



Cost Efficient Test System for Embedded Systems (CETES)

Authors:

Jürgen Hartung, juergen.hartung@koelsch-altmann.de
Kölsch & Altmann GmbH

Florian Prester, florian.prester@seppmed.de
sepp.med gmbh

Robert Schachner, r.schachner@rst-automation.de
RST Industrie Automation GmbH

Abstract

Motivation

Knowing the business of domains like automotive, medical and defense for many years we came up with the following approach:

Developing a really Cost Efficient Test System for Embedded Systems (CETES) – with respect to both test hardware and reinvented test process – is our goal. We provide you with the latest technology, open interfaces and a smart way to connect systems under test with customer specific test environments.

Design,

Starting with experienced model based test design technique (.mzT) customers will save time and money in the first step. From those models test cases are generated automatically. To generate the correct test cases .getmore – a test case generator – is used.

Manage, ...

With the new approach your quality assurance team will be able to manage all kinds of test. In order to support open interfaces and some standards we decided to use the Eclipse platform for developing the test management system. This way we believe that further integration of components used at your business is now much easier to handle than it was in past times.

... and Test

CETES has a strong and trustworthy base for establishing connections to your system under test. Together with Gamma, a software package that has already proven its merits in the context of the automation industry, we deliver a fast, predictable and lean method to control and monitor your connected hardware.

Keywords: Model based testing, Hardware-in-the-loop, Test system, Open hardware interface, CETES

1 Introduction

Costs saving actions are today always looked for. For testing activities there are many types of actions to be considered. New processes and techniques define new ways to reduce costs one-time and even better permanently. Reusability is one method to ensure sustainable efficiency gains.

For test systems the requirements and the derived test cases are one source for optimization. At projects like AUTOSAR [1] standardization about abstraction layers has led to new ideas. Application functions are defined as library components. Test cases therefore should be handled the same way.

Developing software products is a complex and step-by-step process. Depending on the development process and iterations the test needs even more iterations and the objects of test are distinguished. Therefore model based testing (MBT) has been developed. MBT uses development information e.g. models to derive test cases out of those automatically. Unfortunately those test cases are only usable for verification not for validation. To generate the "correct" test cases many strategies are designed and implemented with many approaches and tooling. For example risk based test strategies focus on some single features. Other strategies try to reduce the number of test cases to be executed. This is another source of reducing costs at an earlier phase.

Concerning hardware one goal is to use the equipment for almost every system under test. Because such hardware configuration is mainly driven by the most complex hardware requirements there are hardly any scaling effects. Small required hardware configurations are then based on huge hardware configurations, which make it sometimes hard to get the full picture. Reducing costs requires easy build-up and maintenance of hardware and its configuration on the one side on the other side you need a clear and standard software interface to address different hardware the same and easy way.

In this paper we describe a new solution for handling the actual and future requests to a test system for embedded systems.

2 The new approach

Model based software development is more and more the standard at product development. Based on a graphical description language combined with code generation is one advantage of that approach. Such models are nearly perfect for the usage inside an application function library. Independent from the target specific implementation language reuse is much easier.

In the context of model based testing above mentioned aspects behave quiet similar, but with more controllable options compared to model based software development. This is especially true when using the method model-centric testing (.mzT) and the related tool .getmore.

Different test strategies can be considered within one single model. Unlike lists of test commands copied and changed in minor items, the concept of "single source" patterns is applied here.

By that coverage of generated tests can be controlled at a central place allowing several options. Of course this requires additional support at the stage of test case management. Generated test cases need information about relationship and options to test models, their origin.

Requirements are the base for software development and test design. Every single functional requirement must finally be referenced within test cases. For model based test design requirements can now be inserted into the model. Later on, having test cases generated, this information about requirements usage is tagged to every test case itself. Comparing once used requirements with a defined set of requirements stored at the requirements database (here DOORS) gives everybody a clear message about achieved coverage.

Like in the context of requirements coverage the usage of input or output signals of the SUT is a useful feature. Therefore, together with every test case information about referenced signals is

Cost Efficient Test System for Embedded Systems (CETES)

embedded world 2010 conference

provided internally. Comparing used signals with the defined set of signals indicates possible malfunction or incompleteness as well.

Using a single hardware for multiple systems under tests helps to reduce costs. On the other side this requires methods of managing these hardware configurations properly. Both aspects are tackled by using the tool Gamma, already introduced on the market.

Talking about any technical development environment today often comes to the point where Eclipse is mentioned. Eclipse is on the way to become the industrial standard for tool environment.

Eclipse offers intuitive and native handling on various operating systems. It focuses on optimal usability and workflow support. A huge community worldwide is working with Eclipse.

In the meantime many stand-alone applications based on Eclipse are now offered covering aspects from embedded [2] to business [3] applications. Together with the plug-in mechanism cross sectoral issues like version management or change management are provided.

CETES integrates and combines all the above mentioned improvements in one single tool chain.

Currently there are three companies involved in the development of CETES. All involved parties are experts in different technical domains with small intersection. But one field know-how common to all companies is testing. They are all experienced service providers for specific stages of testing with test process know how in different technical domains like automotive, aerospace or medical.

The CETES partners are

- sepp.med gmbh [4]
- Kölsch & Altmann GmbH (K&A) [5]
- RST Industrie Automation GmbH [6]

All companies are members of the technology cluster BICC-Net [7], section "service provider". Within one of the related meetings the idea of combining several key features to a unique approach was born. K&A has taken the role of the product integrator.

All three companies are members of Embedded4You [8], a federation of companies providing combined support for embedded technologies. Products both software and hardware offered by members allow the cost efficient build-up of hardware-in-the-loop systems. For customers this combination of long experience in the embedded business helps to ensure the success.

Components of CETES

The test system CETES consists of three components:

- Test design
- Test management
- Test platform

In general every component can be used separately by the other components. Of course the benefit, especially for testers, is gained by using them all together and in coordination. This way CETES makes the best out of it. Seamless integration (Fig. 1) is provided by CETES which uses new interfaces designed for optimal usability.

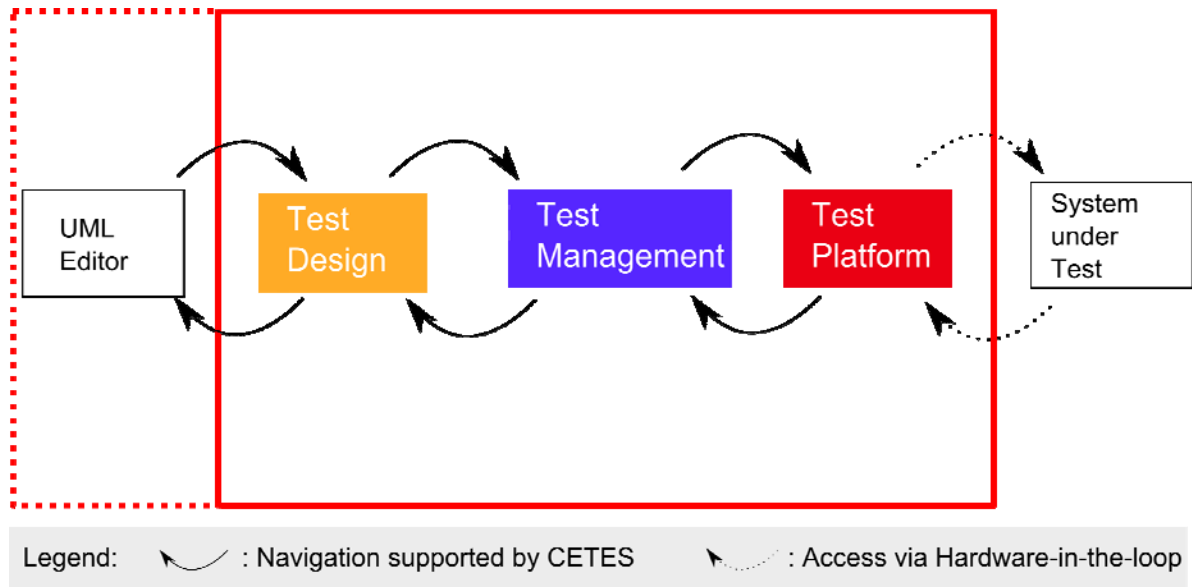


Fig. 1: CETES Components, workflows and seamless integration

Clear responsibilities for every single component are necessary and therefore contributed by the three involved companies.

2.1 Test design

Model-centric testing (.mzT [9]) is an advanced technology of sepp.med gmbh and grew from a small idea to a methodology coming along with tooling (.getmore [10]) and training. .mzT is part of various scientific [11] [12] [13] and industrial projects [14] [15] [16] [17].

2.1.1 Model-centric testing

Model-centric testing (.mzT) is an efficient methodology for design and specification in (software) testing. Applying this methodology can decisively improve the systematic nature and completeness of test coverage and test management. Model-centric testing is based on the existing model-based test design, but extends it with test management information, the mindset of the tester and adaptability to special solutions. This approach can be applied to every stage of the software test. It therefore encompasses all relevant areas between the component view, which is oriented toward complete test coverage, and the system view, which often makes an intelligent reduction necessary because of the limited number of test cases that can be executed. Instead of individual test case descriptions, test design is done using state or activity diagrams. By the use of test case generators, it is possible to automatically generate executable test cases from these models. These generators are the bridge between modeling, test management, and test automation tools. For that purpose sepp.med provides the test case generator .getmore.

The methodology can be introduced successively. In the beginning the model is used for prioritization and communication between different stakeholders. Later on automated test case generation can be added. On all stages of the test process the model provides an overview of the system under test (SUT) and provides an understandable basis for communication, even with “non-technical” participants.

Because the model is based on the test requirements AND the product requirements the quality of the requirements description is already assured during test modeling and is therefore the first step in quality assurance.

Because of the reusability and tool support, only a few elements of the model have to be adapted after a change rather than many test documents. Redundancies are therefore avoided.

.mzT is a logical further development of MBT and therefore a method for the visualization and systematization of test designs and test plans. Whereas with MBT, implementation is verified based on models from the system definition or the system design, .mzT goes one step further and adds system usage and test management aspects to the pure behavior models (Fig. 2).

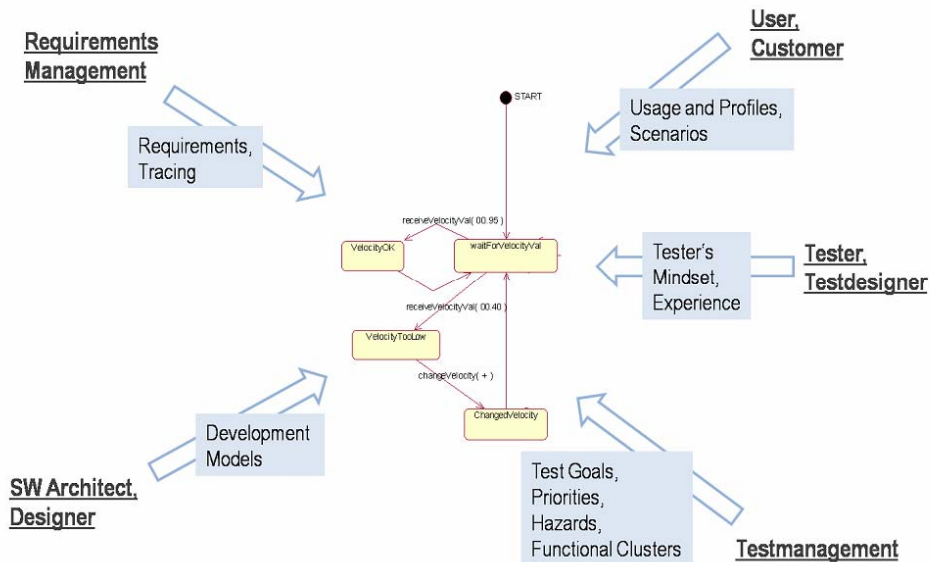


Fig. 2: Model-centric testing: In addition to possible input from development models, test modeling is based on user scenarios, test management information and necessary test cases from the viewpoint of the test (exceptional scenarios, error scenarios).

Thus .mzT emphasizes the focus on the validation: to load the system similar to the user workflow (usage) and to add exceptional situations/worst cases (tester's mindset), while including the procedures and objectives of the model-based approach. This ensures completeness of the test and therefore high test quality and test coverage with regard to the validation objectives. The integration of test management information into the model (such as priorities) and the automatic generation of test cases from the model with the test case generator .getmore also allow meaningful and test target oriented reduction of the test cases ("best test cases"), thus ensuring an economically viable test process. A schematic diagram of the .mzT process is given in Fig. 3.

The basis for the test models are standard and "real life" workflows and the tester's mindset. Manual or automated test cases can be derived from the models taking priorities, guard conditions, and requirements (all described in the model itself) into account. The generated scenarios can be executed immediately without experts having to check and detail the test cases.

The advantages of .mzT as opposed to a "traditional" document or script approach are:

- Systematic design and generation of test cases, providing controllable completeness in terms of coverage and in the reduction of test cases by test management criteria (for example, project risks and knowledge of path coverage in the case of test case reduction).
- Visualization: both developers and testers make a model of the system at least in their heads. Using .mzT both work on the same visual model, typically based on UML (Unified Modeling Language). This ensures better coordination of the test suites including coordination between stakeholders, requirements managers and developers.
- Reuse of information: Avoiding redundancy. The individual test runs no longer have to be created manually and maintained individually with great effort.

- Modularity in the models and the possibility of using different levels of abstraction by means of hierarchies in the test design, with the objective of reduced complexity of the design.
- Automation of generating test cases and of the connection to test management and test execution tools provides economic efficiency and avoidance of errors. (Errors typically occur when test attributes, such as requirements, are held redundantly in the test design and in the test management tool.)

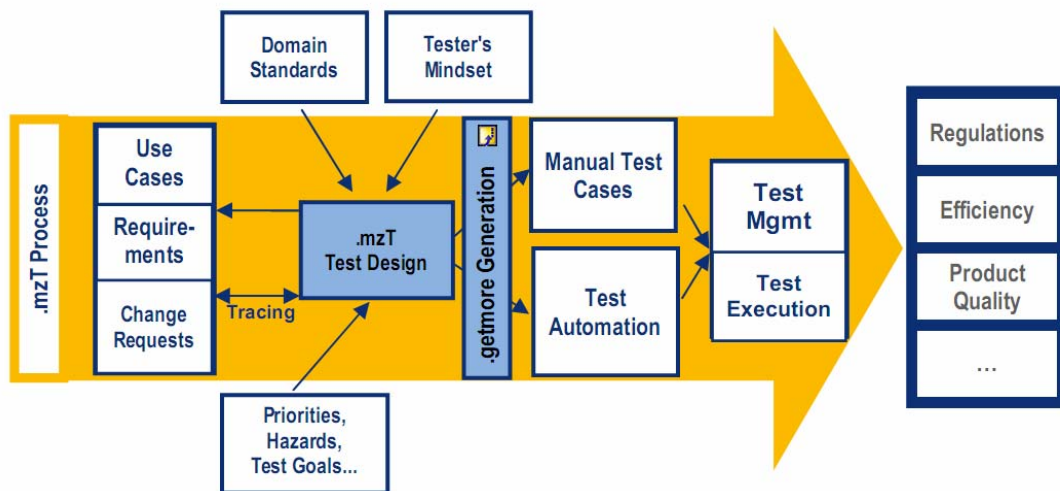


Fig. 3: .mzT integration in the test and development process

The .mzT test design is integrated consistently into the software development process. This allows the automation of process steps, resulting in higher quality and higher process efficiency, and tracing between requirement management, development and testing.

The inputs to the test design are not only product demands, such as use cases, but also requirements or change requests and the tester's mindset. In addition, usable and applicable domain standards (e.g. AUTOSAR), the experience of experts and users of the system are also included (see Fig. 3).

The test model is the central repository of the test process and contains not only the pure test case information but also attributes that are relevant to the test management or the project management (priorities, functional clusters, endangerment relevance, costs, etc.). This permits extensive planning and control of the test process from the test model and considerably improved adaptability of the test design to the requirements of the test process.

A distinction is made between three types of information:

1. Management information, such as
 - a. Priorities
 - b. Requirement IDs
 - c. Functional clusters
2. Flow control information, such as
 - a. Loop counters
 - b. GoTo's
 - c. Conditions/queries
3. Semantic information

For example, a certain feature is only supposed to be accessible if a certain license has been purchased.

It is for instance possible to generate those test cases from the .mzT model that correspond precisely to the test scope and the planned priorities for regression testing. From the requirement to the test report, .mzT and .getmore provide an automated tool chain without gaps that places the .mzT model at the centre of the test activities.

Design Paradigms: Workflow- and Architecture-Oriented

There are two different approaches to modeling .mzT test models: the workflow approach and the architecture approach.

The *workflow approach* is based on the assumption that the system under test (SUT) has features that are not in the focus of the test to be conducted and should therefore not be considered in the test model. There are workflows that describe test activities. These are restricted to just the relevant scenarios, which are then implemented in the workflow-oriented test model. The resulting models are generally less complex and organized in a more top-down fashion.

The *architecture approach*, on the other hand aims for complete feature coverage in the test. The objective is to map all possible user activities into the model in order to be able to generate all possible test cases. These models are strongly reminiscent of system behavior models and can also be verified against them.

The architecture models usually have a high level of complexity and exhibit a high degree of branching – very many test cases are generated based on these properties alone. Ideally, the two approaches should complement each other or melt into one another.

2.1.2 Test Case Generation

The test case generator .getmore (GEnerating Tests from MOdels to Reduce Effort) is the optimum tool support for model-centric testing. .getmore is being developed in coordination with the .mzT method and is being adapted to the principles of the .mzT:

- Intuitive
- Practical
- Flexible

The workflow is simple and divided into 3 steps (Fig. 4):

1. Modeling (in any UML tool)
2. Generation (in .getmore, part of CETES as well)
3. Test management and execution (CETES)

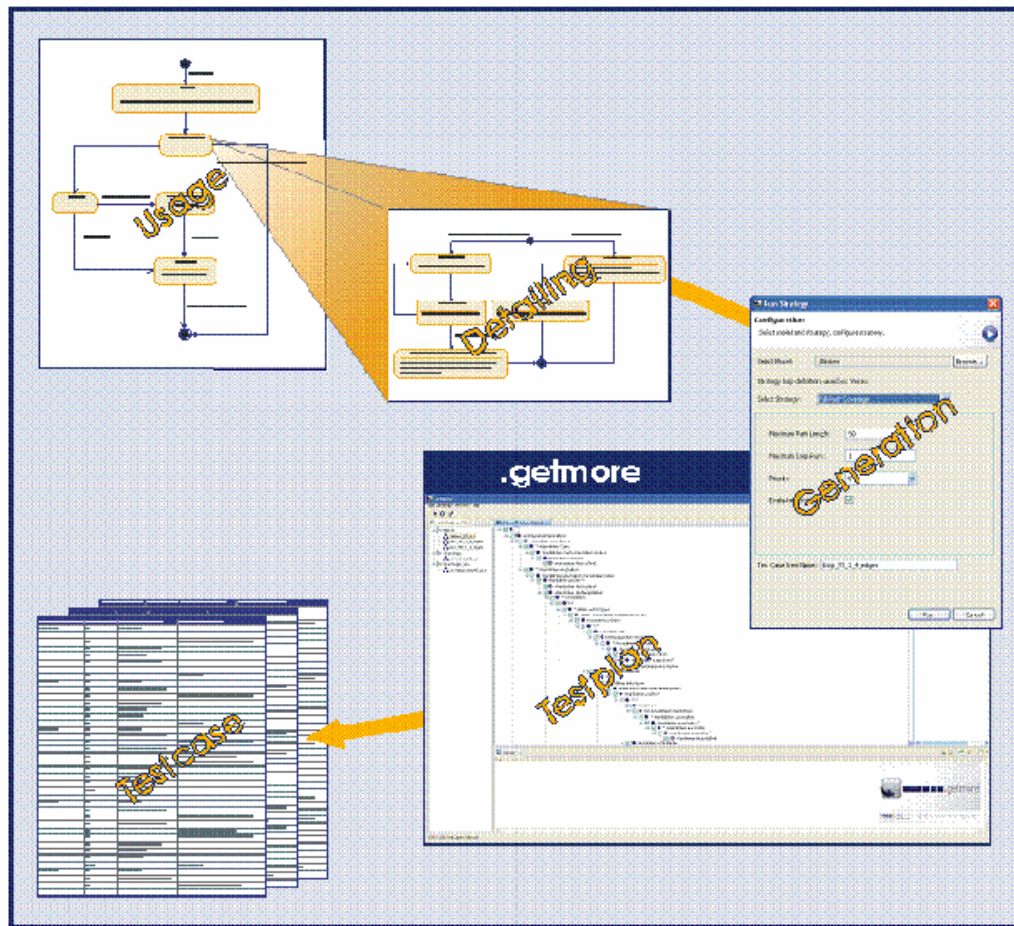


Fig. 4: Workflow from the test model via generation with .getmore to the test plan and executable

One of the strengths of .getmore is flexibility and the option of integrating the tool into existing test processes. That means full support for established tools. For modeling almost every UML-Editor like Enterprise Architect, Artisan Studio or MID Innovator can be used.

For test management and execution beside CETES there are also numerous possibilities (Test Director, Excel, TTCN-3/TTworkbench, etc.)

For test case generation many different strategies are available and wildly combinable, such as coverage strategies (Full Path, Full Edge, Full Node), guided strategies (Named Path, Guided Path) and of course random generation.

The great strength of .getmore is the capability to apply different strategies to different parts of a model and therefore to focus the test case generation on the generation part of the test process, e.g. at the end.

Because of years of experience of (software) testing we came up with the idea of "filtering" test cases. This means firstly we generate all possible test cases and later on we reduce the number of test cases systematically by the use of different strategies – we call that the "best-n" strategy. The tester can "navigate" through all possible test cases and filter the test cases by criteria like requirements, names/tags, cost or simple test case length.

Providing the methodology .mzT and the test case generator .getmore we are able to offer a test design and test cases for execution perfectly fitting to CETES. At the full stage of integration into CETES it will be possible to track back errors detected during execution (using CETES Gamma) via CETES Test Management into the Test Design (.getmore and UML-Editor).

2.2 Test management

Test management as a central discipline of testing requires different tasks to be performed. At the beginning you have to set up all kinds of resources. The way tests are performed later, the goals to be reached must be defined in advance. Technical constraints of the test environment must be considered at this early stage. Later on, during test activities, test management must ensure progress and control.

CETES as a tool chain with main focus on test execution supports mainly technical disciplines in the context of test management. Therefore managing human resources is not part of the tool chain. Despite this fact nearly all roles defined for test management [18] are stakeholders being supported by CETES.

Roles and their support by CETES:

- **Test managers** will be served with test reports showing progress and status of tests. Test goals are mainly derived by system requirements. Test designers construct tests covering those requirements. For the test manager the reports show coverage between used requirements items at the test design and those defined in the requirements database (here DOORS) at a glance. Requirement items are always subject to change in practice, seldom in theory. Comparing once referenced items during test design with actual requirement items requires a database connection.
- **Test designers** are supported during test design by additional information about available test platform process variables, related requirements items and mainly by usage of the model based test design, as described in chapter Test design.
- **Test automation experts** have to cooperate with the test designer on a strong base. Due to automatic test generation test scripts used at the test platform are the result of the underlying generation engine. This of course makes it for example easy for future adaptations to another specific test script language and for some minor changes. With this build-in concept of test generation the test automation expert can mount his work at the central place, here we talk about .getmore within CETES.
- **Test administrators** are served in different ways. First by usage of Eclipse update mechanism, deployment and maintenance of the client installation is supported. Furthermore managing the server infrastructure, the hardware-in-the-loop systems, the set up of the test platform and the operation of the test platform have a powerful base. For more details see next chapter about test platform.
- **Testers** are able to administrate, execute and report tests. That's the main role supported of the CETES test management component. Version management and bug tracking take place at test management of CETES.

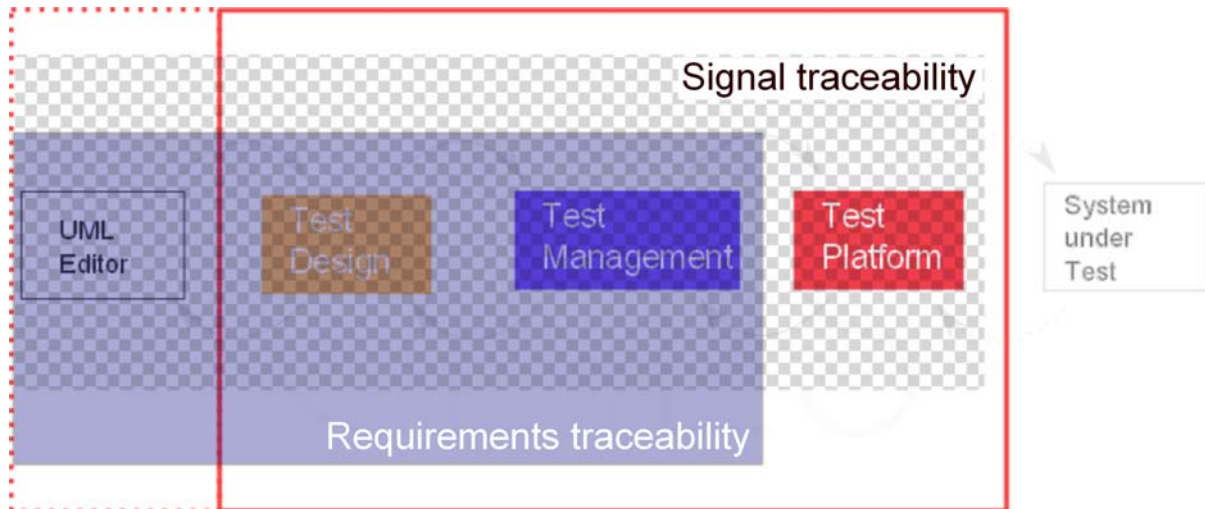


Fig. 5: Covered traceability at CETES

As a summary of the features of CETES mentioned above, the following key disciplines can be identified:

- **Test administration**

One bunch of test cases generated out of one test model used at test design can be imported into one test suite. Even without model based test design manual written test cases can also be imported.

Version management of all kinds of items like test models, test reports, test suites and other are managed by available plug-ins for Eclipse. Available plug-ins are Subversion, CVS or ClearCase. Connection to bug tracker software is provided with the usage of Eclipse.

- **Test execution**

Progressive and regression tests can be performed with hardware connected to local client or via remote access.

- **Test reporting**

Depending on the role two different kinds of reports are implemented.

One report, especially designed for testers, informs about test runs, the results in an overview and afterwards about test case specific information. The other report covers more the requests of the role test manager. Information about requirement coverage between test design and a set of reference requirements is displayed together with a comparison about used and defined process variables (signals) of the test platform (see Fig. 5). Test reports in general can be generated in the document formats HTML, PDF and RTF.

- **Test cockpit function**

The first three features of CETES mentioned above, are common to nearly every testing tool on the market. With the new approach provided with CETES the cockpit function is a synonym for seamless integration of technical testing disciplines which are mostly handled separately in other implementations.

Beginning at the import our goal is to automate this step to a one-click action. Generating the test cases out of the test model starts the importing finally into the administration part.

Inside administration and test execution users are supported to navigate directly to the corresponding test design component and test design editor. By this we solve one concept driven disadvantage of test generation: nameless, anonymous test case identification strings. This doesn't matter anymore: Just navigate to the corresponding test path inside the model.

For reports displayed inside Eclipse navigating to the model is supported as well. For example finding an undefined process variable or requirement item related to a test case is no longer a problem: again just navigate to the corresponding test path inside the model.

These features in summary improve the performance of test teams.

2.3 Test platform

CETES has a strong and trustworthy base for establishing connections to your system under test. Together with Gamma, a software package that has already proven its merits in the context of the automation industry resulted in more than 200 installations, we deliver a fast, predictable and lean method to control and monitor your connected hardware.

The central idea of Gamma is the usage of a data centric model (Fig. 6) based on process variables. Gamma contains other noteworthy abstraction features. For example, the test equipment hardware is managed by a tool that facilitates quickly and easily connecting process variables to hardware channels and automatically maintains these connections transparently at runtime (Fig. 6). Different hardware interface cards can be used for different situations / configurations without necessitating changes in the test system software. Drivers are automatically interchanged and only need to be recompiled if not already done and provided by your version management system.

The central software plug-in concept employed by Gamma allows a wide range of functions to be handled through process variables (Fig. 7). Logging, I/O writethrough, limit testing and others are directly integrated in a completely user transparent way. Simulation is supported both during development and at runtime.

Gamma already includes a library of a wide range of supported hardware interface cards out of the box, which has an additional great advantage. Using these hardware definitions complex systems can be easily configured via drag-and-drop.

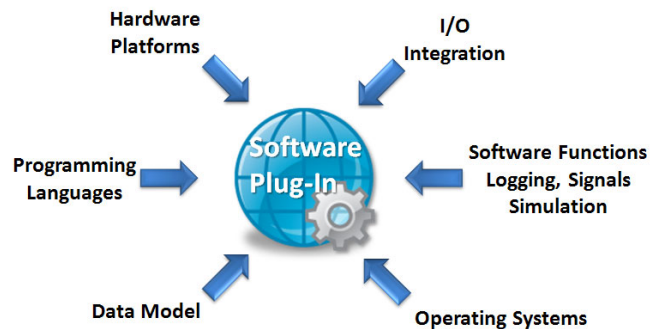


Fig. 6: Software plug-in concept of Gamma

Components out of the shelf (COTS) are based on the following protocols and card definitions: VME-Bus, CompactPCI, MicroTCA, ComExpress. Individual solutions are offered too. Gamma supports standards like CAN, Flexray, Ethercat, ProfiBus, Local IO, Ethernet, ADFX, ARINC429.

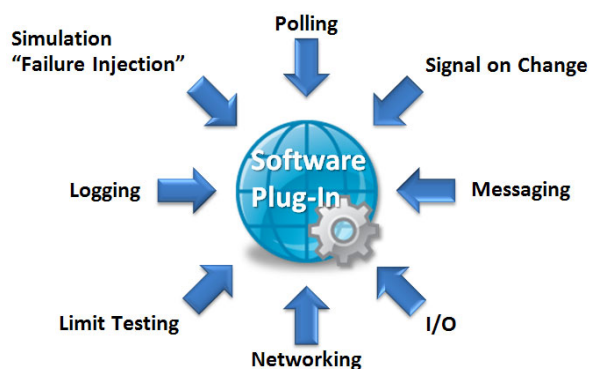


Fig. 7: Accessing the data model

A simple concept like the one presented here - combining use of process variables with the power of tool based configuration management to create a communication setup - allows us to achieve complete support for multi-thread, multi-core and network based communication. This works perfectly for networks and connected computers and without any effort spans different operating systems.

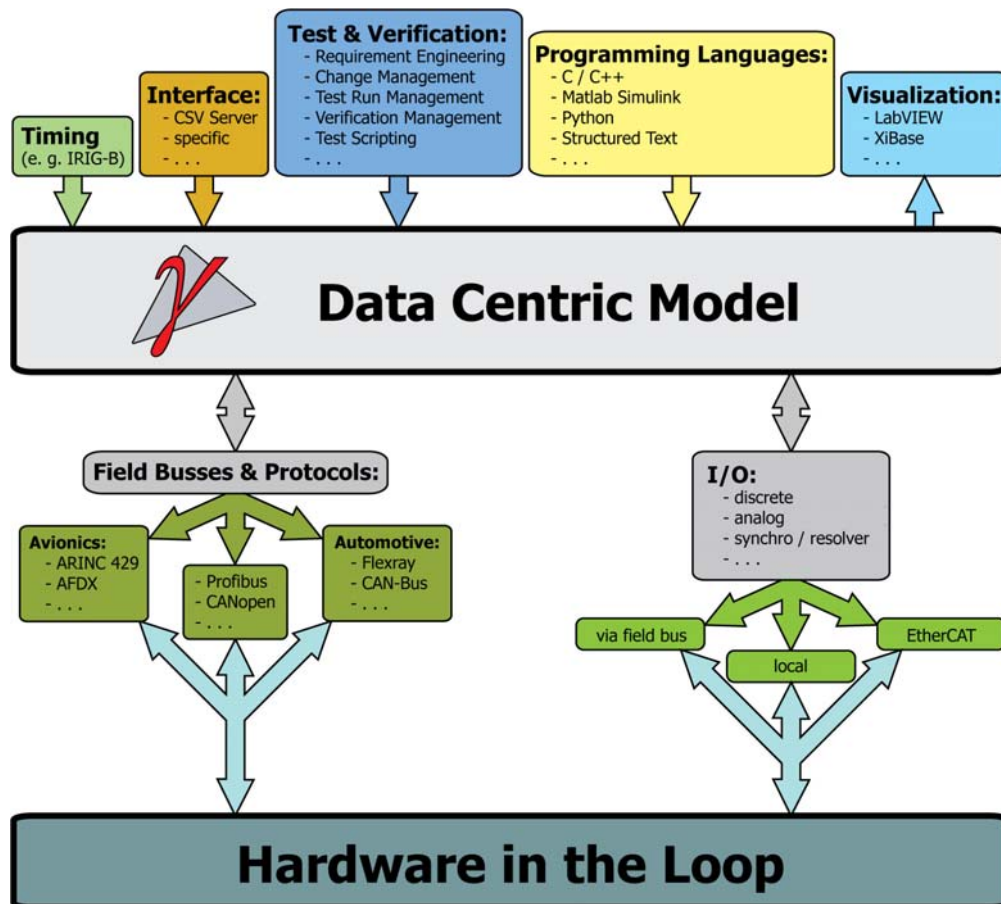


Fig. 8: Data centric model

Together with CETES a plug-in for Eclipse was developed to enable access to Gamma for administration purposes. The existing Gamma IDE for Microsoft Windows operating systems will be available for Eclipse within 2010. By this we achieve the same support as today with the more open approach and better incorporation in products like CETES. As mentioned above test management, test design and test platform are linked together. The usage of hardware access inside test design can be compared to the defined interfaces inside Gamma. Information about unused hardware process variables, components or unresolved usage conflicts are displayed at an early phase of development.

Due to its flexibility and platform independence, Gamma can be employed for an almost limitless number of purposes - be it as an advanced automation application, as a basis for elaborate testing systems or even in the beamline control of the current generation of particle accelerators. Its support for compact embedded components and diverse field buses such as ADFX, ARINC429, CAN or Flexray makes it ideally suitable for avionics and automotive applications. Gamma is also used in highly sensitive medical technology systems, e. g. in radiation therapy.

3 Conclusion and Outlook

Putting all aspects together we offer a test system using COTS elements combined with test experience of over 30 years in various industries and at least three advantages for every project:

1. Test suite visualization & automatic test case generation
2. Adoptable and configurable HW-test-system
3. Complete test process using affordable components and a closed tool chain for the design, generating, management and execution

The new approach presented in this paper combines the experiences and know-how of three companies in a way which can significantly improve your testing efforts – by lower costs.

If you ask yourself why we use the tail of the whale at our product branding you have to go far back to ancient times. Here the word “cetes” has a Latin origin and corresponds to the plural of whale. That’s the whole story.

For further information about CETES please visit www.cetes.eu.

Acknowledgements

The authors would like to thank Dr. Werner Altmann, CEO of K&A for contributing to this paper.

About the authors

Jürgen Hartung

The profession of Jürgen Hartung bases on completed academic studies of both electrical engineering (FH Augsburg) and computer science (FHTE Esslingen).

After his degrees he started to work for K&A in the area of automotive industry.

Since 2002 he is responsible for the automotive business department and since 2008 he is the head of the Realtime & Technology Transfer Department of K&A. His interests are model based software methods and tools for the design of electronic control units for vehicles.

Florian Prester

From 1998 to 2004 Florian Prester studied Computational Engineering at the University of Erlangen-Nuremberg. After his degree he worked as a research assistant at the Regionales RechenZentrum Erlangen-Nuremberg (RRZE) with special interest on network security and abstraction mechanism.

Since 2007 he is working for the sepp.med gmbh, since 2009 he is CEO responsible for the automotive, automation & systems industry.

His interests are test conception, modeling and test improvement.

Robert Schachner

Dipl.-Ing. (FH) Robert Schachner has been active in the embedded market since 1985. After starting at Motorola (in the department that developed the VME Bus), he has been working with his own company RST Industrie Automation GmbH since 1987. Having worked on numerous projects in almost every branch of the automation industry, he has developed a profound knowledge of both the technology and the market. Mr. Schachner is also an active member of the trade organization Embedded4You e.V. where he currently fills the role of press spokesman.

References

- 1 www.autosar.org
- 2 Eclipse CDT, C/C++ Development Tool, www.eclipse.org/CDT
- 3 Eclipse BIRT, Business Intelligence and Reporting Tools Project, www.eclipse.org/BIRT
- 4 www.seppmed.de, Test design with .mzT .getmore
- 5 www.koelsch-altmann.de, Test management with CETES
- 6 www.rst-automation.de , Test platform with Gamma
- 7 bicc-net.de, Bavarian Information and Communication Technology Cluster
- 8 Embedded4You, www.embedded4you.com
- 9 <http://www.seppmed.de/modellzentrierterTest.187.0.html>
- 10 <http://www.seppmed.de/getmore.65.0.html>
- 11 G. Götz, A. Metzger; "Challenges and Solutions for Test Design and Management for Next Generation Medical IT Testbed", Conquest 2008 (Potsdam), ISBN 078-3-89864-567-6
- 12 F.Prester, W. Dulz, "markov@.mzT - .model centric Testing (.mzT) using markov chain theory", EW2009 (Nürnberg), ISBN 978-3-7723-3798-7
- 13 K.-H. Kühnlein, G. Götz, A. Metzger; M. Seel, "Experiences with Model Centric Testing in Standard Based Medical IT Environments", Conquest 2009 (Nürnberg), ISBN 978-3-89864-637-6
- 14 <http://www.seppmed.de/TestNGMOST.230.0.html>
- 15 www.testngmed.de
- 16 M. Beisser-Dresel, Beitrag Kongressband: "Murphy`s Law und Softwarequalitätssicherung", Embedded Software Engineering Kongress 2008 (Sindelfingen)
- 17 G. Kiffe, F. Prester, S. Siegl, M. BeisserM. Seel,"Model-Driven Test Case Generation in HIL Testing: Usage Models and Model-Centric Testing .mzT" ATZelektronik 05/2009
- 18 Andreas Spillner und Tilo Linz, Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard, Dpunkt Verlag, 3. Auflage, ISBN-10: 3898643581