplyTest> getDecendentApplyTest();
38
39     /**
40      * Will validate the test, ensuring everything is set correctly.
            Will be
41      * checked before runtime of the tests.
42      *
43      * @param errorList
44      *            - a list that contain all errors.
45      */
46     protected abstract boolean validateSelf(List<ErrorValue>
           errorList);
47
48     /**
49      * Will validate the test, ensuring everything is set correctly.
            Will be
50      * used before any execution is done or even considered starting
            .
51      *
52      * @param errorList
```

```
53      *            - a list that contain all errors.
54      */
55     public abstract void validateRecursively(List<ErrorValue>
           errorList);
56
57     /**
58      * Sets up the Observer-Observable Hierarchy in order for the
           GUI to
59      * function as intended.
60      */
61     public abstract void setupObserverObservableHierarchy();
62
63     /**
64      * Will interrupt any FutureTask currently running on the test,
           or
65      * FutureTask on the children of the currently Task.
66      */
67     public abstract void interruptFuture();
68
69
70     public abstract ArrayList<TestResult> getTestResults();
71 }
```

model/validators/ExpressionValidator.java

```
1  package util.nominaltest2.model.validators;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.logging.Logger;
6
7  import org.simpleframework.xml.Attribute;
8  import org.simpleframework.xml.ElementUnion;
9  import org.simpleframework.xml.Element;
10
11 import util.nominaltest2.model.errors.ErrorValue;
12 import util.nominaltest2.model.expression.Expression;
13 import util.nominaltest2.model.expression.SingleExpression;
14 import util.nominaltest2.model.expression.intervalExpression.
       IntIntervalExpression;
15 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
16 import util.nominaltest2.model.expression.singleExpressions.
       argument.EnumArgumentSingleExpression;
17 import util.nominaltest2.model.expression.singleExpressions.
       argument.IntArgumentSingleExpression;
18 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.MinusArithmeticSingleExpression;
19 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.MultiplicationArithmeticSingleExpression;
```

```
20 import util.nominaltest2.model.expression.singleExpressions.
       arithmetic.PlusArithmeticSingleExpression;
21 import util.nominaltest2.model.template.BasicTemplate;
22
23 import common.dtusat2.packet.Packet;
24 import common.types.GenType;
25 import common.types.TypeMaps;
26 import common.types.argument.Argument;
27 import common.types.argument.EnumArgument;
28 import common.types.argument.IntArgument;
29 import common.types.confirmation.ConfirmationType;
30 import common.types.parameter.DoubleParameter;
31 import common.types.parameter.EnumParameter;
32 import common.types.parameter.IntParameter;
33 import common.types.parameter.NamedValue;
34 import common.types.parameter.Parameter;
35 import common.types.parameter.ParameterType;
36 import common.types.parameter.StringParameter;
37 import common.types.teledata.TeledataType;
38
39 /**
40  * Is used to validate the {@link Argument}s that a {@link Packet}
        contain. This
41  * will be done with the use of {@link Expression}s.
42  *
43  */
44 public class ExpressionValidator {
45     private final Logger logger = Logger.getLogger(getClass().
          getName());
46
47     @Attribute
48     protected String paramName;
49
50     @ElementUnion( {
51           @Element(name = "EnumArgumentExpression", type =
                EnumArgumentSingleExpression.class, required = false),
52           @Element(name = "IntArgumentExpression", type =
                IntArgumentSingleExpression.class, required = false),
53           @Element(name = "Variable", type = Variable.class,
                required = false),
54           @Element(name = "IntIntervalExpression", type =
                IntIntervalExpression.class, required = false),
55           @Element(name = "MinusArithmeticSingleExpression", type =
                MinusArithmeticSingleExpression.class, required =
                false),
56           @Element(name = "MultiplicationArithmeticSingleExprssion",
                 type = MultiplicationArithmeticSingleExpression.class
                , required = false),
57           @Element(name = "PlusArithmeticSingleExpression", type =
                PlusArithmeticSingleExpression.class, required = false
                ) })
58     private Expression expression;
59
60     @Attribute(name = "validatorType")
61     private ExpressionValidatorType expressionValidatorType;
```

```
62
63      public ExpressionValidator(@Attribute(name = "paramName") String
            paramName) {
64          this.paramName = paramName;
65      }
66
67      public ExpressionValidator(String paramName, Expression
            expression) {
68          this.paramName = paramName;
69          this.expression = expression;
70          getExpressionValidatorTypeFromParamName(paramName);
71      }
72
73      public void setExpression(Expression expression) {
74          this.expression = expression;
75      }
76
77      public Expression getExpression() {
78          return expression;
79      }
80
81      public void setExpressionValidatorType(
82              ExpressionValidatorType expressionValidatorType) {
83          this.expressionValidatorType = expressionValidatorType;
84      }
85
86      public ExpressionValidatorType getExpressionValidatorType() {
87          return expressionValidatorType;
88      }
89
90      /**
91       * Define the different types an ExpressionValidator can take. <
              br/>
92       * <center>As of now we have <b>INT, INTERVAL, DOUBLE, ENUM and
93       * STRING</b>.</center> <br/>
94       * More will have to be added to support more expressions.
95       *
96       */
97      public enum ExpressionValidatorType {
98          INT, INTERVAL, DOUBLE, ENUM, STRING
99      }
100
101     /**
102      * Set the type of the validator corresponding to an
103      * {@link ExpressionValidatorType}.
104      *
105      * @param paramName
106      *              the name for the parameter
107      */
108     private void getExpressionValidatorTypeFromParamName(String
            paramName) {
109         List<ConfirmationType> confTypes = TypeMaps.
                getConfirmationTypeMap()
110                 .getTypes();
111         Parameter param = getMatchingParameter(paramName,
```

```
112             new ArrayList<GenType>(confTypes));
113         if (param == null) {
114             List<TeledataType> teleDataTypes = TypeMaps.
                    getTeledataTypeMap()
115                     .getTypes();
116             param = getMatchingParameter(paramName, new ArrayList<
                    GenType>(
117                     teleDataTypes));
118         }
119 
120         if (param != null) {
121             ParameterType parType = param.getParameterType();
122             if (parType instanceof IntParameter) {
123                 expressionValidatorType = ExpressionValidatorType.INT;
124             } else if (parType instanceof DoubleParameter) {
125                 expressionValidatorType = ExpressionValidatorType.
                        DOUBLE;
126             } else if (parType instanceof EnumParameter) {
127                 expressionValidatorType = ExpressionValidatorType.ENUM;
128             } else if (parType instanceof StringParameter) {
129                 expressionValidatorType = ExpressionValidatorType.
                        STRING;
130             }
131         }
132     }
133 
134     /**
135      * Will locate the given {@link Parameter} in the list of {@link
                GenType}s
136      * given using its name.
137      *
138      * @param paramName
139      *             the name of the parameter to test upon.
140      * @param types
141      *             a list containing all
142      * @return the Parameter to be used.
143      */
144     private Parameter getMatchingParameter(String paramName, List<
            GenType> types) {
145         for (GenType confType : types) {
146             ArrayList<Parameter> params = confType.getParameters();
147             if (params != null) {
148                 for (Parameter par : params) {
149                     if (par.getName().equals(paramName)) {
150                         return par;
151                     }
152                 }
153             }
154         }
155         return null;
156     }
157 
158     public String getParamName() {
159         return paramName;
160     }
```

```
161
162     /**
163      * Will validate if the received Parameter and Argument
           corresponds to what
164      * is expected by the Expression found on the
           ExpressionValidator
165      *
166      * @param receivedParameter
167      * @param recvdArg
168      * @return <b>True</b> in case the Argument was as expected.
169      * @throws Exception
170      */
171     public boolean validate(Parameter receivedParameter, Argument
         recvdArg)
172             throws Exception {
173         if (expression != null) {
174             // Expression expectedValue = expression.getValue();
175             Argument receivedArgument = recvdArg;
176             if (expression.isBounded()) {
177                 // Must convert the Enum to an Int, as we currently
                        uses an int
178                 // for the Enums.
179                 if (expression instanceof EnumArgumentSingleExpression)
                        {
180                     if (receivedParameter.getParameterType() instanceof
                            EnumParameter
181                             && recvdArg instanceof EnumArgument) {
182                         receivedArgument = convertEnumArgument(
183                                 (EnumParameter) receivedParameter
184                                         .getParameterType(),
185                                 (EnumArgument) recvdArg);
186                     }
187                 }
188
189                 return expression.evaluateAgainstArgument(
                        receivedArgument);
190             }
191             throw new Exception(
192                     "Error in ExpressionValidator validat: expectedValue
                            is not bound");
193         }
194         throw new Exception(
195                 "Error in ExpressionValidator validate: expression is
                        not set.");
196     }
197
198     /**
199      * Will convert a received {@link EnumArgument} to an {@link
           IntArgument},
200      * as we are using the numeric values of {@link EnumArgument}s
           to validate
201      * upon.
202      *
203      * @param receivedParameter
```

```
204      *              the {@link Parameter} which the {@link
             EnumArgument} is held
205      *              in.
206      * @param recvdArg
207      *              the {@link EnumArgument} that needs to be
             validated.
208      * @return the convert {@link EnumArgument}
209      */
210     private IntArgument convertEnumArgument(EnumParameter
         receivedParameter,
211             EnumArgument recvdArg) {
212         ArrayList<NamedValue> namedValues = receivedParameter.
             getValues();
213 
214         for (NamedValue namedValue : namedValues) {
215             if (recvdArg.getValue().equals(namedValue.getName())) {
216                 try {
217                     return new IntArgument(namedValue.value());
218                 } catch (Exception e) {
219                     logger.severe(e.getMessage());
220                     return null;
221                 }
222 
223             }
224         }
225 
226         return null;
227     }
228 
229     @Override
230     public ExpressionValidator clone() {
231         if (expression != null) {
232             return new ExpressionValidator(paramName, expression.clone
                 ());
233         } else {
234             return new ExpressionValidator(paramName);
235         }
236     }
237 
238     /**
239      * Update the expression according to the given {@link Variable}
             s. In case
240      * there is a match on the names, the expression will be
             replaced with the
241      * new {@link Variable}.
242      *
243      * @param variables
244      */
245     public void updateExpression(ArrayList<Variable> variables) {
246         if (expression != null && expression instanceof Variable) {
247             for (SingleExpression expr : variables) {
248                 if (expr instanceof Variable) {
249                     Variable variable = (Variable) expr;
250                     if (((Variable) expression).getName().equals(
251                             variable.getName())) {
```

```
252                     expression = expr;
253                 }
254             }
255         }
256     }
257 }
258
259 /**
260  * Check if all details is filled out on the given
261  * {@link ExpressionValidator}.
262  *
263  * @param errorList
264  *             a list containing all errors discovered so far.
265  * @param basicTemplate
266  *             the {@link BasicTemplate} that the
         ExpressionValidator is
267  *             contained under.
268  */
269 public void validate(List<ErrorValue> errorList, BasicTemplate
        basicTemplate) {
270     if (paramName == null || paramName.length() == 0) {
271         errorList.add(new ErrorValue(
272                 "paramName is not set in ExpressionValidator",
273                 basicTemplate));
274     }
275     if (expression == null) {
276         errorList
277                 .add(new ErrorValue(
278                         "Expression not set in ExpressionValidator",
279                         basicTemplate));
280     }
281     if (expressionValidatorType == null) {
282         errorList
283                 .add(new ErrorValue(
284                         "ExpressionValidatorType is not set in
                            ExpressionValidator",
285                         basicTemplate));
286     }
287 }
288
289 /**
290  * Check is the given variable is being used
291  *
292  * @param variable
293  * @return <b>True</b> if the given {@link Variable} is being
         used.
294  */
295 public boolean isVariableInUse(Variable variable) {
296     if (expression == variable) {
297         return true;
298     }
299     return false;
300 }
301
302 @Override
```

```
303     public String toString() {
304         return "ExpressionValidator: " + paramName;
305     }
306 }
```

model/validators/PacketValidator.java

```
1  package util.nominaltest2.model.validators;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.logging.Logger;
6
7  import org.simpleframework.xml.Attribute;
8  import org.simpleframework.xml.Element;
9  import org.simpleframework.xml.ElementList;
10 import org.simpleframework.xml.Path;
11
12 import util.nominaltest2.model.TestManager;
13 import util.nominaltest2.model.TreeItem;
14 import util.nominaltest2.model.errors.ErrorValue;
15 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
16 import util.nominaltest2.model.guard.Guard;
17 import util.nominaltest2.model.guard.RootGuard;
18 import util.nominaltest2.model.results.PacketResult;
19 import util.nominaltest2.model.results.ValidatorResult;
20 import util.nominaltest2.model.results.TestResult.testResultEnum;
21 import util.nominaltest2.model.template.BasicTemplate;
22 import util.nominaltest2.model.template.DTUSatParameterArgument;
23 import common.dtusat2.packet.ConfirmationPacket;
24 import common.dtusat2.packet.Packet;
25 import common.dtusat2.packet.TeledataPacket;
26 import common.dtusat2.packet.Packet.PacketType;
27 import common.types.TypeMaps;
28 import common.types.argument.Argument;
29 import common.types.parameter.Parameter;
30
31 /**
32  * Is used to validate upon a {@link Packet} that is received. This
        is done by
33  * containing multiple {@link ExpressionValidator} that will
       perform the
34  * actually validation upon {@link Argument}s of the Packet. It
       will beside this
35  * have numerous PacketValidators under it, which will be called in
        case this
36  * succeeds.
37  *
38  */
39 public class PacketValidator extends TreeItem {
```

```
40      private final Logger logger = Logger.getLogger(getClass().
            getName());
41
42      @Attribute(name = "typeName", required = false)
43      String typeName;
44
45      @Attribute(name = "packetType", required = false)
46      PacketType packetType;
47
48      Argument[] arguments = {};
49      ArrayList<Parameter> parameters = new ArrayList<Parameter>();
50
51      private PacketResult packetResult;
52
53      @Element
54      RootGuard rootGuard;
55
56      @Path("ExpressionValidators")
57      @ElementList(name = "ExpressionValidators", inline = true,
            required = false)
58      ArrayList<ExpressionValidator> exprValidators = new ArrayList<
            ExpressionValidator>();
59
60      @ElementList(required = false, inline = false)
61      ArrayList<PacketValidator> children = new ArrayList<
            PacketValidator>();
62
63      public PacketValidator() {
64         setRootGuard(new RootGuard());
65      }
66
67      public PacketValidator(@Attribute(name = "typeName") String
            typeName) {
68         this.typeName = typeName;
69         setPacketType(typeName);
70         setRootGuard(new RootGuard());
71      }
72
73      public void setRootGuard(RootGuard rootGuard) {
74         this.rootGuard = rootGuard;
75         rootGuard.addObserver(this);
76         setChanged();
77         notifyObservers();
78      }
79
80      public RootGuard getRootGuard() {
81         return rootGuard;
82      }
83
84      public void setTypeName(String typeName) {
85         this.typeName = typeName;
86         setPacketType(typeName);
87         setChanged();
88         notifyObservers();
89      }
```

```
90
91      public String getTypeName() {
92          return typeName;
93      }
94
95      public PacketType getPacketType() {
96          return packetType;
97      }
98
99      public void addExpressionValidator(ExpressionValidator v) {
100         exprValidators.add(v);
101         setChanged();
102         notifyObservers();
103     }
104
105     public void deleteExpressionValidator(ExpressionValidator v) {
106         if (exprValidators.contains(v)) {
107             exprValidators.remove(v);
108             setChanged();
109             notifyObservers();
110         }
111     }
112
113     public ArrayList<ExpressionValidator> getExpressionValidators()
            {
114         return exprValidators;
115     }
116
117     public ArrayList<PacketValidator> getPacketValidators() {
118         return children;
119     }
120
121     /**
122      * Sets the packet type depending on what TypeMap the name given
             is found.
123      * The type is defined as a {@link PacketType}.
124      *
125      * @param typeName
126      */
127     private void setPacketType(String typeName) {
128         if (TypeMaps.getConfirmationTypeMap().getType(typeName) !=
                 null) {
129             this.packetType = PacketType.CONFIRM;
130         } else if (TypeMaps.getTeledataTypeMap().getType(typeName) !=
                   null) {
131             this.packetType = PacketType.TELEDATA;
132         }
133     }
134
135     @Override
136     public void doAddChild(TreeItem item) {
137         if (item instanceof PacketValidator)
138             children.add((PacketValidator) item);
139     }
140
```

```
141     @Override
142     public ArrayList<TreeItem> getChildren() {
143        return new ArrayList<TreeItem>(children);
144     }
145
146     /**
147      * Validates the given packet with the underlying
148      * {@link ExpressionValidator}s to see if it is something that
           was expected.
149      *
150      * @param packet
151      *            the {@link Packet} to be validated upon
152      * @return A {@link PacketResult} stating if the {@link Packet}
           was accepted
153      *         by the {@link PacketValidator}
154      */
155     public PacketResult validate(Packet packet) {
156        packetResult = new PacketResult(this);
157        try {
158           packetResult.setReceivedPacket(packet);
159           String packetName = TestManager.getPacketTypeName(packet);
160           if (packetName != null && typeName.equals(packetName)) {
161              if (packetType != null && packet.getPacketType() ==
                      packetType) {
162                 if (getParamsArguments(packet)) {
163                    if (exprValidators.size() == 0) {
164                       packetResult.noValidators();
165                    }
166                    for (int i = 0; i < exprValidators.size(); i++) {
167                       ExpressionValidator val = exprValidators.get(i
                              );
168                       String paramName = val.getParamName();
169                       DTUSatParameterArgument dtuParamArg = new
                              DTUSatParameterArgument(
170                             paramName);
171                       ValidatorResult vResult = new ValidatorResult(
                              val,
172                             dtuParamArg, checkParamName(val,
173                                   dtuParamArg));
174                       packetResult.addValidatorResult(vResult);
175                    }
176                 }
177                 // validate
178              }
179           } else {
180              packetResult.setWrongPacket();
181           }
182        } catch (Exception e) {
183           logger.severe(e.getMessage());
184        }
185        return packetResult;
186
187     }
188
189     /**
```

```
190     * Checks if the given {@link Parameter} received on the packet
191     * corresponding to the {@link DTUSatParameterArgument}
          parameter name is
192     * accepted by the given {@link ExpressionValidator}.
193     *
194     * @param val
195     *            the {@link ExpressionValidator} to be tested upon.
196     * @param paramArg
197     *            the {@link DTUSatParameterArgument} containing the
          parameter
198     *            name to be tested. Will have the corresponding
199     *            {@link Argument} stored on it afterwards.
200     * @return <b>True</b> in case that we had received what was
          expected.
201     * @throws Exception
202     */
203    private boolean checkParamName(ExpressionValidator val,
204          DTUSatParameterArgument paramArg) throws Exception {
205
206       for (int j = 0; j < parameters.size(); j++) {
207          logger.info("We have " + paramArg.getParameterName()
208                + " and compare with " + parameters.get(j).getName()
                      );
209
210          if (parameters.get(j).getName().equals(paramArg.
               getParameterName())) {
211             Argument arg = arguments[j];
212             paramArg.setArgument(arg);
213             logger.info("We get result = "
214                   + val.validate(parameters.get(j), arg)
215                   + " on parameter :" + parameters.get(j)
216                   + " and argument " + arg);
217             return val.validate(parameters.get(j), arg);
218          }
219       }
220       return false;
221    }
222
223    /**
224     * Will extract that {@link Parameter}s and {@link Argument}s
          from the
225     * received packet, and store them in local ArrayLists for later
          usage. <br />
226     * Currently we support {@link ConfirmationPacket}s and
227     * {@link TeledataPacket}s.
228     *
229     * @param packet
230     *            the {@link Packet} which the parameters and
          arguments will be
231     *            extracted from.
232     * @return <b>True</b> if everything was as expected and the
          arguments and
233     *         parameters were extracted properly.
234     */
235    private boolean getParamsArguments(Packet packet) {
```

```
236         boolean result = false;
237         if (packet.getPacketType() == PacketType.CONFIRM) {
238             ConfirmationPacket cPacket = (ConfirmationPacket) packet;
239             result = cPacket.getName().equals(typeName);
240 
241             parameters = cPacket.getConfirmationType().getParameters()
                    ;
242             arguments = cPacket.getResults();
243 
244         } else if (packet.getPacketType() == PacketType.TELEDATA) {
245             TeledataPacket tPacket = (TeledataPacket) packet;
246             result = tPacket.getName().equals(typeName);
247 
248             parameters = tPacket.getDataType().getParameters();
249             arguments = tPacket.getData();
250         }
251 
252         return result;
253 
254     }
255 
256     @Override
257     public String toString() {
258         if (typeName != null) {
259 
260             return getTypeName();
261         }
262         return "Packet not set";
263     }
264 
265     @Override
266     public PacketValidator clone() {
267         PacketValidator clonedSelf;
268         if (typeName != null) {
269             clonedSelf = new PacketValidator(new String(typeName));
270         } else {
271             clonedSelf = new PacketValidator();
272         }
273 
274         for (ExpressionValidator exprValOrig : exprValidators) {
275             clonedSelf.addExpressionValidator(exprValOrig.clone());
276         }
277 
278         for (PacketValidator packetValOrig : children) {
279             try {
280                 clonedSelf.addChild(packetValOrig.clone());
281             } catch (Exception e) {
282                 logger.severe(e.getMessage());
283             }
284         }
285 
286         RootGuard clonedRoot = rootGuard.clone();
287         clonedSelf.setRootGuard(clonedRoot);
288 
289         return clonedSelf;
```

```
290     }
291
292     /**
293      * Will update the expression so they corresponds to the given
           expressions.
294      * This will be done when a Parent has been cloned and all
           expression need
295      * to be updated to math the parents new expressions.
296      *
297      * @param expression
298      */
299     public void updateReferences(ArrayList<Variable> expression) {
300        for (PacketValidator packetValidator : children) {
301           packetValidator.updateReferences(expression);
302        }
303
304        for (ExpressionValidator exprVal : exprValidators) {
305           exprVal.updateExpression(expression);
306        }
307
308        rootGuard.updateExpression(expression);
309     }
310
311     @Override
312     protected void doDeleteChild(TreeItem child) throws Exception {
313        children.remove(child);
314
315     }
316
317     public PacketResult setGuardFail() {
318        if (packetResult == null) {
319           packetResult = new PacketResult(this);
320        }
321        packetResult.setGuardFai();
322        return packetResult;
323     }
324
325     @Override
326     public testResultEnum getEnumResult() {
327        if (packetResult == null)
328           return testResultEnum.NONE;
329        return packetResult.getResult();
330     }
331
332     public PacketResult getPacketResult() {
333        return packetResult;
334     }
335
336     /**
337      * Call .evaluate() on the the root guard
338      *
339      * @return <b>true</b> if all guards succeeded.
340      * @throws Exception
341      */
342     public boolean evaluteGuard() throws Exception {
```

```
343         boolean result = rootGuard.evaluate();
344         logger.info("Result: " + result + " when evaluating root");
345         return result;
346     }
347 
348     /**
349      * Validates whether all information needed has been set. This
           means that
350      * all {@link ExpressionValidator}s, {@link PacketValidator} and
351      * {@link Guard}s contained will be validated for errors.
352      *
353      * @param errorList
354      *            a list containing all errors occured so far.
355      * @param basicTemplate
356      *            the {@link BasicTemplate} which the {@link
           PacketValidator} is
357      *            included in.
358      */
359     public void validate(List<ErrorValue> errorList, BasicTemplate
           basicTemplate) {
360         if (typeName == null || typeName.length() == 0) {
361             errorList
362                     .add(new ErrorValue(
363                             "No packet has been chosen to validate against
                                   in PacketValidator",
364                             basicTemplate));
365         }
366         if (packetType == null) {
367             errorList.add(new ErrorValue(
368                     "No packetType set in PacketValidator",
                            basicTemplate));
369         }
370         if (rootGuard != null) {
371             rootGuard.validate(errorList, basicTemplate);
372         } else {
373             errorList.add(new ErrorValue(
374                     "RootGuard is not set in PacketValidator",
                            basicTemplate));
375         }
376 
377         if (exprValidators != null) {
378             for (ExpressionValidator exprVal : exprValidators) {
379                 exprVal.validate(errorList, basicTemplate);
380             }
381         }
382 
383         if (children != null) {
384             for (PacketValidator pVal : children) {
385                 pVal.validate(errorList, basicTemplate);
386             }
387         }
388     }
389 
390     /**
```

```
391      * Checks if the given variable is being used by its children, the
392      * {@link ExpressionValidator}s or the {@link Guard}s, inorder to determine
393      * if it can be deleted.
394      *
395      * @param variable
396      *            the variable to be checked.
397      * @return <b>True</b> if the variable is being used.
398      */
399     public boolean isVariableInUse(Variable variable) {
400        for (PacketValidator pv : children) {
401           if (pv.isVariableInUse(variable)) {
402              return true;
403           }
404        }
405
406        if (rootGuard.isVariableInUse(variable)) {
407           return true;
408        }
409
410        for (ExpressionValidator exprVal : exprValidators) {
411           if (exprVal.isVariableInUse(variable)) {
412              return true;
413           }
414        }
415        return false;
416     }
417
418     /**
419      * Setup the Observer-Observable Hierarchy, so the GUI can be notified about
420      * changes.
421      */
422     public void setupObserverObservableHierarchy() {
423        for (PacketValidator pv : children) {
424           pv.addObserver(this);
425           pv.setupObserverObservableHierarchy();
426        }
427        if(rootGuard!=null)
428        {
429           rootGuard.addObserver(this);
430           rootGuard.setupObserverObservableHierarchy();
431        }
432
433     }
434
435  }
```

model/Scenario.java

```
1  package util.nominaltest2.model;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.concurrent.ExecutionException;
6  import java.util.concurrent.Future;
7  import java.util.logging.Logger;
8
9  import org.simpleframework.xml.Element;
10 import org.simpleframework.xml.ElementList;
11 import org.simpleframework.xml.ElementListUnion;
12 import org.simpleframework.xml.Path;
13 import org.simpleframework.xml.Root;
14
15 import util.nominaltest2.model.errors.ErrorValue;
16 import util.nominaltest2.model.results.TestResult.testResultEnum;
17 import util.nominaltest2.model.template.BasicTemplate;
18 import util.nominaltest2.model.template.DefinitionTemplate;
19 import util.nominaltest2.model.template.ParameterTemplate;
20 import util.nominaltest2.model.template.Template;
21 import util.nominaltest2.model.test.RootTest;
22 import util.nominaltest2.model.test.Test;
23
24 @Root
25 public class Scenario extends TreeItem implements Runnable {
26    private final Logger logger = Logger.getLogger(getClass().
          getName());
27
28    @Path("RootTemplates")
29    @ElementListUnion( {
30          @ElementList(entry = "BasicTemplate", type = BasicTemplate
                .class, inline = true, required = false),
31          @ElementList(entry = "DefinitionTemplate", type =
                DefinitionTemplate.class, inline = true, required =
                false) })
32    ArrayList<ParameterTemplate> rootTemplates = new ArrayList<
          ParameterTemplate>();
33
34    @Element(name = "RootSequentialCollectionTest")
35    RootTest rootTest;
36
37    // MainFrame frame;
38    // JFrame popUp;
39    TestManager model;
40    Thread thread;
41
42    public void setThread(Thread thread) {
43       this.thread = thread;
44    }
45
46    public Scenario() {
47       rootTest = new RootTest();
48       rootTest.addObserver(this);
49    }
```

```
50
51      public Scenario(TestManager model) {
52          this();
53          this.model = model;
54      }
55
56      public void setModel(TestManager model) {
57          this.model = model;
58      }
59
60      public Test getRootTest() {
61          return rootTest;
62      }
63
64      public void addTemplate(ParameterTemplate t) {
65          rootTemplates.add(t);
66          t.addObserver(this);
67          t.setRoot(true);
68      }
69
70      @Override
71      public ArrayList<TreeItem> getChildren() {
72          return new ArrayList<TreeItem>(rootTemplates);
73      }
74
75      public void addTest(Test test) {
76          try {
77              rootTest.addChild(test);
78          } catch (Exception e) {
79              logger.severe(e.getMessage());
80          }
81      }
82
83      /**
84       * Sends the rootTest and fetches results
85       */
86      public void sendRootTest() {
87          if (rootTest != null) {
88              rootTest.send();
89          }
90      }
91
92      /**
93       * Will execute the RootTest using Futures, getting the result.
94       */
95      public void getResults() {
96
97          Future<Void> futureRes = TestManager.getExecutor().submit(
                rootTest);
98          try {
99              futureRes.get();
100         } catch (InterruptedException e) {
101             logger.info("The execution of Scenario was interrupted");
102             rootTest.interruptFuture();
103             Thread.currentThread().interrupt();
```

```
104             return;
105         } catch (ExecutionException e) {
106             logger.severe(e.getMessage());
107         }
108 
109 
110         model.saveResult();
111     }
112 
113     public void delete(TreeItem item) {
114         rootTemplates.remove(item);
115     }
116 
117     @Override
118     protected void doAddChild(TreeItem item) throws Exception {
119         if (item instanceof ParameterTemplate) {
120             addTemplate((ParameterTemplate) item);
121         } else {
122             throw new Exception("Can only add ParameterTemplate to
                    TestManager");
123         }
124 
125     }
126 
127     public void addChildToItem(TreeItem parent, TreeItem child)
128             throws Exception {
129         parent.addChild(child);
130     }
131 
132     public boolean deleteItem(TreeItem parent, TreeItem child)
133             throws Exception {
134         if (child instanceof ParameterTemplate) {
135             ParameterTemplate paramTemplate = (ParameterTemplate)
                        child;
136             if (paramTemplate.isRoot()
137                     && (templatesHasDescendant(child) || rootTest
138                             .hasDescendant(child))) {
139                 return false;
140             }
141         }
142         if (parent != null && child != null) {
143             parent.deleteChild(child);
144         }
145         return true;
146     }
147 
148     private boolean templatesHasDescendant(TreeItem item) {
149         for (ParameterTemplate t : rootTemplates) {
150             if (t.hasDescendant(item)) {
151                 return true;
152             }
153         }
154         return false;
155     }
```

```
156     @Override
157     protected void doDeleteChild(TreeItem child) throws Exception {
158        rootTemplates.remove(child);
159
160     }
161
162     @Override
163     public testResultEnum getEnumResult() {
164        return rootTest.getEnumResult();
165     }
166
167     @Override
168     public String toString() {
169        return "Root Templates";
170     }
171
172     public void validateRecursively(List<ErrorValue> errorList) {
173        logger.info("Validating rootTemplates");
174        for (Template t : rootTemplates) {
175           t.validateRecursively(errorList);
176        }
177        logger.info("Validating rootTest");
178        rootTest.validateRecursively(errorList);
179     }
180
181     public void setupObserverObservableHierarchy() {
182        for (Template t : rootTemplates) {
183           t.addObserver(this);
184           t.setupObserverObservableHierarchy();
185        }
186        rootTest.addObserver(this);
187        rootTest.setupObserverObservableHierarchy();
188     }
189
190     @Override
191     public void run() {
192        sendRootTest();
193        if (!Thread.currentThread().isInterrupted()) {
194           getResults();
195        }
196     }
197 }
```

model/TestManager.java

```
1  package util.nominaltest2.model;
2
3  import java.io.File;
4  import java.io.IOException;
5  import java.util.ArrayList;
6  import java.util.Date;
```

```
7  import java.util.Observable;
8  import java.util.Observer;
9  import java.util.concurrent.ExecutorService;
10 import java.util.concurrent.Executors;
11 import java.util.logging.FileHandler;
12 import java.util.logging.Handler;
13 import java.util.logging.Level;
14 import java.util.logging.Logger;
15 import java.util.logging.SimpleFormatter;
16
17 import org.simpleframework.xml.Root;
18 import org.simpleframework.xml.Serializer;
19 import org.simpleframework.xml.core.Persister;
20 import org.simpleframework.xml.strategy.CycleStrategy;
21 import org.simpleframework.xml.strategy.Strategy;
22 import org.simpleframework.xml.stream.CamelCaseStyle;
23 import org.simpleframework.xml.stream.Format;
24 import org.simpleframework.xml.stream.Style;
25
26 import util.nominaltest2.model.expression.singleExpressions.
       Variable;
27 import util.nominaltest2.model.results.TestResult;
28 import util.nominaltest2.model.sutAdapter.SAI;
29 import util.nominaltest2.model.sutAdapter.dummy.SUTAdapterDummy;
30 import util.nominaltest2.model.template.ParameterTemplate;
31
32 import common.dtusat2.packet.ConfirmationPacket;
33 import common.dtusat2.packet.Packet;
34 import common.dtusat2.packet.SignalPacket;
35 import common.dtusat2.packet.TeledataPacket;
36 import common.types.TypeMaps;
37
38 @Root
39 public class TestManager extends Observable implements Observer {
40    private static SAI sa = new SUTAdapterDummy();
41    private static ExecutorService executor = Executors.
          newCachedThreadPool();
42
43    private static int basicTemplateIDCounter;
44    private static int definitionTemplateIDCounter;
45
46    private final Logger logger = Logger.getLogger(getClass().
          getName());
47
48    Serializer serializer;
49    Scenario currentScenario;
50
51    public TestManager() {
52
53       Handler handler;
54       try {
55          handler = new FileHandler("Nominaltest2.log");
56          handler.setFormatter(new SimpleFormatter());
57          Logger.getLogger("").addHandler(handler);
```

```
58          Logger.getLogger("").setLevel(Level.INFO); // TODO: Load
                fra cfg-fil
59          // i stedet?
60       } catch (SecurityException e) {
61          logger.severe(e.getMessage());
62       } catch (IOException e) {
63          logger.severe(e.getMessage());
64       }
65
66       resetIDCounters();
67
68       setScenario(new Scenario(this));
69
70       Strategy strategy = new CycleStrategy("id", "ref");
71       Style style = new CamelCaseStyle();
72       Format format = new Format(style);
73       serializer = new Persister(strategy, format);
74
75       try {
76          TypeMaps.loadTypes();
77       } catch (Exception e1) {
78          logger.severe(e1.getStackTrace().toString());
79       }
80       sa.createConnection();
81    }
82
83    public static int getNextBasicTemplateID() {
84       return basicTemplateIDCounter++;
85    }
86
87    public static int getNextDefinitionTemplateID() {
88       return definitionTemplateIDCounter++;
89    }
90
91    private void setScenario(Scenario scenario) {
92       currentScenario = scenario;
93       currentScenario.addObserver(this);
94       setChanged();
95       notifyObservers("NewScenario");
96    }
97
98    public static SAI getSA() {
99       return sa;
100   }
101
102   public static ExecutorService getExecutor() {
103      return executor;
104   }
105
106   /**
107    * Saves the current scenario to the given file. This file must
          support the
108    * XML-format used in the system.
109    *
110    * @param file
```

```
111      * @throws Exception
112      */
113     public void saveScenario(File file) throws Exception {
114        logger.info("Saving to " + file);
115        serializer.write(currentScenario, file);
116     }
117
118     /**
119      * Opens the scenario specified in the given file. This file
           must support
120      * the XML-format used in the system.
121      *
122      * @param file
123      * @throws Exception
124      */
125     public void openScenario(File file) throws Exception {
126        logger.info("Opening scenario");
127        resetIDCounters();
128        Scenario newScenario = serializer.read(Scenario.class, file);
129        setupObserverObservableHierarchy(newScenario);
130        newScenario.setModel(this);
131        setScenario(newScenario);
132     }
133
134     private void resetIDCounters() {
135        basicTemplateIDCounter = 1;
136        definitionTemplateIDCounter = 1;
137     }
138
139     private void setupObserverObservableHierarchy(Scenario
           newScenario) {
140        newScenario.setupObserverObservableHierarchy();
141     }
142
143     public static String getPacketTypeName(Packet packet) {
144        if (packet instanceof ConfirmationPacket) {
145           return ((ConfirmationPacket) packet).getName();
146        } else if (packet instanceof TeledataPacket) {
147           return ((TeledataPacket) packet).getName();
148        } else if (packet instanceof SignalPacket) {
149           return ((SignalPacket) packet).getName();
150        } else {
151           return "Packet not supported (is not of type Confirmaion
                or Teledata";
152        }
153     }
154
155     public void addVariableToParameterTemplate(Variable p,
           ParameterTemplate t) {
156        t.addVariable(p);
157        notifyVariablesChanged();
158     }
159
160     public void deleteVariableFromParameterTemplate(Variable p,
161           ParameterTemplate t) {
```

```
162         t.removeVariable(p);
163         notifyVariablesChanged();
164     }
165 
166     private void notifyVariablesChanged() {
167         setChanged();
168         notifyObservers("parameterExpressions");
169     }
170 
171     @Override
172     public void update(Observable o, Object arg) {
173         setChanged();
174         notifyObservers(arg);
175     }
176 
177     public Scenario getScenario() {
178         return currentScenario;
179     }
180 
181     public void saveResult() {
182         TestRunResults result = new TestRunResults();
183         Date time = new Date(System.currentTimeMillis());
184         result.setTime(time);
185         ArrayList<TestResult> results = currentScenario.getRootTest()
186                 .getTestResults();
187         for (TestResult tr : results) {
188             tr.updateForSaving();
189         }
190         result.setResults(results);
191 
192         boolean folderExist = true;
193         File folder = new File("results");
194         if (!folder.exists()) {
195             if (!folder.mkdir()) {
196                 folderExist = false;
197             }
198 
199         }
200         if (folderExist) {
201             File file = new File("results/TestExecution on the " +
                    time
202                     + ".xml");
203             Serializer serializer = new Persister();
204             try {
205                 serializer.write(result, file);
206             } catch (Exception e) {
207                 logger.severe("File " + file + " not saved");
208                 logger.severe(e.getMessage());
209             }
210         }
211     }
212 
213 }
```

model/TestRunResults.java

```
1  package util.nominaltest2.model;
2
3  import java.util.ArrayList;
4  import java.util.Date;
5
6  import org.simpleframework.xml.Attribute;
7  import org.simpleframework.xml.ElementList;
8
9  import util.nominaltest2.model.results.TestResult;
10
11 public class TestRunResults {
12
13     @Attribute
14     Date time;
15
16     @ElementList
17     ArrayList<TestResult> results = new ArrayList<TestResult>();
18
19
20     public void setResults(ArrayList<TestResult> results) {
21         this.results = results;
22     }
23
24     public void setTime(Date time) {
25         this.time = time;
26     }
27
28 }
```

model/TreeItem.java

```
1  package util.nominaltest2.model;
2
3  import java.util.ArrayList;
4  import java.util.Observable;
5  import java.util.Observer;
6
7  import org.simpleframework.xml.Attribute;
8
9  import util.nominaltest2.model.results.TestResult.testResultEnum;
10
11 /**
12  * Used in order to incorporate the same functionality in most
       parts of our
13  * system, as well as provide a an interface to be used by the GUI.
14  *
15  */
```

```
16 public abstract class TreeItem extends Observable implements
       Observer {
17
18     @Attribute(required = false)
19     protected String name;
20     @Attribute(required = false)
21     protected String description;
22
23     public void setName(String name) {
24        this.name = name;
25        setChanged();
26        notifyObservers();
27     }
28
29     public String getName() {
30        return (name == null) ? "" : name;
31     }
32
33     public void setDescription(String description) {
34        this.description = description;
35     }
36
37     public String getDescription() {
38        return (description == null) ? "" : description;
39     }
40
41     /**
42      * Returns a HTML-description where the string is broken into
            lines if
43      * necessary
44      *
45      * @return
46      */
47     public String getHTMLDescription() {
48        final int charsPerLine = 50;
49
50        String rStr = "<HTML>";
51        if (description == null || description.equals("")) {
52           return "<no description>";
53        } else {
54           int totalLength = description.length();
55           int currentPos = 0;
56           while (currentPos < totalLength) {
57              int newPos = currentPos + charsPerLine;
58              rStr += description.subSequence(currentPos,
59                    newPos < totalLength ? newPos : totalLength)
60                    + "<br>";
61              currentPos = newPos;
62           }
63        }
64        return rStr + "</HTML>";
65
66     }
67
68     public abstract ArrayList<TreeItem> getChildren();
```

```
69
70      /**
71       * Gets the actual result from the template stored on the test.
72       *
73       * @return {@link testResultEnum}
74       */
75      public abstract testResultEnum getEnumResult();
76
77      /**
78       * Returns true if this TreeItem has the given node as a
             descendant
79       *
80       * @param descendant
81       * @return
82       */
83      public boolean hasDescendant(TreeItem descendant) {
84          ArrayList<TreeItem> children = getChildren();
85          if (children.contains(descendant))
86              return true;
87
88          for (TreeItem child : children) {
89              if (child.hasDescendant(descendant))
90                  return true;
91          }
92          return false;
93      }
94
95      protected abstract void doAddChild(TreeItem item) throws
             Exception;
96
97      public final void addChild(TreeItem item) throws Exception {
98          doAddChild(item);
99          item.addObserver(this);
100         setChanged();
101         notifyObservers();
102     }
103
104     protected abstract void doDeleteChild(TreeItem child) throws
             Exception;
105
106     public final void deleteChild(TreeItem item) throws Exception {
107         doDeleteChild(item);
108         setChanged();
109         notifyObservers();
110     }
111
112     @Override
113     public void update(Observable o, Object arg) {
114         setChanged();
115         notifyObservers(arg);
116
117     }
118
119     @Override
120     public String toString() {
```

```
121         return getName();
122     }
123
124 }
```