A Functional Requirement Traceability Management Methodology for Model-based Testing Framework of Automotive Embedded System

Kabsu Han, Jiae Youn, Jeonghun Cho
School of Electronics
Kyungpook National University
Daegu, Republic of Korea
{kabus, jiae0620}@knu.ac.kr, jcho@ee.knu.ac.kr

***Abstract*— We present an automated functional requirement traceability generation and management methodology for model-based testing framework. Traceability of software was recognized in 1960s and international standard was established in 1980s. In automotive industry, lots of researches for the requirement traceability are performed but not practical for testing. This paper presents traceability fundamental and practical case study for model based testing of automotive embedded system that includes generation of the functional requirement traceability.**

Keywords - Model-based testing; Rrequirement management; Test automation; Traceability; Functional requirement ;

I. INTRODUCTION

The traceability was pointed as an issue of interest in software engineering and recognized to discuss the problem of software engineering in 1968 [1]. In 1980s, traceability was founded as a requirement in lots of national and international standards for software and system development. In automotive industry, automotive embedded systems increase steadily as the requirements and functionalities increase. Furthermore lots of companies, such as OEM, suppliers, are involved in developing the automotive embedded system. Although model-based development and testing are widely used [5][6], the requirements and traceability of automotive embedded system cannot be managed easily. This paper introduces the concept of required traceability for model-based testing and proposes practical framework that include bidirectional traceability among requirements, models and test cases. Also, practical requirements tracing with commercial tools are described.

Section 2 describes entire model-based testing process. Section 3 and Section 4 describe background knowledge about requirement engineering and traceability with standard and COTS tools. Section 5 shows case study for model-based testing of automotive embedded system. Finally, Section 6 describes conclusion.

II. MODEL-BASED TESTING

In model-based testing (MBT), the test developer simply describes a functional model of the system under test (SUT). A test sequence generation algorithm that can be selected by hand in the test case generator creates test cases to verify and validate the functional model of the SUT. A test case generator creates test cases that can run on the SUT from the functional test cases. After that, a test automation tool executes the test cases on the SUT automatically. Reports that compare each output from the SUT and the expected results are generated automatically. Test coverage and reliability of the test depend on the model of the SUT and the test sequence generation algorithm; even test cases can be generated manually and automatically. Figure 1 shows the entire process of MBT [2].

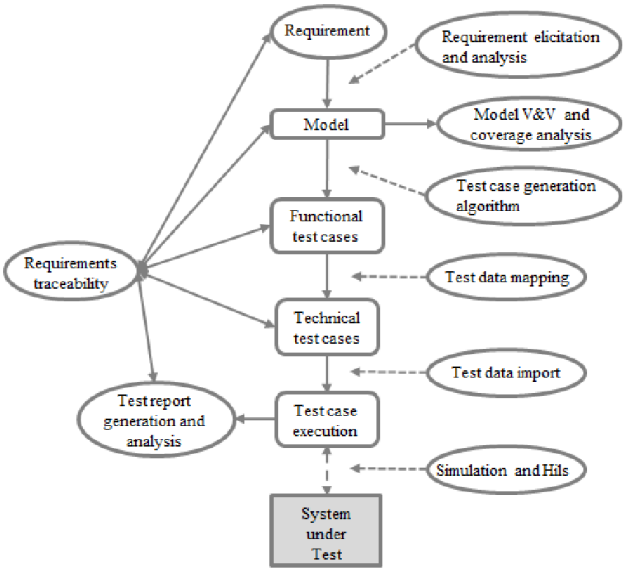


Figure 1 Model-based testing process

The MBT method requires more steps and tools than the manual testing method, such as modeling, test case generation, and test case execution. Making a model of the SUT is describing a functional model of the system that needs to be tested. The modeling has to focus on the functional requirements of the system that the test developer wants to test. The model of the SUT may omit a lot of the details of the SUT that are not related to the testing. After describing the model, it has to be verified and validated for MBT. Most modeling tools provide automated verification

 ISBN: 978-1-61208-348-3

and validation tools. Also, a graphical verifier is very useful to easily check the model.

The next step is generation of functional test cases from the model. The test developer has to decide the test selection criteria in order to generate efficient test cases. Because infinite numbers of test case are available, a plan to test all cases is impractical. Through selection criteria, coverage of the test cases is decided, and functional test cases that are test sequences of the model are generated. Figure 2 shows a transition based test coverage of black-box testing. The functional test cases are a kind of simple view of the SUT, so they do not contain detailed information to execute test cases directly on the SUT.

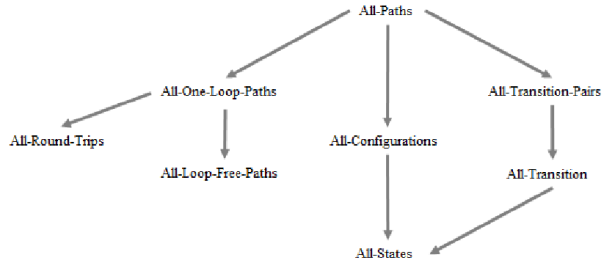


Figure 2 Transition based test coverage

The generation of an executable test case, called a test script, is required to execute the generated test cases on the SUT. The adaptation and transformation approach can execute test cases on the SUT. The test case generation tools have to fill in detailed information of a low-level SUT that are not described in the functional model.

One of the benefits of MBT is independence between test cases and test environment. By regeneration of executable test cases, the same set of test cases that includes the models can be reused in different test environments.

III. REQUIREMENT ENGINEERING

The requirement engineering phase is the first step of model-based testing. The requirement engineering process can be divided into 6 processes like below [1][3][5].

- Requirement elicitation
- Requirement analysis
- Requirement specification
- System modeling
- Requirement validation
- Requirement management

During early phase of the requirement engineering, user requirements are elicited and analyzed. The requirement elicitation is about the understanding the problems to solve. Because user requirements can be conflicting among them, requirment engineer have to make decisions to elicit and analyze the requirements that have to be specified. After the problems to solve are understood and analyzed, they have to be described for the requirement specification. The requirement specification has to describe the product to be developed not the process. In automotive industry, some certification standards, such as IEC 61508 and ISO26262 for the product, are proposed. To specify requirements, lots of techniques can be used, such as informal and formal description. In model-based testing, system modeling will be described with appropriate modeling language, such as FSM, MSC and UML, according to the requirement specification. After that, the requirement specification can be verified and validated through the system modeling. Depending on the modeling language, lots of verfication and validation method can be used, such as simulation and formal verification. Also, the requirement specification has to be managed during the entire project. These requirements consist of functional things that have to be provided and non-functional thing such as performance, reliability, cost. Throughout in this paper, the functional requirements are considered and the requirement management tool is used to manage the requirements.

In many cases, requirements are elicited as documents format, such as MS word and excel. But these cannot be used for requirement specification and requirement management tool directly. Also, the requirement specification in requirement management tools cannot be exchanged easily. To solve this problem, automotive industry proposed requirement exchange format, called Rule Interchange Format (RIF) [7]. The new name Requirement Interchange Format (ReqIF) was introduced by OMG in 2011 [8]. RIF/ReqIF is an XML file format that can exchange the requirements between requirement management tools from different vendors. Also, the requirement exchange format defines a process to transform the requirements between partners. EAST-ADL2, a kind of European architecture description language, proposed a RIF importer/exporter extension already. IBM DOORS, the requirement management tool, supports RIF/ReqIF importer and exporter and MS documents importer/exporter. Also, the Requirement Modeling Framework (RMF), open-source-framework with requirements, supports ReqIF standard [9].

IV. TRACEABILITY

In a software and system engineering area, the trace can be defined like below [1].

- *A specified triplet of element comprising: a source, a target and a trace link which connecting a source and a target. When more than a source and a target are associated by a trace link, such as a sub-pair of a source and a target, the sub-pair are treated as a single source or a target.*
- *The action of following a trace link from a source to target.*

The trace can either be atomic or chained. The traceability is the potential ability for traces. To assure the traceability, each of the sources, targets and trace links have to be acquired and stored. After that, software and system

engineering activities and task can be traced as shown in Figure 3. The traces exist within specific development and maintenance life cycles. Also, the trace can be reused in different life cycles. The requirement traceability is the ability to describe and follow the requirement lifecycle in forwards and backwards direction. The tracing is the activity of either establishing or using traces. The tracing can be divided into 3 types, manual, automated and semi-automated.

- *Manual tracing – traceability is established by human tracer. Traceability creation and maintenance with drag and drop user interfaces are used in requiremnt management tools commonly.*
- *Automated tracing – traceability is established via automated tools and methods. Typically, traceability creation and trace link maintenance are automated.*
- *Semi-automated tracing – traceability is established via combination of automated tools and human activities. For example, automated tools sugguest candidate trace links and human tracer verify them.*

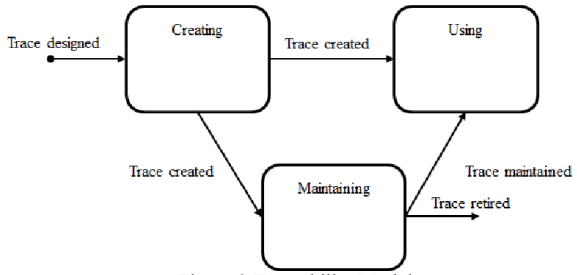


Figure 3 Traceability Model

In model-based testing, lots of traceability links are required like below [3].

- *traceability between requirements*
- *traceability between requirements and system model*
- *traceability between requirements and test cases*
- *traceability between requirements and test reports*

The traceability between requirements can be supported by the requirement document tools and the requirement management tools. In case of MS documents, MS office XML format are XML-based document formats and XML schema introduced in Office 2007. MS word and MS excel documents can import from and export to XML format. IBM DOORS can import from MS document and export to MS document. If importer and exporter between tools are not supported directly, RIF/ReqIF can be used to exchange the requirements, such as Papyrus MDT and plug-in. Figure 4 shows exporter of MS word and IBM DOORS.

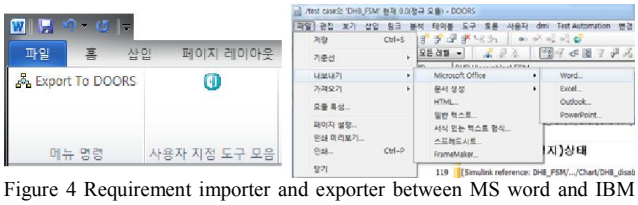


Figure 4 Requirement importer and exporter between MS word and IBM DOORS

The traceability between requirements and system model also can be supported the requirement management tools and modeling tools. Mathwork MATLAB/SIMULINK with verification and validation toolbox supports traceability link to MS word, MS excel and IBM DOORS. This toolbox can generate multiple traceability links with MS word bookmark, MS excel cell and DOORS object semi-automatically. When traceability links are generated, MS documents and DOORS objects are indicated with MATLAB/SIMULINK icon. Figure 5 shows traceability links on Stateflow model and Figure 6 shows Traceability links on MS documents and DOORS objects.

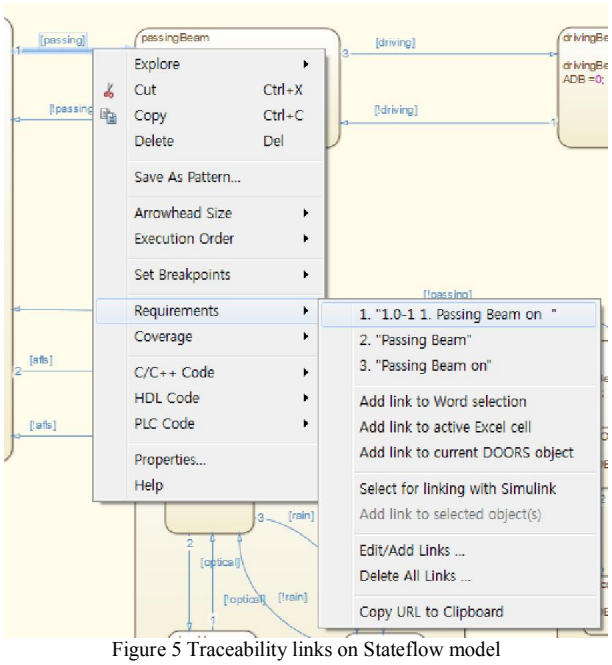


Figure 5 Traceability links on Stateflow model

In case of automotive embedded system, to execute the test cases that are generated from the functional requirements with System under test (SUT), I/O ports and in-vehicle network (IVN) interfaces are required [4]. Appropriate commercial-off-the-shell (COTS) tools as a de-facto in the automotive industry, such as Vector CANoe and dSPACE microautoboxII, can be used to execute test cases. Vector Test Automation Editor can generate executable test cases with XML format and supports traceability between requirements and test cases/test reports. The generated test cases can be executed on Vector CANoe through IVN. Depending on the DOORS objects, test groups and test cases are generated automatically in XML test module. The

 ISBN: 978-1-61208-348-3

title of test groups and test cases are object id of DOORS. The test descriptions are imported from DOORS and external reference to DOORS are generated automatically.

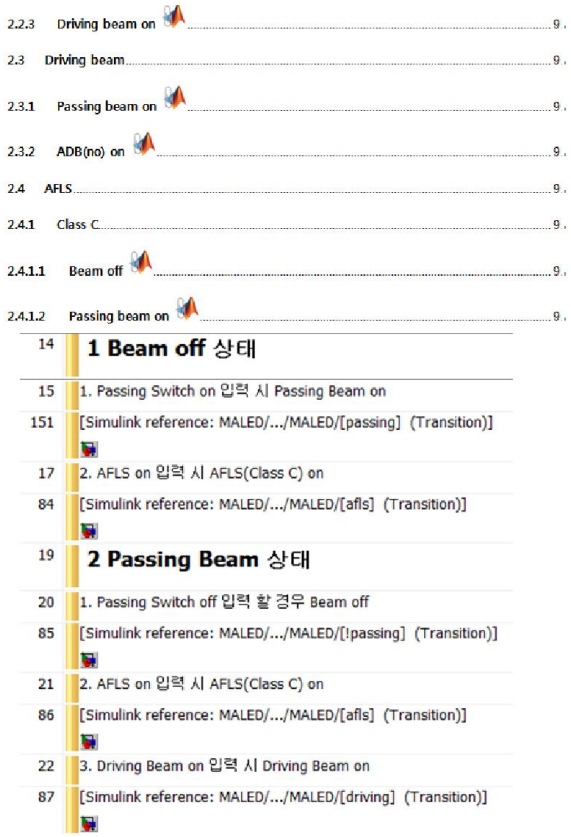


Figure 6 Traceability links on MS documents and DOORS objects

Test reports can be exported to DOORS through test report data mapping and import test report data. Also, test reports contain external reference to DOORS for traceability between requirements and test reports. Figure 7 shows traceability between DOORS and TAE as traceability between requirements and test cases. Figure 8 shows generated test groups and test cases include description and external references.

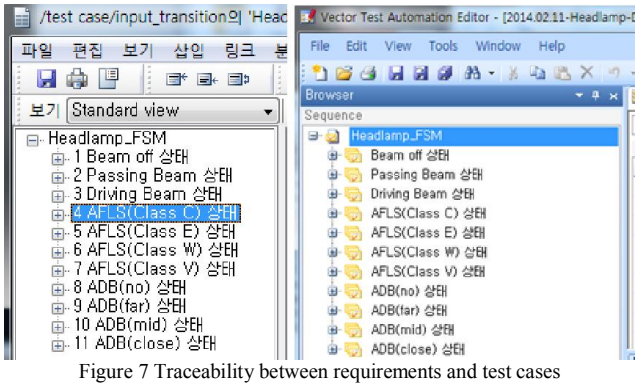


Figure 7 Traceability between requirements and test cases

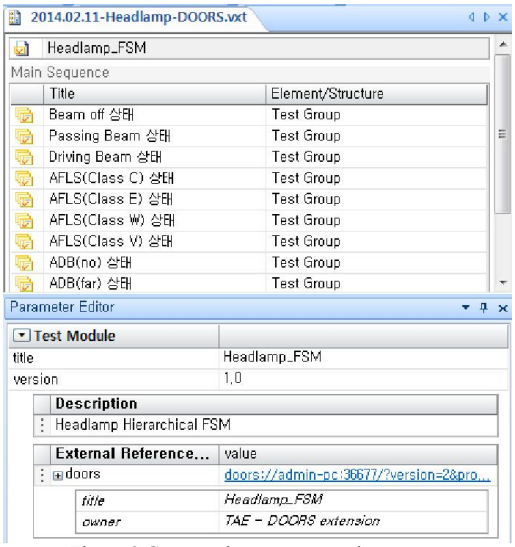


Figure 8 Generated test groups and test cases

V. CASE STUDY

To create and manage functional requirement traceability of model-based testing framework, intelligent headlamp system that includes adaptive front lighting system (AFLS) and adaptive driving beam (ADB) is adopted. The functional requirements elicited from a part of vehicle regulation of UNECE, such as R.48 and R.123, and requirements of OEM. The target system is an ECU of intelligent headlamp system. The main functional requirements of intelligent headlamp system consist of passing beam, AFLS, driving beam and ADB. The functional requirements of AFLS consist of class C, class E, class V and class W that elicited from the regulation of UNECE. The functional requirements of ADB are elicited from OEM. Figure 9 shows the functional requirements of AFLS and Figure 10 shows the functional requirements of ADB.

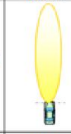
Mode			Driving beam	ADB	AFLS			
					Class C	Class E	Class V	Class W
Function			High Beam	Anti-glare High Beam	Default Low Beam	High speed Low Beam	Town Low Beam	Rainy Low Beam
Environmental Condition								
Op	Switch		High	Auto-High	Auto-Low	Auto-Low	Auto-Low	Auto-Low
	Speed (km/h)	On	-	40~	20~100	100~	0~30	Rainy
		Off	-	30~	25~95	95~	0~25	

Figure 9 Functional requirements of AFLS

The ECU of intelligent headlamp system receives environmental information, such as vehicle speed, illumination and other vehicle, from other ECUs and controls the headlamps of vehicle. At the first phase, the functional requirements are elicited from informal documents that contain functional and non-functional requirements for ECU, R.48 and R.123 of UNECE, and described in MS word. The

 ISBN: 978-1-61208-348-3

functional requirements in MS word are exported to DOORS for requirement management.

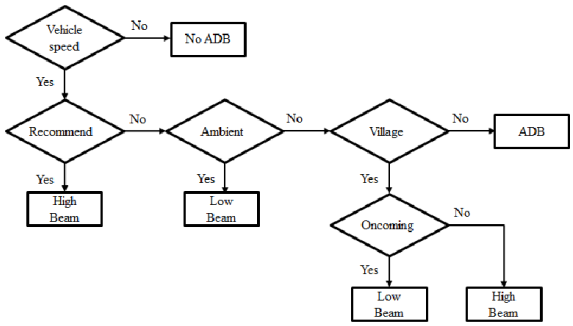


Figure 10 Functional requirements of ADB

During this phase, 117 functional requirements for headlamp system and 60 functional requirements for ADB are generated as DOORS objects. After that, the functional requirements in MS word are described in MS excel to generate a functional model. Because the functional requirements of ECU can be modeled as a discrete system, Stateflow are used to generate the functional model. Transition table function in Stateflow can generate the functional model with tabular description automatically, shown as Figure 11.

When the traceability links between the functional model and requirement are generated, the functional requirements verification and validation can be done through the functional model. If any inconsistency and corruption exist in the functional model, model analyzer will find it. SIMULINK design verifier analyzes the function model and generates test cases for structural coverage, such as condition, decision and MC/DC. During this phase, 167 test cases are founded and 48 test cases are generated. Figure 12 shows the report of SIMULINK design verifier.

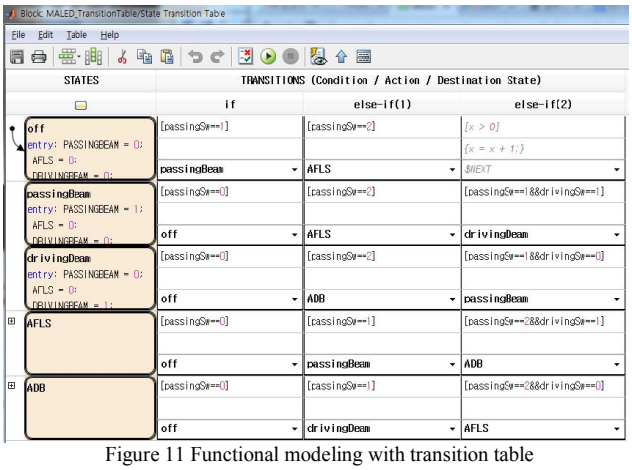


Figure 11 Functional modeling with transition table

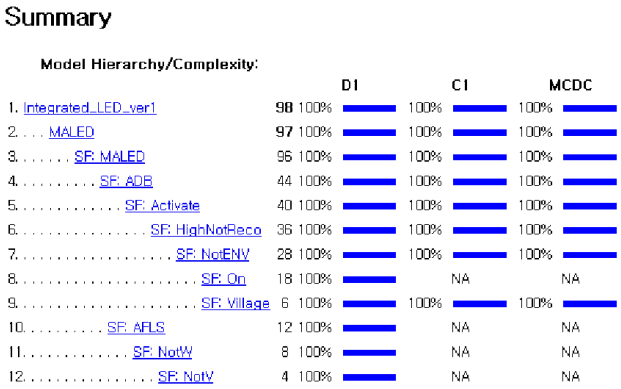


Figure 12 Validation result of the functional model

When the validation of the functional requirements through the functional model is finished, test cases can be generated from the functional requirements. Through the DOORS interface, XML test modules can be generated and associated automatically. Since the title of each test case is an object ID of DOORS module, traceability between requirements and test cases can be managed easily. Vector TAE is used to edit the XML test modules and Vector CANoe is used to execute the XML test modules. Because generated XML test modules contain test sequence, description and external reference to DOORS only, test engineer have to develop each test case according to functional requirements. During this phase, states and transitions in the functional model are mapped to technical signals in technical model. Depending on the technical model, various signal format, such as CAN, LIN and FlexRay, can be used. In this case study, headlamp ECU is connected with other ECU through CAN network. 12 messages with 49 signals are in CAN database file and 30 environment variables are developed to controls the CAN message and test environment. Figure 13 shows test case generation with DOORS-TAE interface and Figure 14 shows developed test cases with technical signals.

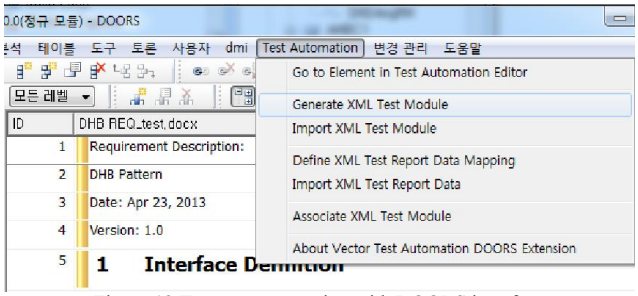


Figure 13 Test case generation with DOORS interface

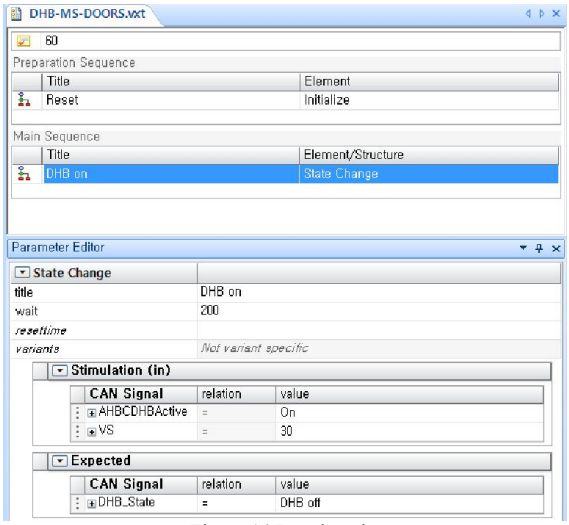


Figure 14 Developed test cases

When test cases are developed, each test case can be executed on the Vector CANoe with SUT. If real SUT is not available yet, simulation model can replace the real SUT as well as other ECUs. In the test environment, 9 ECUs are simulated that are not available in the LAB., such as Transmission control Unit (TCU), Engine Management System (EMS) and camera module, and a prototype and simulation model of ECU of intelligent headlamp are used to test. After execution of test cases, test report of the test cases that includes test verdict and traceability links are generated automatically. Also, the test report contains detail test step with time stamp and statistics. Figure 15 shows a part of test report that includes timestamp, test step, verdict and traceability link. Also, DOORS can import XML test report data through DOORS interface. With the test report, traceability between functional requirements and test reports can be established and managed.

Main Part of Test Case

Timestamp	Test Step	
1. DHB on: Passed		
0.806167		Successfully set CAN signal 'AHBCDHBActive': 1 (
0.806167		Successfully set CAN signal 'VS': 30
0.806167	1	Stimulation of the input parameters
1.006167	Resume reason	Elapsed time=200ms (max=200ms)
1.006167	2	Waited for 200 ms.
1.006167		Check CAN signal 'DHB_State' value passed: = 0
1.006167	3	Validation of the expected parameters

1.1.2 61: Passed

Data type: DHB (Input signal)

Test case begin: 2014-02-17 16:55:16 (logging timestamp 1.006167)
Test case end: 2014-02-17 16:55:17 (logging timestamp 1.306167)

DOORS Link (managed by: TAE - DOORS extension): 61

Figure 15 Test report

VI. CONCLUSION

To create and manage functional requirement traceability for model-based testing framework of automotive embedded system, automated and semi-automated tracing is considered. Bidirectional traceability between functional requirements, MS documents and IBM DOORS, are created through IBM DOORS interface. Also, traceability between functional requirements and functional model and traceability between requirements and test cases are created through COTS tools, such as MATLAB SIMULINK and Vector CANoe, for practical requirement tracing. The case study shows discrete system only but applicable to continuous system. Automated tracing for model-based testing framework is very helpful to verify and validate automotive embedded system.

ACKNOWLEDGMENT

This research was supported by the MSIP(Ministry of Science, ICT & Future Planning), Korea, under the C-ITRC(Convergence Information Technology Research Center) support program (NIPA-2013-H0401-13-1005) supervised by the NIPA(National IT Industry Promotion Agency.)

REFERENCES

[1] J. Huang, O. Gotel, and A. Zisman, Software and Systems traceability. Springer-Verlag, London, 2012.

[2] M. Utting and B. Legeard, Practical model-based testing, 1st ed., vol. 1. Elsevier: San Francisco, pp.19–35, 2007,

[3] M. Adedjouma, H. Dubois, and F. Terrier, “Requirements exchange:from specification documents to models” The 16th International Conference on Engineering of Complex Computer System (ICECCS 2011) IEEE, April, 2011, 27-29, pp. 350-354, ISBN:978-1-61284-853-2.

[4] K. Han, I. Son, and J. Cho, “A study on test automation of IVN of intelligent vehicle using model-based testing” The Fifth International Conference on Ubiquitous and Future Networks (ICUFN 2013) IEEE, July, 2013, 2-5, pp. 123–128, ISSN:2165-8528, doi:10.1109/ICUFN.2013.6614794.

[5] R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. Akbar Raja, and K. Kamran, “Requirement traceability: A systematic review and industry case study” Int. J. Soft. Eng. Knowl. Eng., May, 2012, vol. 22, pp. 1-49, ISSN:0218-1940, doi: 10.1142/S02181940120 05846.

[6] M. Weber and J. Weisbrod, “Requirement engineering in automotive development-experiences and challenges” IEEE Joint International Conference on Requirement engineering, 2002, Sep, 9-13, pp. 331-340, ISSN:1090-705X, doi: 10.1109/ICRE.2002.1048546.

[7] World Wide Web Consortium. *RIF Overview*. [Online]. Available from: http://www.w3.org/TR/2010/NOTE-rif-overview-20100 622/ 2014. 05. 07

[8] Object Management Group. *Requirement Interchange Format*.[Online]. Available from: http://www.omg.org/spec/ReqIF/ 2014.05.07

[9] Eclipse Incubation. *Requirement Modeling Framework*. [Online] Available from: http://www.eclipse.org/rmf/ 2013.05.07