SPECIFICATION-DRIVEN TESTING OF SMART CARD INTERFACE USING A FORMAL MODEL

Ramaswamy Chandramouli
National Institute of Standards and Technology
Gaithersburg, MD, USA
ramaswamy.chandramouli@nist.gov

Mark R. Blackburn
T-VEC Technologies, Inc.
Herndon, VA, USA
blackburn@t-vec.com

ABSTRACT

Model-Driven Engineering (MDE) is emerging as a promising approach that uses models to support various phases of system development lifecycle such as Code Generation and Verification/Validation (V &V). In this paper, we describe the application of a model-driven process in the V &V phase for developing automated tests for testing the conformance of a smart card implementation to an interface specification. The smart card implementation under focus is the Personal Identity Verification (PIV) cards to be issued to the employees/contractors of the US government for physical access to government facilities and logical access to government IT systems. Our description of the model-driven conformance test generation application includes model development from specification and the subsequent use of the model for automated test generation, test execution and results analysis. We also illustrate the re-usability of model components for modeling related specifications as well as the extensibility of the model for testing smart card use case scenarios that involve invocation of sequence of commands to form a transaction.

KEYWORDS

Modeling, security properties, automatic test generation, formal models, smart cards, model reuse.

1. INTRODUCTION

Model-Driven Engineering (MDE) is emerging as a promising approach that uses models to support various phases of system development lifecycle such as Code Generation and Verification/Validation (V &V). In this paper, we describe the application of a model-driven process in the V &V phase for developing automated tests for testing the conformance of a smart card implementation to an interface specification. The smart card implementation under focus is the Personal Identity Verification (PIV) cards [FIPS 201] to be issued to the employees/contractors of the US government for physical access to government facilities and logical access to government IT systems.

The remainder of the paper is organized as follows. Section 2 provides an overview of the PIV specification, verification objectives, model organization, and conceptual architecture of the verification system. Section 3 describes the characteristics of the particular PIV sub- specification (i.e. PIV Card Command Interface specification) and the features provided by the candidate formal modeling system (i.e., SCR) for modeling this specification. Section 4 explains in detail the development of verification model for PIV Card Command Interface specification and the traceability of the model elements to specification elements. Section 5 describes the test generation and test execution details. Section 6 outlines the extensibility and re-use features of the model, followed by a summary of the benefits and final conclusions.

2. OVERVIEW

Following a presidential directive [HSPD-12] to develop a uniform, interoperable and tamper-proof set of credentials for personal identify verification of US government employees and contractors, the National Institute of Standards and Technology (NIST), an agency under department of commerce, developed a specification for personal identity verification (PIV) system based on dual-interface (contact and contactless) smart cards [FIPS 201]. This specification had a companion document (SP 800-73-1) [Dray] that specified in detail the following aspects regarding the PIV card:

- PIV Middleware Interface Specification (also called PIV Client Application Programming Interface specification) – SP1
- PIV Card Command Interface Specification (also called PIV Card Edge specification) – SP2
- PIV Data Model – SP3

In addition to specifications (SP1 through SP3 above), NIST was also tasked with the responsibility to develop test suites for testing commercial products for conformance to the above specifications.

NIST decided to use the specification-based test automation process, an integral part of MDE, for developing the conformance test suite. The first step in this process required the development of formal models of PIV specifications (SP1 through SP3). However, in this paper, we only describe the modeling and conformance test development for the PIV Card Command Interface specification (SP2).

Figure 1 provides a conceptual architecture of the elements involved in the overall modeling and testing process. The inter-dependency of the models developed to support the above specifications is shown in Figure 1 as the PIV Specification Models. In this figure, the PIV Specification Models include the block labeled PIV API, which stands for PIV Middleware Interface specification (SP1) and the one marked PIV APDU denotes PIV Card Command Interface Specification (SP2). The PIV Data Model (SP3) is included under PIV Requirements block in the diagram and is not shown explicitly. Test vectors and test drivers for conformance testing are generated from these models. The generated tests are packaged into the PIV Tester that is an installable and executable program that runs through Java. The PIV tester permits users to configure and execute the generated tests against implementations of the PIV specification, as well as to view tests logs and test reports. Configuration data is required to support different implementations (product offerings) under test, as well as variable data within a single implementation (e.g., PIN, Public Key Certificates etc).

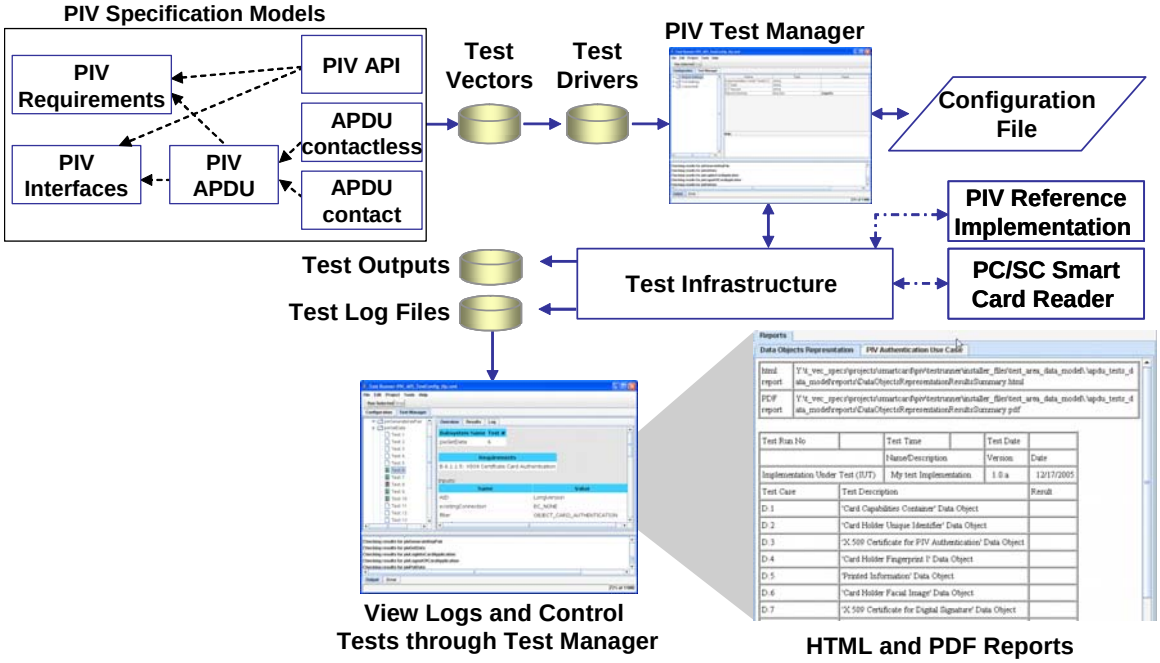

Figure 1. Conceptual Architecture of Model-driven Testing Application

3. CHARACTERISTICS OF SMART CARD COMMAND INTERFACE SPECIFICATION

The PIV smart card supports a single application called *PIV Application*. Hence the card command interface specification mostly pertains to behavior of a set of commands (or APDUs) in the context of this single application. The only exception is the SELECT APDU which contains a parameter that can carry the value of any valid identifier (called AID) of an application resident on the card. The application contains some status variables whose values determine the state of the application (*Application State*). The application state therefore directly pertains to the semantics of the application logic. For some APDUs, the application state determines its successful execution. This state therefore becomes the pre-condition for that APDU. Some APDU executions alter the application state and the new application state therefore becomes the post-condition for that APDU. The overall specification for the interface which stipulates behavior for each of the commands or APDUs, therefore consists of the following:

- Pre-Condition (application state before APDU invocation) (SE1)
- Set of parameters (or APDU components) in the designated sequence and associated valid values – Collectively this is called Function Signature specification.(SE2)
- Expected return codes and data (where appropriate) for a given combination of parameter values and/or pre-conditions (SE3)
- Post-Condition (application state after successful APDU execution) (SE4)

Interface specification for functions operating on stateless software will have just the elements SE2 and SE3 as there is no concept of application state. An example is the function that requests a static webpage from a webserver. Interface specifications for some applications that have the concept of user sessions (such as Query-Only Databases) may have specification elements SE2, SE3 and a rudimentary pre-condition element (SE1) such as status of login authentication. In these cases, the pre-condition is not strictly an application state but an environmental condition. Also in these types of applications, there is no new application state that results from the execution of the function. However, most of the function (command) specifications for the PIV smart card interface has associated with it a pre-condition and post-condition. Interface specifications that describe application state changes define what is known as Finite State Machine (FSM) model and the underlying system is called a FSM. The PIV smart card therefore is an FSM. Further, an interesting characteristic of the PIV card command specification is that the pre-condition for some commands is dependent upon the parameter value used in that particular command invocation. An example is the GET DATA APDU which requires the pre-condition PIN VERIFIED for some application objects (parameter value P1 in GET DATA APDU) and not for others.

We used the SCR formal model [Heitmeyer] for modeling the PIV Card Command Interface specification since there is a track record of case studies involving successful application of this model for requirements analysis of many high assurance systems by Naval Research Lab. The SCR uses tables to model the behavior of functions in the interface in terms of data types and variables pertaining to those functions. Variables can be defined in terms of primitive types (e.g., Integers, Float, Boolean, Enumeration), or user-defined types. The application state changes due to execution of functions that result in a finite state machine behavior are also modeled using a combination of tables called Mode Tables, Condition, or Event Tables. The constructs of the SCR method support directly those needed to model PIV Smart Card Command interfaces (i.e., SE1, SE2, SE3, and SE4).

4. MODELING OF SMART CARD INTERFACE SPECIFICATION

The PIV Card Command Interface specification [Dray et al] specifies the behavior of 8 card commands or APDUs. The specification elements related to pre-condition (SE1), function-signature, mostly parameters (SE2) and post-condition (SE4) are given in Table 1 below: (The expected return codes and/or data (SE3) are not shown in order to avoid cluttering up the table).

Table 1. PIV Card Command Interface Specification Elements

APDU (Card Command)	Pre-Condition (SE1)	Parameter(s) (SE2)	Post-Condition (SE4)
SELECT	N/A	Application ID (AID)	APPLICATION SELECTED
VERIFY	APPLICATION SELECTED	PIN Identifier PIN data	(a)PIN VERIFIED (b) RETRY COUNTER = Reset Value(OR) (a) PIN AUTHENTICATION BLOCKED - TRUE
GET DATA	(a) APPLICATION SELECTED (b) PIN VERIFIED (for some data objects)	Data Object Tag	N/A
CHANGE REFERENCE DATA	(a) PIN AUTHENTICATION BLOCKED - FALSE	(a) EXISTING PIN (b) NEW PIN	(a)PIN VERIFIED (b) RETRY COUNTER = Reset Value (OR) (a) PIN AUTHENTICATION BLOCKED - TRUE
RESET RETRY COUNTER	(a) RESET AUTHENTICATION BLOCKED - FALSE	(a) PIN UNBLOCKING PIN (b) NEW PIN	RETRY COUNTER = Reset Value (OR) RESET AUTHENTICATION BLOCKED = TRUE
GENERAL AUTHENTICATE	PIN VERIFIED (if Internal Authenticate)	(a)CRYPTO ALGORITHM CODE (b) KEY REF (c) CHALLENGE (or) RESPONSE	AUTHENTICATED
PUT DATA	AUTHENTICATED	Data Object Tag	N/A
GENERATE ASYMMETRIC KEY PAIR	AUTHENTICATED	(a)KEY REF (b) REF TEMPLATE (dependent upon crypto algorithm)	N/A

4.1 SCR And TTM Modeling of Interface Specification

Each of the APDU behavior (output) is modeled using a SCR's condition table. The condition table is named the same as card command (see Table 1), with an "**_apdu"** extension. For example, the condition table for the GET DATA command, shown in Figure 2, is named get_data_apdu. As shown in Table 1, the pre-condition for many APDUs are outcomes from some of the other APDUs. To facilitate expression of these dependencies between APDU commands, each modeled APDU output is related to one term variable, because term variables in SCR represent intermediate models that can be referenced (reused). This makes the model modular since the modeled sub-elements can simply be referenced rather than re-specified for each output. In addition, terms can reference other terms.

The term related to the output of the command is prefixed with a "**t_**" followed by the name of the command. For example, consider the model for GET DATA command shown through the **get_data_apdu**, table in Figure 2**.** The get_data_apdu model references the term table **t_get_data_apdu** for each of the testable return codes. In addition, the model also contains input variables (e.g., objectid = CARD_HOLDER_FACIAL_IMAGE).The model thus captures the output behavior of the GET DATA command for all Return Codes or Status Word Types. The dependency relationship is shown more clearly for the GET DATA APDU through Figure 3. This figure also shows other term dependencies. The term **t_verify_apdu** is dependent on **t_select_data_apdu**, and **t_get_data_apdu** is dependent on both **t_select_apdu** and **t_verify_apdu**.

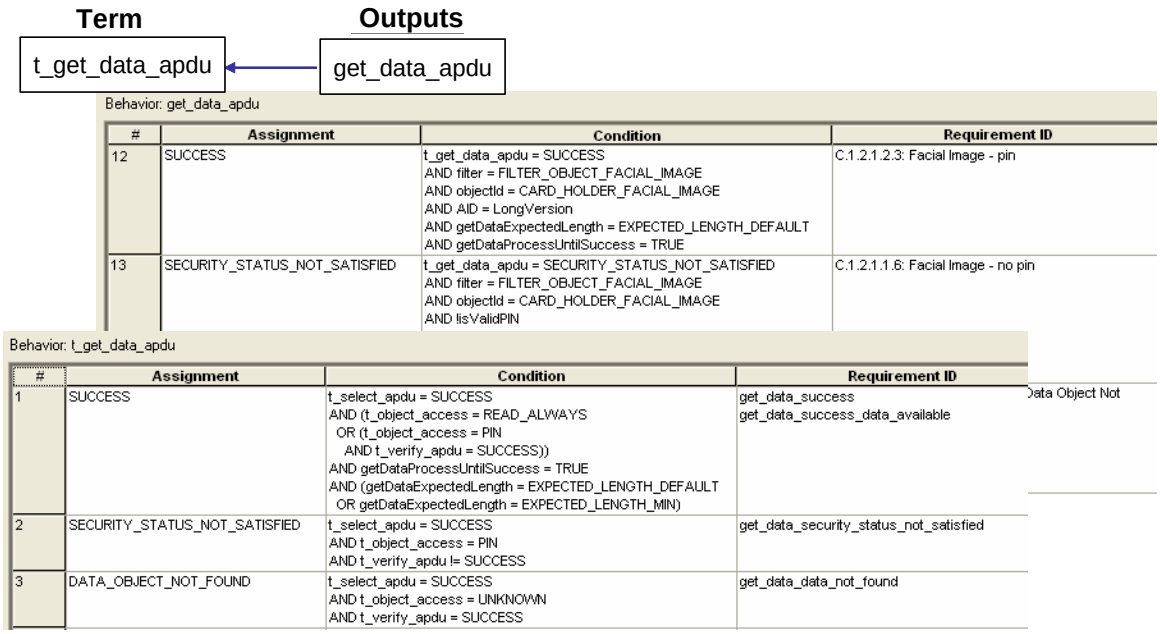

Figure 2. Condition Table for GET DATA Card Command

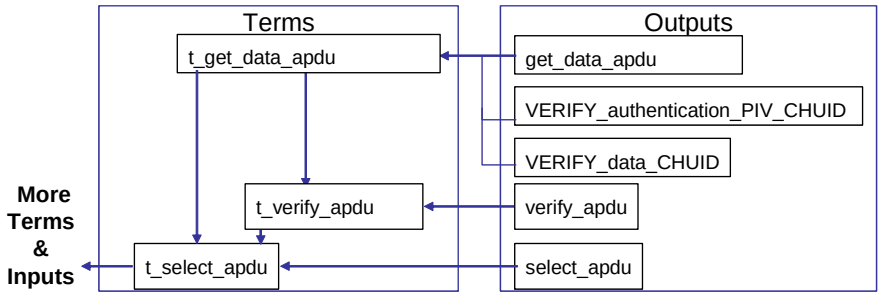

Figure 3. Dependency Relationships for GET DATA & OTHER COMMANDS

From our above discussion, we could see that the model for a command or APDU in the PIV Card Command Interface consists of input variables, terms and an output variable. In addition, the model for some APDUs can contain constants as well. The SCR model has features to define data types for each of these variables. For example, StatusWord or Return Code variable in our model is an enumerated type that identifes the possible response codes (e.g., SUCCESS, SECURITY_STATUS_NOT_SATISFIED, DATA_OBJECT_NOT_FOUND, etc.) for an APDU invocation. Our overall SCR model for the PIV Card Command Interface specification consisted of 19 data types, 47 input variable, 27 terms and 28 constants..

Please note that our modeling discussion so far pertains to only one specification related to PIV Card (PIV Card Command Interface Specification – SP2). In order to link up this model with models for the other two PIV specifications referred to earlier (i.e., PIV Middleware Interface Specification – SP1 & PIV Data Model specification – SP3) for the purpose of re-using modeling elements, a model management framework is needed. The T-VEC Tabular Modeler [TTM] is a tool that provides this management framework to manage SCR and other formal models. The TTM modeling framework supports the inclusion of existing models of other requirements, interfaces, or functional behavior. This feature helps consolidate behavior common to multiple models into a single model and includes it in other models where needed. Further, the TTM tool enables organization of these specifications or requirements in a hierarchical fashion as well as capturing of the PIV requirements header so that requirement headers can be linked to the detailed requirements for traceability, as shown in Figure 4.

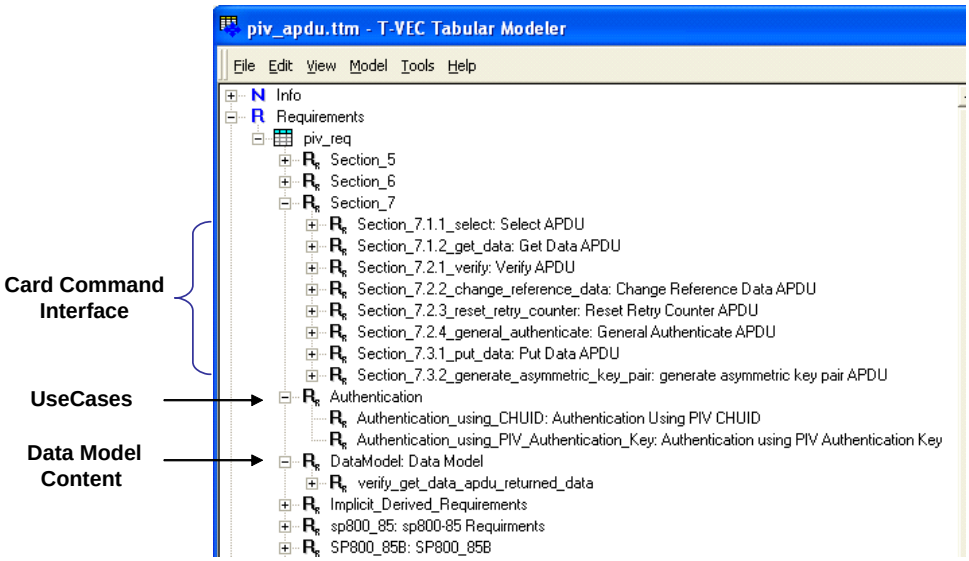

Figure 4. Requirements for Traceability to Model

5. TEST GENERATION AND VERDICTS

The section discusses the process for using the models imported into TTM to support test vector and driver generation, the use of configuration information to control the tests, and test execution and reporting.

5.1 Test Vector Generation

To generate test vectors, the SCR model must be translated 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 [Hierons]. Test vectors generation selects test data for subdomains of an input space based on the constraints of each DNF. A subdomain convergence algorithm is used to determine a DNF subdomain. If a nonempty subdomain exists for a DNF, then the input values associated with a test point are selected for the borders of the subdomain. A ***border*** is defined by evaluating the predicates of a DNF for a set of input values. For example, test points for numeric objects are selected for both upper and lower domain boundary values. This results in test points for subdomain borders based on all low-bound values and high-bound input values that satisfy the DNF predicate evaluation.

The PIV Card Command Interface specification specified behaviors for APDUs or commands accessible through both contact and contactless card interfaces. Our model generated 72 test vectors for testing the Contact interface and 29 tests for the contactless interface, but sub tests involving data (e.g., Object Tag for GET DATA APDU through Contact interface) in the contact interface were re-used in the tests for contactless interface.

5.2 Test Driver Generation

The generated test vectors include generic inputs and terms for each output associated with model variable, but they must be mapped to parameters or data values associated with the test environment to support test driver generation. The test drivers can then be executed against a particular smart card or reference implementation. Configuration information, described in Section 5.3, is used to support the test execution process.

Each input in the PIV Card Command Interfaces model is mapped (e.g., AID – Application ID, PIN) into variables in the test harness (actual software variables in the environment under test) and then their actual values must be set. The data structure that provides this translation is known as "object mapping." An object

mapping specifies the relationship between model entities and implementation interfaces that are used for sending and receiving commands to the smart card.

Also for execution of each test iteration, the input values have to be reset, a new test vector(s) has(have) to be loaded and the generated values have to be cleaned up at the end of the test. All these house-keeping activities are encoded into an algorithmic pattern of sequence of steps called "test schema." The behavioral model, the test vectors, the object mapping file and test schema file form all the ingredients necessary for generating executable code and are thus fed into the test driver generator to produce input that can execute test code against the target environment. The test driver generator generates also and Expected Output file (EOT) based on the test vectors (that form sets of input/output values) in the test vector suite.

The test driver code is executed against an implementation (Dual interface smart card loaded with PIV Application) to verify whether its behavior conforms to PIV specification. Configuration information is used to support this activity as cards can have optional data objects and choices in the cryptographic algorithms supported etc, and some of the test functions are dependent on data resident within the card (e.g., PIN).

5.3 Configuration Settings and Test Execution

As reflected in Figure 5, the implementation under test carried a number of configuration variables that must be specified to control the test environment through the test harness. An interesting aspect of these configuration variables is that they are not part of the parameters of the functions of the API that is being tested. However they can have an impact on the internal state of the smart card before and after the exercise of an APDU function.

The PIV Tester loads tests created by the test driver. For those variables that have specific values within the test environment, for example PIN_VALID shown in Figure 5, reflects that actual PIN required for login to a card under test, the PIV Tester must use the appropriate value from the configuration settings to carry out the test functions.

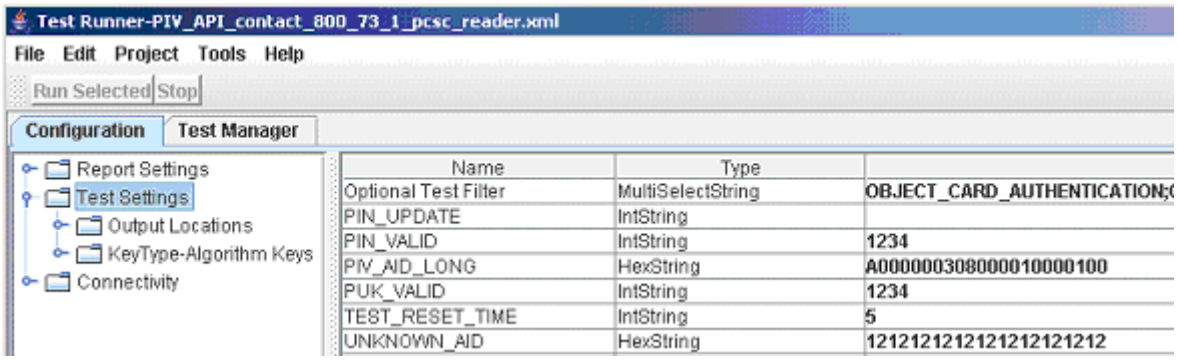

Figure 5. PIV Tester Configuration Information

This test execution generates the Actual Output file (AOT). A cross comparator tool then performs a check to ensure that the actual output and expected output are correct. A standard reporting structure was generating the test results report by comparing the EOT and AOT. The test results output is compliant with FIPS 201 Testing Guidelines documents [Chandramouli].

6. SUMMARY, BENEFITS AND CONCLUSIONS

We have shown that the model for a single PIV component (i.e., PIV Card Command Interface) is highly modular and built with many re-usable elements such as terms related to output variables. After a PIV card is validated for interface conformance, the next stage of the PIV card issuance lifecycle involved populating the card with credential data (called card personalization). At this lifecycle stage, the PIV card content must be tested for conformance to the data model specification (SP3 in section 2). The data model content verification model was also imported into TTM and merged with the PIV Card Command Interface model as illustrated by the marker "Data Model Content" in Figure 4. This enabled mapping of the model to generate tests

pertaining to data model structure and content. Further PIV card deployment architectures involved the use of PIV Middleware. To promote interoperability between several different middleware products and PIV cards, the middleware had to be tested for conformance to PIV Middleware interface specification (SP1 in section 2). Here again our PIV Card Command Interface model was re-used as well. Further real-world usage scenarios (e.g., Authentication) involved a defined sequence of APDU command exchanges called transaction. Our model was extended for testing the behavior of these transactions by combining the modeling elements relating to output of an APDU with the input elements of the succeeding APDU in the transaction definition.

Summarizing the model-driven conformance test generation application described here has the following primary benefits:

- Collection of tests that provide complete specification coverage
- Test system that provides features for semi-automated test execution, automated results analysis and report generation.
- Extensibility of the Model that enables it to be used for testing proper behavior for Usage Scenarios in addition to testing for interface conformance.
- Re-usability of the model components for testing an entirely different component such as the PIV Middleware (in the verification model used for testing PIV Middleware interface)
- Re-usability of the model components for testing the same product in the next stage of system lifecycle (i.e., testing a personalized smart card for data model conformance).

There are some secondary benefits for the model-driven test generation application as well. They are:

- The mathematically sound verification model for PIV smart card interface testing improved the quality of the specification by identifying some anomalies and eliminating several ambiguous interpretations that might have led to divergent implementations.
- As the specifications continued to evolve and undergo changes, easy update of modular model and automated regeneration of tests supported cost-effective re-verification along with comprehensive regression testing.

The deployment of the model-driven test generation application to validate and certify smart card offerings from several leading smart card vendors for conformance to PIV Card Command Interface specifications provides ample testimony to the robustness of the underlying methodology. The flexibility of the approach is illustrated by the fact that the various smart card products certified had implementation differences (different cryptographic algorithms and different subsets of data objects).

REFERENCES

Chandramouli, R et al, 2006. PIV Card Application and Middleware Interface Test Guidelines. http://csrc.nist.gov/publications/nistpubs/800-85A/SP800-85A.pdf

Chandramouli, R et al, 2006. PIV Data Model Test Guidelines. http://csrc.nist.gov/publications/nistpubs/800-85B/SP800-85b-072406-final.pdf

Dray, J. et al ,2006. Personal Identity Verification, NIST Special Publication 800-73-1. http://csrc.nist.gov/publications/nistpubs/800-73-1/sp800-73-1v7-April20-2006.pdf

FIPS 201,2006. Personal Identity Verification (PIV) of Federal Employees and Contractors. http://csrc.nist.gov/publications/fips/fips201-1/FIPS-201-1-chng1.pdf

Heitmeyer, C. , 2002. Software Cost Reduction. http://chacs.nrl.navy.mil/publications/CHACS/2002/2002heitmeyer-encse.pdf

Hierons, R, 2004. Testing from a Z Specification. http://www.brunel.ac.uk/~csstrmh/research/test_z.html.

HSPD 12, 2004. Homeland Security Presidential Directive -12. http://www.whitehouse.gov/news/releases/2004/08/20040827-8.html

Raytheon, 2003. Voice of the Customer,. *In Technology Today*, Vol. 2, No. 1, Spring 2003.

TTM , 2002. Tabular Model. http://www.t-vec.com/solutions/ttm.php