Automated Working Environment for Test and Simulation (AWETS)

H. Kolkman[1], L. Timmermans[2], M. Neefs[3]

[1]*TASK24*
P.O.Box 85066, 3508 AB Utrecht, The Netherlands
Email: han.kolkman@task24.nl

[2]*National Aerospace Laboratory NLR, Space department*
P.O.Box 15, 8300 AD Emmeloord, The Netherlands
Email: leo.timmermans@nlr.nl

[3]*Dutch Space B.V.*
P.O.Box 32070, 2303 DB Leiden, The Netherlands
Email: m.neefs@dutchspace.nl

ABSTRACT

A new environment for test automation using modern standards, technologies and platforms such as Eclipse, Java, XTCE, XML and EMF was developed. This environment automates the work necessary for test data definition, test procedure definition, execution and results analysis to a high degree. New concepts were implemented in a demonstrator system for evaluation.

INTRODUCTION

Current test and simulation tools often provide limited support for controlled execution of the work, as required for most space related projects. In preparation of a test or simulation, the user often needs to define and/or setup his/her working environment without any support by the tool. This entails related activities such as definition of monitoring and control data, test definition, test execution and test analysis and reporting. These preparatory and supporting activities lead to additional costs for projects, often not accounted for.

The Automated Working Environment for Test and Simulation (AWETS) is developed to study how to improve the efficiency of test and simulation activities, in particular for EuroSim based systems. The project was funded by the Netherlands Space Office (NSO) and executed by a consortium of NLR, Dutch Space and TASK24.

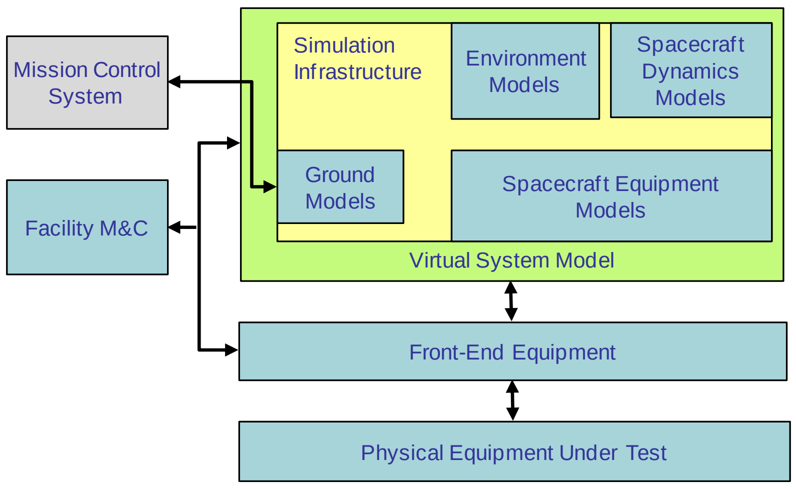

Mission Control System
Simulation Infrastructure
Environment Models
Spacecraft Dynamics Models
Ground Models
Spacecraft Equipment Models
Facility M&C
Virtual System Model
Front-End Equipment
Physical Equipment Under Test

Fig. 1. ECSS representation of a Test System (source: [1]). AWETS focuses on the Facility M&C part.

AWETS focuses on the "Facility Monitoring & Control (M&C)" part of a test/simulation system as defined by ECSS in [1], see Fig. 1. The AWETS platform monitors and controls test and simulation execution. In the AWETS demonstration setup that was created, EuroSim [2] was used for the simulation infrastructure and a demo simulation model served as the System Under Test (SUT).

The AWETS platform also adds tooling to efficiently prepare a test and simulation session beforehand and analyze the results afterwards. Four basic processes are distinguished within AWETS in this perspective: 1) Define data, 2) Define tests, 3) Execute tests and 4) Analyze test results. This is depicted in the data flow diagram in Fig. 2.

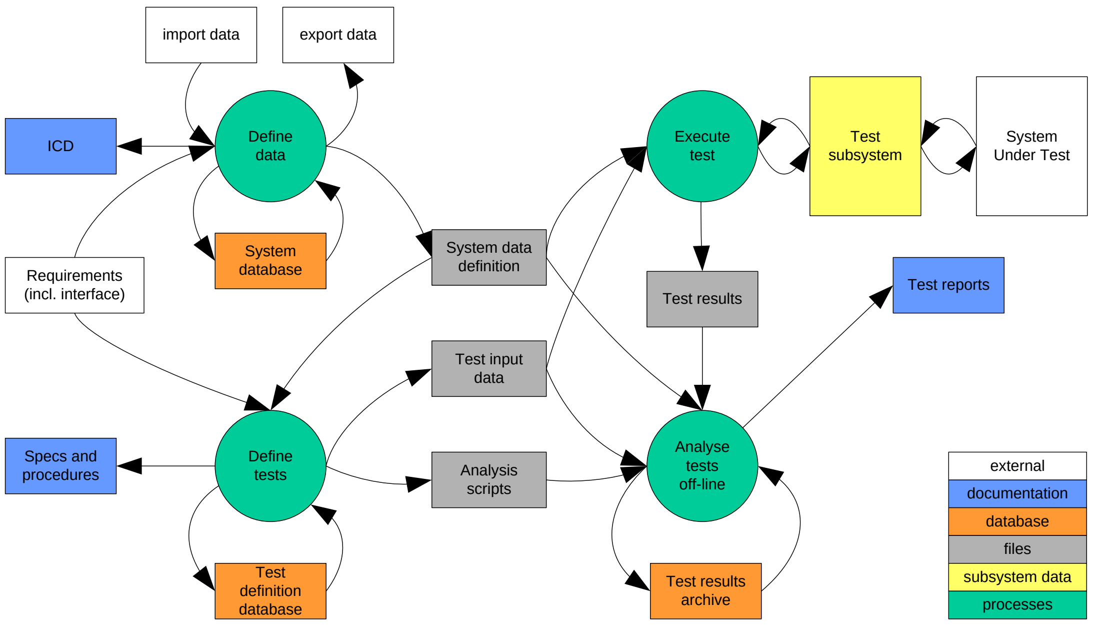


Fig. 2. Data flow diagram showing the four basic processes (in green) distinguished within AWETS and their data interactions.

OBJECTIVES

The main objectives of the AWETS system are:

- Automate simulation and verification activities during the complete spacecraft lifecycle.
- Create a re-usable test automation environment that can easily be expanded for specific project needs and test subsystems (TS) through well defined interfaces and data models.
- Integrate Test Data Definition, Test (Script) Definition, Test Execution and Test Results Analysis and Reporting processes in one environment.

Additional to these main objectives there were objectives such as to keep the platform portable and light weight (it should run on standard Linux and Windows PC) and to use open source and open standards as much as possible to keep the operational costs low. The system should provide detailed logging and high level test result summaries including system requirements coverage. Finally the platform should be easy to use with short learning times and usable in other domains than space too.

DESIGN DRIVERS AND ARCHITECTURAL CHOICES

Following design drivers were considered essential for the architecture:

- Light weight (it should run on standard PC)
- Portable (Linux, Windows)
- Use of modern and well accepted open source tooling
- Use of open standards
- Easy to use in new test and verification projects
- Near real time (data structures like used in TMTC have to be processed in such way that the data is available for automatic verification in test scripts)
- Easy to expand and possibility to integrate different features and processes in one environment.

Based on these design drivers it was decided to use a platform that supports a pluggable architecture. For this reason the open source Eclipse framework was selected as the base for the AWETS platform. Eclipse has a high degree of acceptation in the development community, extensive library support and proven flexibility to build dedicated user applications. The available Eclipse SDK's also provide excellent user friendly environments for development and debugging of test scripts.

Usage of well defined data models was another important decision in order to be able to develop generic tooling like data editors and data processing functionality, making the platform easy and flexible to use in new projects. Therefore the implementation of a test definition database was based on XML with a logical database model (structure definition) in XML Schema Definition (XSD). The XML solution is light weight and avoids the need for an additional database server.
Likewise, XML/XSD technology was selected for TMTC data definition in combination with XML Telemetric and Command Exchange (XTCE) XSD [3]. XTCE aims to guarantee the interchangeability of TMTC specifications between multiple partners in a space project. It allows development of standard tooling such as automatic generation of code for data packet processing from TMTC specifications.

Java was selected as the language for the tool implementation because of its platform independence, the availability of extensive open source libraries (e.g. for graphics (JFreeChart) and XML data binding) and because it is the Eclipse native language.

Java was also selected as Test Script language though the choice for Eclipse and the AWETS architecture supports implementation of other test script languages in future. Java offers the advantage of AWETS being capable to use so-called dynamic compilation (compiling and loading the test script code at test execution time) which provides great test script development, editing and debugging facilities in combination with the Eclipse Java SDK.

The Eclipse Modelling Framework (EMF) is an open source XML data binding library [4] that was selected because it runs as an Eclipse plugin, it is well supported and documented and it turned out to be easy to integrate in comparison to similar tools like e.g. Castor. In addition to that it offers a feature to build dedicated graphical XML editors from a XSD which run as a plugin in eclipse and can be used to create XSD compliant XML. With the EMF Ecore tooling relational databases are supported in XSD/XML.

AWETS provides basic functionality to monitor TS data. For more sophisticated data visualization, a coupling with NLR's Vincent tool was created via a built-in data server. Vincent provides a tool suite for rapid development (and evaluation) of simple to highly complex Synoptic Displays and Human Machine Interfaces [5].

ARCHITECTURE

Overview

Currently AWETS consists of the following components:

- AWETS Test Definition Database Editor: an EMF generated XML editor plugin that can be used to create and update a project test database.
- AWETS TMTC Definition Database Editor: an EMF generated XML editor plugin that can be used to create and update a file with monitoring and command packet definitions. The file complies with the XTCE meta-definitions (XML Schema)
- AWETS TMTC Library Code Generator: A code generation tool that reads in a project's TMTC definition file in XTCE format, analyses it and generates a Java code library for the AWETS Test Execution application. The library provides e.g. factories for TMTC packet object instantiation, functionality for TC data population, commutation and decommutation of packages and provides datapool (access) facilities.
- AWETS Test Script Development and Execution Application: The actual test execution platform which primary task is to read/parse an XML test database and execute the selected tests. Numbers of passed and failed steps are counted and at the end a HTML test results summary is compiled including a requirements coverage overview. The Execution Environment provides facilities to monitor SUT data dictionary variables or datapool variables within numerical or graphical monitoring windows and includes a data interface to stream any selected data to dedicated Vincent Synoptic Displays. The same application can also be used to write and debug Java test scripts as it uses the full Java SDE from Eclipse. It is possible to halt between individual test cases and scripts and repeat any steps after e.g. script modification thanks to use of dynamic compilation.

- A Conversion tool to convert TMTC specifications between XTCE and SCOS2000 database format based on XML Style sheet Language Transformation (XSLT).
- A Conversion tool to convert an Excel list of requirements in AWETS Test Definition format.

As shown in Fig. 3 the AWETS components should ultimately integrate as plugins in the Eclipse framework to complete an AWETS Rich Client Platform using other plugins like EMF, Java SDE and Control Version System (CVS) to manage test and simulation projects. Note that AWETS offline test analysis tooling is not yet implemented.

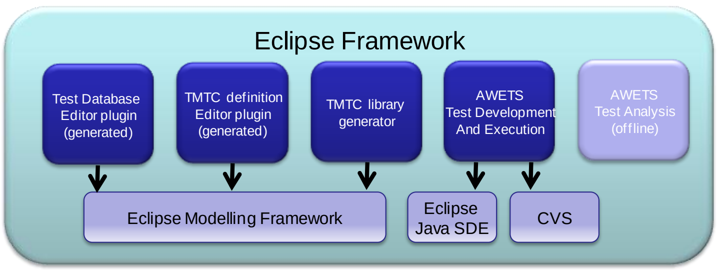

Eclipse Framework
Test Database Editor plugin (generated)
TMTC definition Editor plugin (generated)
TMTC library generator
AWETS Test Development And Execution
AWETS Test Analysis (offline)
Eclipse Modelling Framework
Eclipse Java SDE
CVS

Fig. 3. Integration of AWETS in the Eclipse framework and other Eclipse plugins.

The major components or data elements are further elucidated below.

Test Definition Database

As shown in Fig. 4, the AWETS test database is an XML database to store test specifications, test cases and test steps as major entities that make up a test set. In addition it stores related information such as requirements, Software Problems Reports (SPR's) and configuration data and link these to test specifications, test cases or test steps. Test specifications are the highest level in the hierarchy. A test specification could for example contain all or a (sub)set of component tests, system tests or integration tests. A test specification can contain one or more test cases that have to be executed in a defined order.

Test cases often refer to the verification of one or more specific requirements. Each test case relates to a specific configuration of AWETS and the SUT and before running a test case the system normally goes through an initialization/start up cycle. A test case may consist of multiple test steps that have to be executed in a defined order. A specific test abort step may be defined in case of fatal errors.

Test steps are the smallest entities pointing to one particular Java test script that has to be executed. It is up to the test developer to decide on how to spread test stimuli and verifications over the individual test steps.

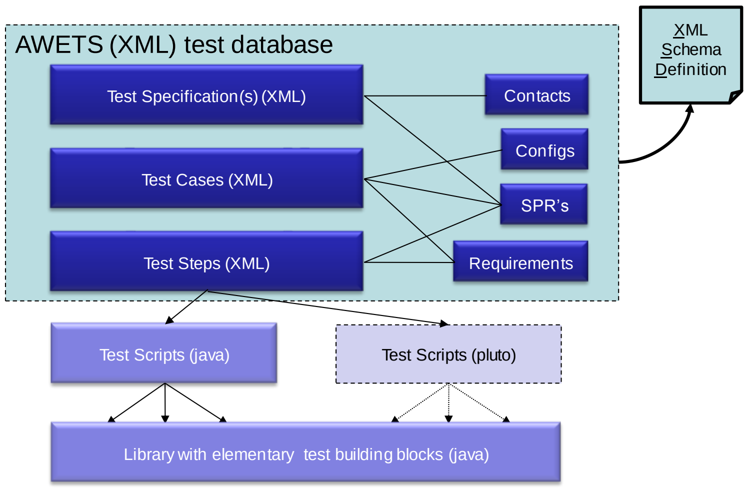


Fig. 4. The AWETS test definition database

AWETS Test Development and Execution Application

The AWETS test execution application has a layered architecture as shown in Fig. 5.

The Test Execution module in the Application layer controls the execution of tests by parsing the XML data definitions files and executing the referenced test steps by compiling, loading and executing Java test scripts at run time. It controls the execution progress and monitors the results. At the end of a test run it compiles a HTML formatted test report.

The other block in the Application layer is the Monitoring Control module which function is to setup specific monitoring windows per user request and connect the monitors to the relevant data sources. Standard monitors are a journal and a TMTC package monitoring window (shown in the top User Interface layer). Optionally numerical and graphical (JFreeChart [6]) monitors and/or data streaming to external Synoptic Displays can be can setup to monitor specific data parameters.

Central layer in the diagram is the Communication layer hiding all direct access to the IO channels to Test Subsystem (TS) devices. The communication manager module creates the correct TS interfaces at test initialization and addresses commands to the correct TS. The module subscribes itself to monitoring and event data from the TS and redistributes the data on request to specific observers in the higher layers such as the test execution module or to external Vincent synoptic display clients. The communication manager has access to the (de)communication libraries to decommutate TM data packets and access to datapool parameters on observer request. Via the communication manager TC messages can be prepared and sent to the TS.

The Test Subsystem Interface layer at the bottom offers a generic TS interface to the communication layer and implements a factory to create specific TS interfaces based on the specific interface API's from the Driver layer.

The top User Interface layer finally shows the user interfaces: the Main GUI from within the user can control test execution, the already mentioned monitoring windows and the standard Eclipse Java SDE GUI's which are important when modifying and debugging test scripts.

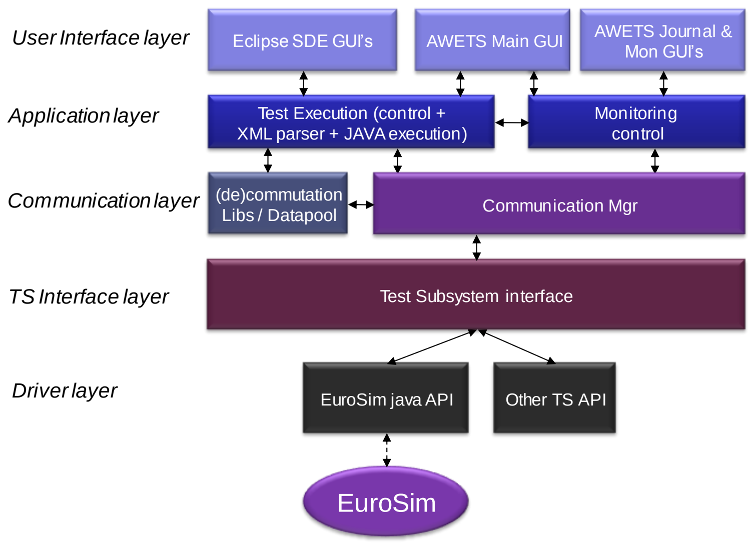


Fig. 5. Test execution architecture

AWETS TMTC Library Code Generator

As a TMTC specification for a project only specifies *how* the data in the project TMTC data packages is organised, additional structures are needed in test execution software that can be used to store *what* data (content) is in those packages. Preferably it shall be possible to store decoded TM package data contents in a more permanent way to avoid repeated decommutation of the same TM packages in time. In addition to that it was decided to prefer hardcoded above dynamic commutation and decommutation functionality for reasons of resource efficiency and to create a datapool structure with the collection of all TM parameters to be able to store and have easy access to the latest received TM parameter values. For these purposes a code generation tool was developed to generate Java library code for project specific TMTC processing.

The code generation tool reads and analyses a project specific TMTC definition in XML format, which is possible thanks to the TMTC meta-definitions in XTCE to which the TMTC definition must comply. This concept is elucidated in Fig.6. The tool uses the EMF plugin to read a TMTC definition file and bind the data to objects for efficient analysis. The analysis consists of searching for XTCE specific structures and creating the Java code.

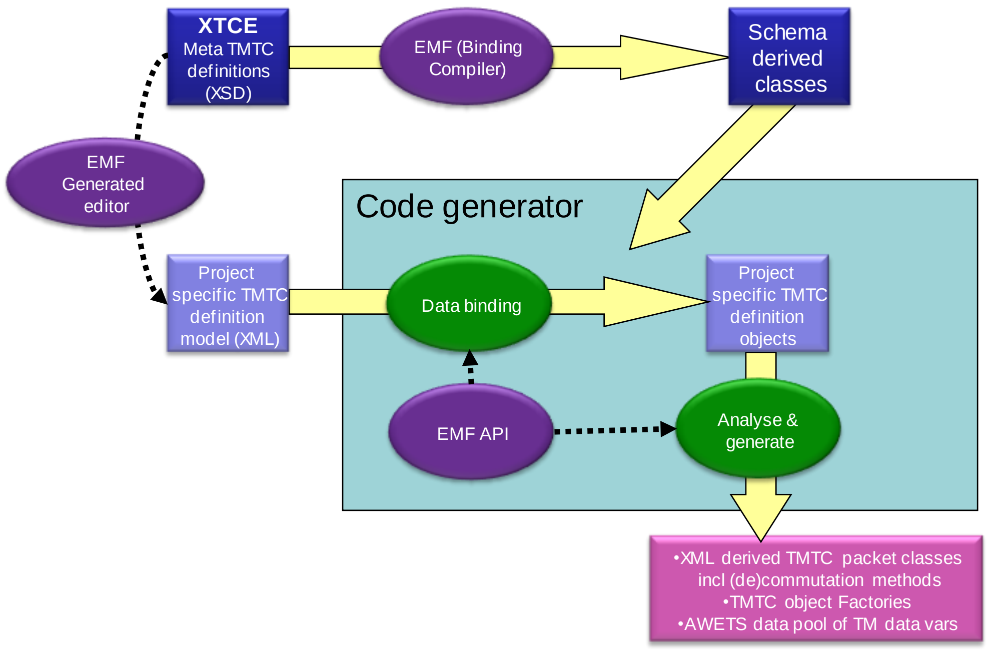

XTCE
Meta TMTC definitions (XSD)
EMF (Binding Compiler)
Schema derived classes
EMF Generated editor
Code generator
Project specific TMTC definition model (XML)
Data binding
Project specific TMTC definition objects
EMF API
Analyse & generate
•XML derived TMTC packet classes incl (de)commutation methods
•TMTC object Factories
•AWETS data pool of TM data vars

Fig. 6. Code generation process to create Java code from TMTC definitions defined in XML

THE AWETS DEMONSTRATOR

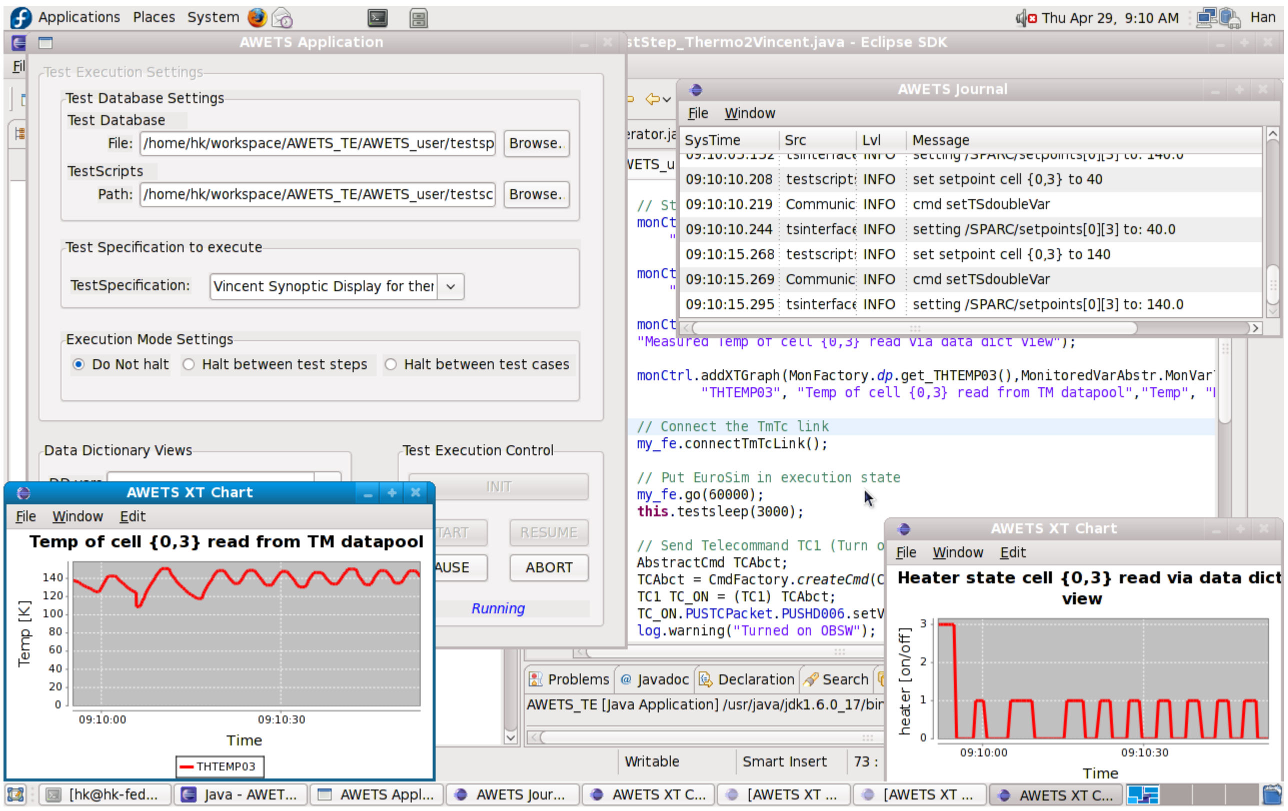
An AWETS demonstrator system was successfully built consisting of the above mentioned components and with the aim to create a platform to evaluate the concepts and that can serve as a starting base for future projects. The system is based on Eclipse Ganymede 3.4.2, with EMF 2.4 and Java 1.6. The system works with XTCE version 1.1 but only a limited set of the large number of XTCE capabilities is used by the demo SUT and therefore is actually implemented in the TMTC library code generation.

The TS consists of EuroSim running a simple model with three telecommands and one telemetry message that comply with PUS. The model runs on latest EuroSim development version Mk4.2. The interface to EuroSim uses the existing EuroSim external client Java API.

The test execution application implements a data server to supply data streams to Vincent synoptic display clients and can handle multiple channels with mixes of TM datapool variables, simulation model data dictionary variables and EuroSim parameters. A number of basic Synoptic Displays were created for testing.

All components were developed and tested on Linux. AWETS test execution application testing was as much as possible, automated using the features of the application itself.

Fig. 7 shows a screenshot of the test execution application.



Test Execution Settings

Test Database Settings

Test Database
File: /home/hk/workspace/AWETS_TE/AWETS_user/testsp

TestScripts
Path: /home/hk/workspace/AWETS_TE/AWETS_user/testsc

Test Specification to execute
TestSpecification: Vincent Synoptic Display for ther

Execution Mode Settings
[● Do Not halt] ○ Halt between test steps ○ Halt between test cases

Test Execution Control: Running

```
SysTime       Src          Lvl   Message
09:10:10.208  testscript:  INFO  set setpoint cell {0,3} to 40
09:10:10.219  Communic     INFO  cmd setTSdoubleVar
09:10:10.244  tsinterface  INFO  setting /SPARC/setpoints[0][3] to: 40.0
09:10:15.268  testscript:  INFO  set setpoint cell {0,3} to 140
09:10:15.269  Communic     INFO  cmd setTSdoubleVar
09:10:15.295  tsinterface  INFO  setting /SPARC/setpoints[0][3] to: 140.0
```

```
"Measured Temp of cell {0,3} read via data dict view");

monCtrl.addXTGraph(MonFactory.dp.get_THTEMP03(),MonitoredVarAbstr.MonVar
        "THTEMP03", "Temp of cell {0,3} read from TM datapool","Temp", "I

// Connect the TmTc link
my_fe.connectTmTcLink();

// Put EuroSim in execution state
my_fe.go(60000);
this.testsleep(3000);

// Send Telecommand TC1 (Turn o
AbstractCmd TCAbct;
TCAbct = CmdFactory.createCmd(C
TC1 TC_ON = (TC1) TCAbct;
TC_ON.PUSTCPacket.PUSHD006.setV
log.warning("Turned on OBSW");
```

AWETS_TE [Java Application] /usr/java/jdk1.6.0_17/bin

Fig. 7. AWETS screenshot showing the test execution application.

CONCLUSIONS AND LESSONS LEARNED

A new test automation environment was developed and new tools and concepts were successfully implemented in a demonstrator system. The demonstrator system was used to get experience which in general proved the usefulness of these tools and concepts. Below summarizes the main conclusions:

XTCE

In general it can be concluded that using TMTC specifications in XML format that comply with a XSD standard, allows the development of standard tooling to supports tool re-use and specification exchangeability. When working with strictly PUS compliant TMTC specifications it was felt however that XTCE is too generic. This causes problems when converting TMTC specifications between an XTCE format and stricter PUS specific database formats like e.g. SCOS2K MIB. The logical way to handle this is to define a specific PUS template in XTCE format using strict naming conventions that comply with dedicated conversion tooling. Advantage of the generic character of XTCE on the other hand, is that it allows the format and tooling to be used also for packet communication protocols other than PUS and in domains other than space.

EMF

The XML data binding tooling in EMF proved to be reliable and easy to integrate and was successfully used in the TMTC code generation tool. The graphical XML editor plugins that can be generated with the EMF for AWETS test database editing and TMTC definition editing, are easy to generate from XSD and in principle work well. However for large XML file editing these editors are not very suitable as everything data item has to be input manually.
The relations between the objects in the XML file are stored as an index number of the objects. This makes it tedious and difficult to review (changes to) the XML file, or edit the file manually.

Java

The use of Java for AWETS application development and as language for test scripts, turned out well. Though the AWETS application has no hard real time requirements, and no severe loads where tested yet, the general feeling is that modern Java run time engines are resource efficient. A demo application with EuroSim demo model, Vincent and

AWETS test execution application was installed and ran on Linux running in a virtual machine on a basic Windows laptop. In this configuration AWETS is roughly using same amount of processor resources as the demo’s EuroSim Controller process. The number of synoptic displays attached does hardly influence this. Adding graphical JFreeChart monitoring windows however adds a few percent per plot. Apparently these are not that efficient.
Biggest advantage of Java is the availability of many reliable open source libraries for e.g. logging, threading, XML/XSLT processing, graphics, user interfaces, etc., which form truly platform independent solutions.
As a language for test scripts Java provides a good solution too: with dedicated libraries of test functions much complexity can be hidden for the test writer and well readable and powerful test scripts can be created in short term. Finally editors with content assist and syntax high lighting as provided by the Eclipse SDE in combination with good use of Javadoc, are of great help to the test writer.

XSLT
XSLT version 1.0 has been used to convert XTCE database format to e.g. SCOS2000 MIB format. However, the language has limited capabilities to handle the more complex XTCE constructs like expansion of array constructs. XSLT 2.0 is expected to solve this.

Outlook

A number of issues still need to be worked out like: - further integration of all AWETS components into one AWETS Rich Client Application (RCP), - an offline test analysis tool including report over different test runs, - archiving of TMTC, and - automatic generation of documentation. To achieve this possible options being considered by the consortium are: - to include development in a real project, - search cooperation with other parties, - make AWETS open source and / or - get additional funding otherwise.

The results of the ESA ITT AO/1-6412/10/NL/LvH concerning “Automation of Space System Test Data Collecting, Processing and Reporting” will also provide valuable input for the further development of the AWETS system.

Acknowledgements

The authors like to thank all organisations and persons for their contributions and support to the AWETS project, including the Netherlands Space Office (NSO) for providing part of the budget from their PEP programme, Awad Hassan (TASK24), Piet Vriend, Erik de Koster (DS), Jacques Spaa, Fennanda Doctor, Ronald Verhoeven (NLR), Serge Valera (ESTEC), Kevin Rice (NASA GSFC, GST Inc.).

References

[1] ECSS-E-10, Space Engineering – System Modelling and Simulation
[2] EuroSim, http://www.eurosim.nl
[3] XTCE, http://en.wikipedia.org/wiki/XML_Telemetric_and_Command_Exchange
[4] EMF, http://en.wikipedia.org/wiki/Eclipse_Modeling_Framework
[5] Vincent, http://vincent.nlr.nl
[6] JFreeChart, http://www.jfree.org/jfreechart/index.html