Model-Based Testing Traceability

Mark R. Blackburn, Robert D. Busser, Aaron M. Nauman, Travis R. Morgan
T-VEC Technologies, Herndon, VA

ABSTRACT

This paper discusses requirements for model-based testing tools that are important for adoption by organizations that develop and test large-scale critical applications. It discusses the importance for the tools to provide full life cycle support starting from requirement-to-test traceability from requirement management tools, through requirement and design modeling, to model-based test generation, automated test execution and analysis, to test coverage analysis. The paper provides an example of requirement-to-test traceability tool support and discusses the benefits, which include faster test failure analysis, better assessment of requirement-to-test completeness, and critical support for project measurement and management. A second example illustrates requirement-to-model defect traceability.

Keywords

Model-based testing, automatic test generation, requirement and design models, model checking, requirement-to-test traceability.

1 INTRODUCTION

The users of new tools and technology in a research community are far more tolerant of gaps, holes, and bugs than line engineers and their management that must produce application releases on tight schedules and constrained budgets. Over the past several years, we have had the opportunity to work with different organizations, in various application domains, and been involved in the transition and adoption process of model-based testing into their organization. There are many requirements on the organization, users, and developers of the tools that appear almost mandatory as part of an effective adoption process.

The completeness of the model-based testing environment is a critical element for many organizations. Although it is often possible to test a subset of an application using model-based testing tools, organizations often resist adoption if a relatively complete approach is not available during pilot project trials. The modeling techniques and languages must be relevant to the applications under test (e.g., embedded systems with complex math, avionics, command and control, language processing, client-server), and support automated test execution against various languages in different environments. The completeness of the modeling language for specifying the behavior of the target application as well as the ease of use of the model representation technique is often critical too. The learning curve must be relatively short, usually fewer than three months, but with a rapid feasibility demonstration, usually within three days.

Many organizations that have adopted model based testing have mature processes and use some form of requirement management process or tool to organize and manage requirements that come from customers and other internal and external stakeholders. Requirements often come in the form of interface control documents, with separate functional requirements that often are not contained in one single document or specification. Integration of requirement management with model-based testing tools provides a better way to link the models to the requirements, and when combined with automatic test generation provide better requirement-to-test traceability, support for assessment of the completeness of the requirements as well as the modeling process, measurement for project management, and failure and fault analysis.

1.1 Requirements

The following is a non-exhaustive list of requirements for a model-based testing environment and process that is often demanded of organizations that develop business-critical software-intensive applications:

1. Automated and comprehensive test generation and with test execution support for most any environment and language
2. Expressive modeling language that is easy-to-use, but scalable to large and complex applications supporting multi-person development teams
3. Integration into the full development life cycle, providing support from requirement-to-test traceability through configuration management
4. Supports project management and measurement critical to the development of on-time product delivery
5. Provides return on investment (ROI) in short timeframe

1.2 Organization of Paper

This paper discusses requirements and related implications that have evolved out of the use of model-based testing tools, and describes fundamental aspects underlying end-to-end requirement-to-test traceability, and key benefits that can be derived from it that can be a driver for organizational adoption. A brief example is provided to illustrate the full life cycle support through an integrated set of tools. Lastly, model checking and the associated benefits related to requirement-to-model traceability are discussed at the end of the paper.

2 MODEL BASED TESTING CONTEXT

Automated test generation is the enabling technology for model-based testing. One of the earliest approaches to test automation transforms the model into Disjunctive Normal Form (DNF) and a partition of the input domain is formed from the preconditions of the disjuncts. A disjunct is a logically AND'ed set of Boolean-value condition. Test cases are drawn from each subdomain of the partition [1].

As shown in Figure 1, we have developed model-based testing tools that transform models characterizing requirement, design

and application properties (e.g., safety), based on representations such as decision tables, state machines, control system, and code, into a hierarchical DNF-like form [2]. The modeling languages support functions or other forms of model references that are required to scale to large and complex applications, with multi-person development teams. The underlying modeling language provides support for an extensive set of mathematical operators (e.g., trigonometric, intrinsic, integrators, quantization, matrix) that extend standard arithmetic operators to specify functional behavior supporting various applications domains. Finally, the test vector generator integrates with a test driver generator to produce test drivers that automate test execution for most any language and test environment with automated test results analysis.

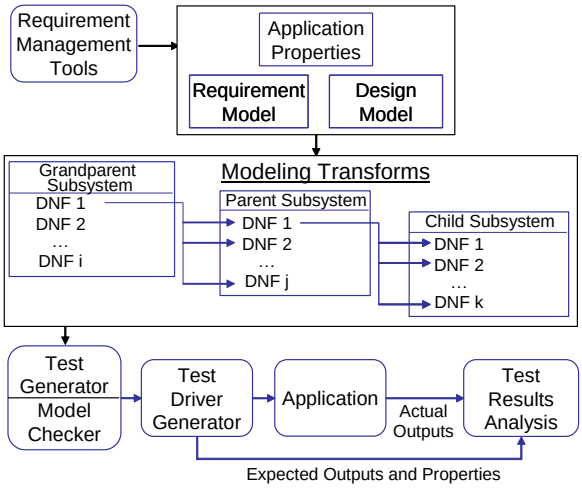

Figure 1. Model Based Generation Flow

3 LIFE CYCLE SUPPORT

The integrated environment generically referred to as the Test Automation Framework (TAF) integrates government and commercially available model development and test generation tools. One of the latest additions to TAF integrates the DOORS® requirement management tool with the T-VEC Tabular Modeler (TTM) that supports the Software Cost Reduction (SCR) method [3] for requirement modeling. DOORS integrates also with Simulink®, which supports design-based models, and TAF integrates requirement models with design models to provide full traceability from the requirements source to the generated tests, as reflected in Figure 2.

3.1 Example Requirements

The following vertical tracker example is simplified from a requirement of the Traffic and Collision Avoidance System (TCAS). A vertical tracker would track another aircraft relative to one's current altitude and must maintain the tracking state. Figure 3 shows an image of the vertical tracker requirements that were entered into DOORS. Each statement of the requirement is associated with a DOORS ID (ID). Some of the requirements used in the traceability discussion are related to the vertical tracking state as defined by the following IDs:

1. vt_3: another aircraft is considered to be in "in the altitude window" if it is within 2700 feet above or below own aircraft
2. vt_6: the vertical tracking for own aircraft shall be in TRACKING state if own aircraft is at or above 10,000 feet in altitude, but no other aircraft is in the altitude window.
3. vt_7: the vertical tracking for the own aircraft shall be in ADVISORY state if the own aircraft is at or above 10,000 feet in altitude, and other aircraft in the altitude window.
4. vt_8: the vertical tracking for the own aircraft shall be in NOT_TRACKING state if the own aircraft is less than 10,000 feet in altitude.

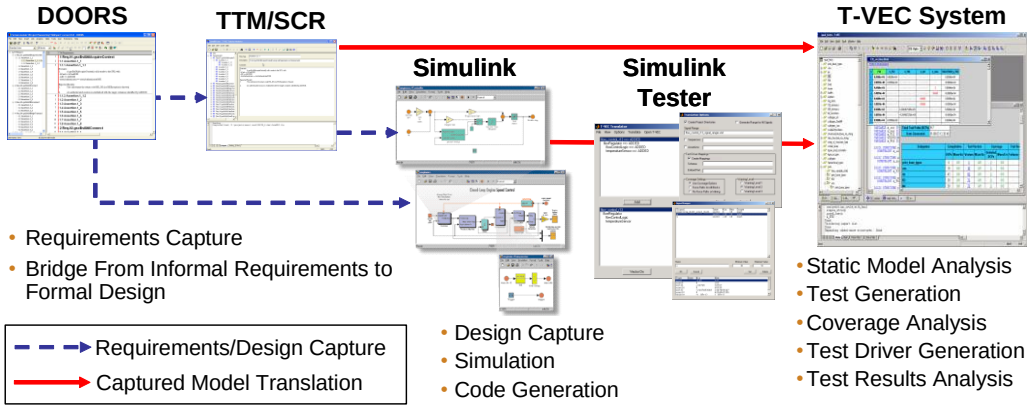

Figure 2. TAF Integrated Environment

3.2 Requirement-to-Test Traceability

This section provides an example to explain the process for linking DOORS requirements to the TTM requirements model. The tool support for requirement-to-test traceability involves linking various sources of requirements through the model. The model transformation, test vector generation, and test driver generation provide the tool support to link the requirements to the test vectors, test drivers, and test reports. The process, as shown in Figure 3 has three basic steps:

1. A DOORS module is imported into the TTM. There are options to add or delete a DOORS module to TTM or synchronize DOORS modules when they are updated. There is a one-to-one correspondence between a DOORS ID and a

TTM requirement ID.

2. Imported requirements maintain the outline structure that they have within the DOORS environment. One or more DOORS requirements can be linked to an element of a TTM model (e.g., condition/assignment) as shown in Figure 3, or linked to a higher-level in the TTM model, such as a condition, event or mode table as shown in Figure 4.

3. The model translation maintains the link between the requirement ID, and during test generation the requirement link is an attribute of the test vector. During test driver generation, requirement IDs can be output to the test driver to provide detailed traceability to the executable test cases.

TTM provides requirement management functionality that is similar to a DOORS module. Imported DOORS modules are linked into TTM as read-only modules. Changes to the requirements must be made within DOORS and then synchronized within TTM. Additional requirements can be created directly in TTM if they are not contained within DOORS or if the source requirements are not in a requirement management system such as DOORS. The process to link a requirement to the model is the same.

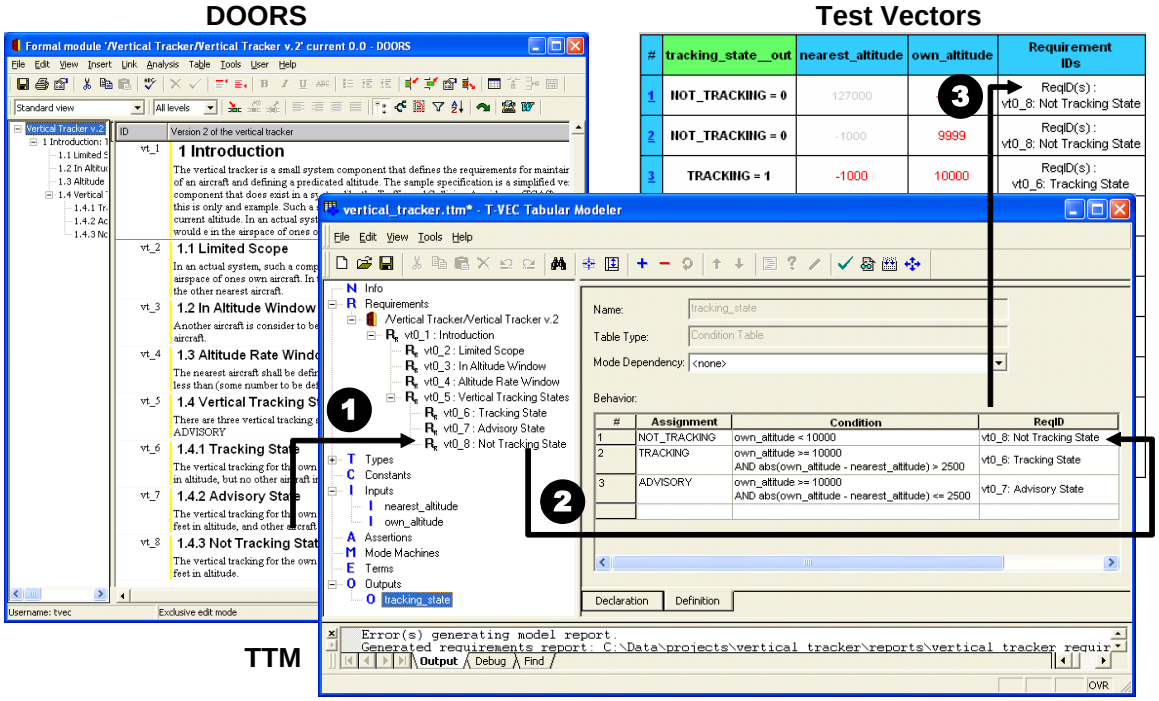

Figure 3. Requirement Links From Model to Test Vectors

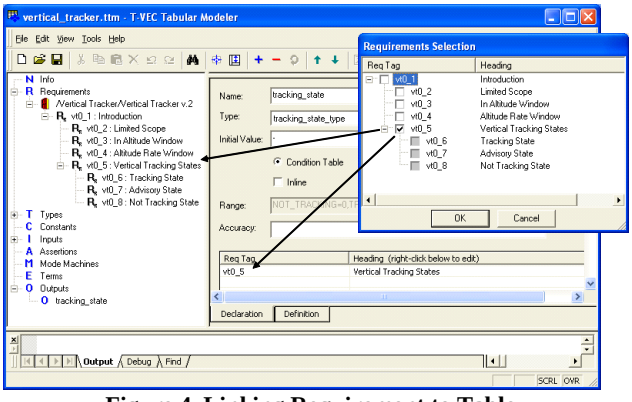

Figure 4. Linking Requirement to Table

3.3 Design Model Traceability

This section describes a process for linking requirements into a Simulink design model and tracing the requirement to test vectors. Figure 5 shows a Simulink model for the vertical tracker requirements. Linking within Simulink is on a block-by-block basis (e.g., relational operator, absolute value, switch), and are sometimes difficult to pinpoint, because requirements are associated with a thread through the model. For example the switch block labeled "Advisory or Track" is related to two requirements associated with the requirements defined by DOORS ID vt_6 and vt_7. A switch block pass through input 1 when input 2 is true otherwise it passes through input 3. The Tag field of the associated Block Property for the switch block contains a reference to both DOORS requirements, because the outputs of the switch block can be either Advisory or Track.

During the translation process, which is carried out by the Simulink Tester shown in Figure 2, the associated Tag fields are included as attributes in the transformed model from which test vectors are generated. During the test vector generation process, the requirement IDs are associated with the generated test vectors. The resulting test vectors, shown in Figure 5, include a list of the requirement IDs associated with each Simulink block in the path for the test vector.

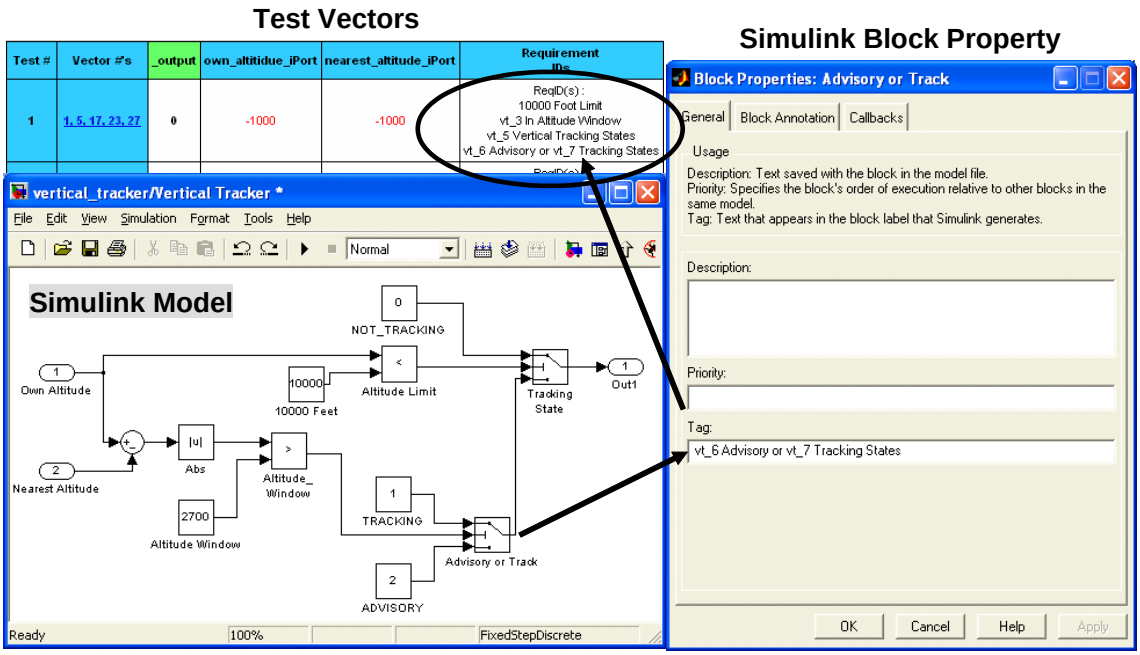

Figure 5. Requirement Links in Simulink Model

Test drivers are generated for both the MATLAB simulator, which executes the tests against the Simulink model, and code that is generated through the MATLAB Real-time workshop. Once the test drivers are executed a test report file is produced as shown in Figure 6. If a failure occurs there is a one-to-one correspondence between the test number in the test comparison report and the test vector report, allowing the failed test to be traced back to the model and to the associated requirement.

Expected Results File	vertical_tracker_root.eot
Actual Results File	vertical_tracker_root.out
Number Vectors Checked	12
Time Run	02-16-05 15:49:58
Total Comparisons/Bound	12/12
Errors	0
Test Status	PASSED

Test	Object	Expected	Actual	Result
1	_output	0	0	OK
2	_output	0	0	OK
3	_output	2	2	OK
4	_output	2	2	OK
5	_output	0	0	OK
6	_output	0	0	OK
7	_output	1	1	OK
8	_output	2	2	OK
9	_output	0	0	OK
10	_output	0	0	OK
11	_output	1	1	OK
12	_output	1	1	OK

Figure 6. Test Results Report

4 MODEL TRACEABILITY

Requirements often come from different sources and requirement change notices are a very common way to document new features or capabilities that are added to a system through incremental releases. During the modeling process, the requirement features may be allocated to different subsystems with dependency relationships, and some features allocated to different subsystems may be inconsistent resulting in a model defects. Model traceability helps to identify defects within a model. This is even more important for larger models that are composed of sets of related models.

T-VEC performs a form of model checking on each DNF during test vector generation, and creates reports identifying the defects. A simple example of a model defect is a logical contradiction, where a constraint such as `(x > 0) & (x < 0)` is in the DNF specification. Model checking hierarchically composed subsystems involves checking the satisfiability of constraint or function references between higher-level (i.e., grandparent) and lower-level (i.e., child) subsystems. For example, as conceptually shown in Figure 1, if there is a constraint, `x > 0` in a DNF thread from the grandparent subsystem to a child subsystem, there must be at least one DNF through the parent and child that permits `x > 0`. If such a constraint in the grandparent cannot be satisfied, then the input space for that DNF of the grandparent is empty (i.e., null), and no test inputs can be selected; this is a model defect. During test generation, model traceability is used to link a model-defect report back to the model location that is a likely source of the defect.

4.1 Model Traceability Linkage Examples

The example in Figure 7 represents a trivial model with four SCR condition tables modeled in TTM. This simplified model

has a seeded defect to illustrate the model traceability links from a model report to the model. The tables have dependency relationships to illustrate the use of model traceability. Each row of each table in the transformed model has a one-to-one correspondence with a DNF thread. The highest-level subsystem, `hierarchical_root` has one DNF that references `child_yz`, and `parent_xy`, each with two DNF threads. `Parent_xy` references `child_xy`, which also has two DNF threads.

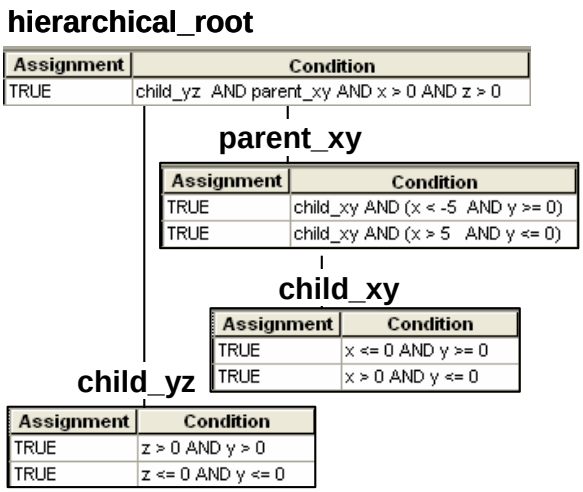

Figure 7. Hierarchical TTM Model

The traceability links from the status and error reports to the likely source of the model error is shown in Figure 8. The status report provides a summary for each subsystem, including the number of DNFs (referred to as Domain Convergence Paths or DCPs [1]) derived during the compilation process of the model. The summary report provides the number of test vectors, and the number of model coverage errors. Hyperlinks from the project status report link to other reports including the model defect error report that is produced for each DCP that has a defect. A hyperlink from the model error reports traces back to the model specification that is the likely source of the problem.

The defect exists because there is no combination of DNF threads through the lower-level subsystems that permit both x and z to be greater than zero when the output (i.e., assignment) of `hierarchical_root` must be TRUE. The model `child_2_xy` requires y <= 0 when x > 0, but `child_2_yz` requires y > 0 when z > 0. Thus, a contradiction exists between the logic of `hierarchical_root` and logic across two dependent subsystems.

Figure 9 is a Simulink model that represents the same specification as defined in Figure 7. Like the TTM model, the Simulink is translated and an attempt is made to generate test vectors, however, due to the defect it is not possible to produce a test vector when the output value of `hierarchical_root is` TRUE. The tools produce a similar set of hyperlinked reports as shown in Figure 10, that link back to the source of the Simulink® model problem.

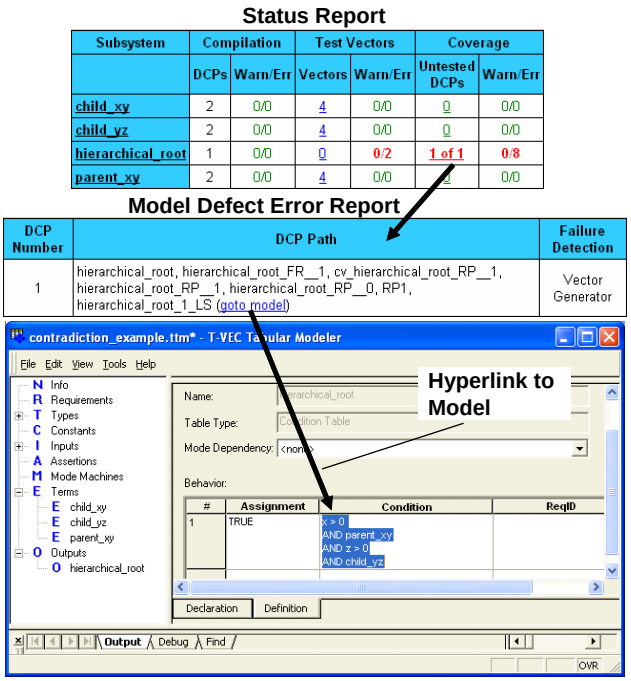

Figure 8. Model Defect Traceability to TTM

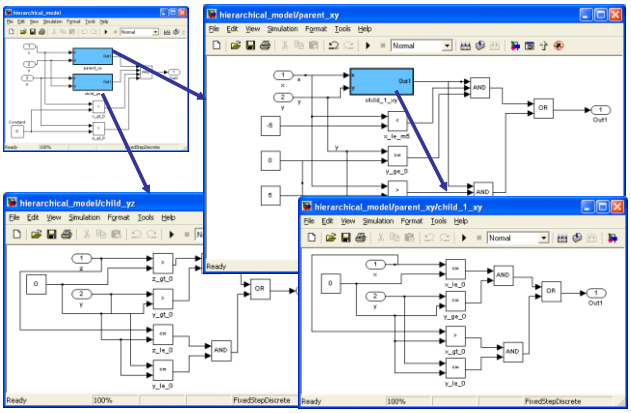

Figure 9. Model Defect Traceability to Simulink

4.2 Other Model Checking

The model checking capability supports also proof of properties (e.g., safety). Model assertions representing safety properties are specified external to the model, and during the test generation process, if test vectors are generated from a safety property assertion that is associated with a model, the test vector identifies a DNF thread through the model, where the safety property is violated. If a test vector is not produced, the safety property is not violated.

For example, one of the largest and most complex Simulink models where T-VEC is being applied is the avionics system control law model for the Lockheed Martin Joint Strike Fighter. A common safety-related situation to avoid for an avionics system is to ensure that the radar is not enabled when the aircraft is on the ground, usually referred to as weight on wheels (WOW), because this could cause harm to people near the aircraft. An example of safety property to model check would be WOW & Radar = ENABLED. This assertion would

be combined with the entire Simulink model, and if a test vector is produced, it identifies the thread through the model where the safety property is violated.

Other checks such as mathematical errors or potential errors (e.g. division by a domain that spans zero) are flagged as being a potential divide-by-zero hazard, or range overflow or underflow, where variables of the model have values outside the specified bounds of the type of that variable. The error reports generated for these errors link back to the model source.

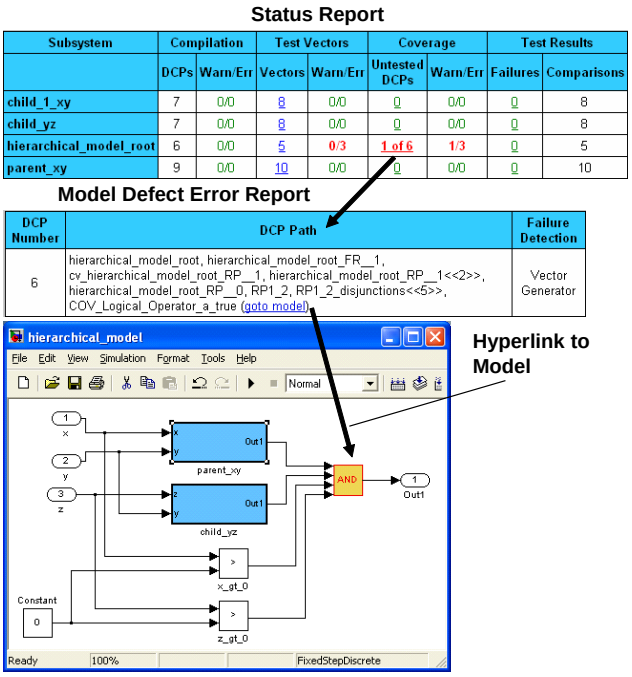

Figure 10. Simulink Model Traceability

5 SUMMARY

Model-based testing has many benefits including better quality requirements, better tests, and faster test design, but in working with companies since 1996, organizations are often resistant to adopt technology until it is demonstrated to satisfy a large set of requirements. Such requirements include end-to-end life cycle support, extensive expressiveness in the modeling techniques and languages supporting test execution for many languages and environments, with return on investment in a short timeframe.

During the past several years, most organization we have worked with have been requesting integration of requirement management tools with our requirement and design-based modeling and test generation tools. This paper describes several of the recent developments that include the integration of the DOORS requirement management tool with both the TTM requirement modeler, as well as the integration through the Simulink-based design modeling environment to provide full model-based test generation and test driver generation support. The integration with DOORS provides full requirement-to-test traceability, and some of the key benefits derived through the traceability process help to foster organizational adoption. A few examples include:

1. The requirement modeling process includes support for linking requirements to detailed modeling statements and makes the completeness of the model with respect to the requirements document more visible. Requirements without a link in the model are exposed; missing links can imply that a requirement is not modeled, or that the requirement is not modelable or testable, and requires further refinement.

2. Requirements that are too vague are exposed early during the modeling process when they can be more cost-effectively partitioned or refined. For, example, a requirement is too large when a requirement must be linked to an entire table (e.g., SCR condition table) in a model or multiple tables rather than to a table element.

3. Models without a link to a requirement are made visible; such requirements may be implementation-derived, or inherent, possibly requiring further documentation that should be included in the requirement source.

4. Requirements linked forward to each executable test case permits test failures to be linked back from the implementation, to the model and to the requirement source. This can significantly reduce the failure analysis time and cost.

5. Project managers can better understand how measures derived from model-based analysis and testing can be used to help in project, product, and process measurement, where such measures support decision making within their organizations. A recent paper [4] describes the fundamental units of measure derived from the model-based artifacts. Requirement-to-test traceability provides key information to automate the management and generation of project measurement information. We have created another tool to support the use historical measures for predicting project duration and usage of real-time project data to predict the completion of an ongoing project.

Another recent addition to the tools provides model traceability links from the model defect error reports to the model locations that are the most likely source of the model defects. Model traceability through hyperlinked error reports allows model or requirement defects to be quickly identified, reducing the cost of rework.

Users have been the key drivers for the model-based test traceability discussed in this paper. Many of the same users have reported on the benefits of TAF's model-based testing support in terms of cost savings through test automation [5, 6], early identification of requirement defects to reduce rework cost [7], systematic support to identify critical system defects [8], advanced capabilities [9], and applicability to other domains such as security [10, 11]. However, continually extending model-based testing to support the full life cycle seems to be significantly important for additional organizational adoption.

6 ACKNOWLEGEMENT

We would like to acknowledge Chris Snyder who did an excellent job on the integration of DOORS to TTM.

7 REFERENCES

[1] Rob.Hierons.
http://www.brunel.ac.uk/~csstrmh/research/test_z.html.

[2] Blackburn, M.R., R.D. Busser, Automatic Generation of Test Vectors for SCR-Style Specifications. Proceeding of the 12th Annual Conference on Computer Assurance, June, 1997. http://www.t-vec.com/download/papers/tvec_scr2tvec.pdf

[3] Heitmeyer, C., R. Jeffords, B. Labaw, Automated Consistency Checking of Requirements Specifications. ACM TOSEM, 5(3):231-261, 1996. See http://chacs.nrl.navy.mil/personnel/heitmeyer.html.

[4] Blackburn, M.R., Objective Measures from Model-Based Testing, STAREAST, Orlando, May 2004.

[5] Statezni, David, Industrial Application of Model-Based Testing, 16th International Conference and Exposition on Testing Computer Software, June 1999.

[6] Statezni, David. Test Automation Framework, State-based and Signal Flow Examples, Twelfth Annual Software Technology Conference, May 2000.

[7] Safford, Ed, L. Test Automation Framework, State-based and Signal Flow Examples, Twelfth Annual Software Technology Conference, May 2000.

[8] Blackburn, M. R., R.D. Busser, R. Knickerbocker, R. Kasuda, Mars Polar Lander Fault Identification Using Model-based Testing, NASA Software Engineering Workshop, November 2001. See http://www.software.org/pub/taf/downloads/mars_polar_lander_2001.pdf.

[9] Busser, R.D., L. Boden, Adding Natural Relationships to Simulink Models to Improve Automated Model-based Testing, Digital Avionics Systems Conference, October 2004. See http://www.software.org/pub/externalpapers/papers/busser-2004-2.doc.

[10] Chandramouli, R., M. R. Blackburn, Model-based Automated Security Functional Testing , 7th Annual Workshop on Distributed Objects and Components Security (DOCSEC), Baltimore, MD, April 2003.

[11] Blackburn, M. R., R. Chandramouli, Using Model-Based Testing to Assess Smart Card Interoperability Conformance, International Conference on Computing, Communications and Control Technologies, Austin, August 2004.