Using Model-Based Testing to Assess Smart Card Interoperability Conformance

Mark BLACKBURN
T-VEC Technologies/SPC Herndon, VA, USA
blackburn@tvec.com

and

Ramaswamy CHANDRAMOULI
National Institute of Standards & Technology
Gaithersburg, MD, USA – mouli@nist.gov

ABSTRACT

Smart cards are being used to provide security for many types of applications, and with an estimated market of 3.3 billion in 2005, their usefulness is based on their intrinsic portability and security. The National Institute of Standards and Technology (NIST) initiated the Smart Card Interoperability Program to provide standards (Government Smart Card Interoperability Specification –GSC-IS) and tests to accelerate the use of this technology. The GSC-IS provides specifications for interoperability for Smart Card-Based Applications at two layers – the first one called the Basic Service Interface (BSI) layer and the second one called the Virtual Card-Edge (VCEI) layer. The BSI layer is the interface between a client application and the smart card middleware while the VCEI layer is the interface between the Smart Card Middleware and on-card application. This paper describes the approach and results of a model-based test generation effort that tests a smart card middleware implementation for conformance to the BSI layer of GSC-IS specifications. Our approach consists of using the function signatures in the BSI layer of GSC-IS as requirements to develop a verification model and then generating test vectors and executable test code based on the model to facilitate testing of a smart card middleware implementation. The tests are generated for Java language binding of the BSI specification.

Keywords: Smart Card Interoperability Specification, Test Automation, Model-Based Testing, Requirement-based Testing, Requirement Modeling.

1. INTRODUCTION

The Government Smart Card Program is a joint program led by NIST and GSA, in conjunction with other federal agencies, with the primary goal of building a framework for smart card interoperability, enabling broad adoption of this critical technology by the public and private sectors. As a first step towards this goal, NIST developed the Government Smart Card Interoperability Specification (GSC-IS) [1], a set of interface requirements that provides interoperability at two layers and consists of the following:

- A set of interface specifications called the Basic Service Interface (BSI) that will enable any Client Application to interact with Smart Card Middleware (also called the Smart Card Service Provider Module (SCSPM)) to obtain a pre-defined set of services using standardized method calls.
- A set of interface specifications called the Virtual Card Edge Interface (VCEI) that specifies the format of commands (called Application program Data Units (APDUs)) that a GSC-IS compliant Smart card must support. To allow for the possibility of the usage of smart cards whose native APDU set is different from the set defined in VCEI, the GSC-IS also calls for definition of a data structure called the Card Capability Container (CCC) that contains the mapping from the GSC-IS specified VCEI APDU set to the Smart Card's native APDU set and the associated functionality within SCSPM to actually generate the mapped APDU.

To test any vendor offering for conformance to GSC-IS, NIST has to develop conformance testing approaches. One such approach is described in [2]. In another parallel effort, NIST also developed a conformance test development approach that is based on a well-established framework called the Test Automation Framework [3]. Based on this framework, we are seeking to develop a formal verification model of all functional requirements (in our case the GSC-IS interface specifications) for the targets under test (in our case the Smart Card Middleware (SCSPM) and the Smart Card itself) and use that as the basis for automatic generation of executable test code. However out of the two interoperability layers covered by GSC-IS, the discussions in this paper pertain to the modeling and development of tests for the 23 methods in the BSI layer of GSC-IS (i.e., between client application and smart card middleware (SCSPM)).

The organization of the rest of the paper is as follows. In section 2, we describe the overall GSC-IS architecture showing the role of the BSI and VCEI layers in the Smart-Card based application infrastructure. Section 3 outlines the salient steps involved in our model-based test generation approach for testing a Smart Card Middleware (referred to as SCSPM in this paper) for conformance to BSI method definitions. In section 4 we illustrate using an example the process of developing a Software Cost Reduction (SCR) [4] model of a BSI method and the logic involved in combining the models of individual functions (methods) to model transactions (sequence of method calls) and to generate conformance tests for those transactions. In the absence of a GSC-IS conformant Smart Card Middleware (SCSPM), we developed a SCSPM simulator to validate the generated test code. Section 5 provides summary statistics on the model paths and test vectors generated for testing a smart card middleware for GSC-IS BSI layer conformance.

2. GSC-IS ARCHITECTURE & TESTING GOALS

The foundational architecture of a smart card-based application based upon which GSC-IS interoperability specification has been formulated consists of the following components:

- Client Application
- Smart Card Middleware
- Smart Card Environment consisting of Card Reader Driver, Card Reader and on-card application.

The interaction layer between the client application and the Smart Card Middleware (SCSPM) is called Basic Service Interface (BSI). Interoperability at the BSI layer is defined in terms of function signatures for 23 BSI methods that should be supported by any Smart Card Middleware (SCSPM) implementation. The other interaction layer in the GSC-IS architecture is the one between the SCSPM and Smart Card Environment called the Virtual Card Edge Interface (VCEI). Interoperability at the VCEI layer is theoretically defined in terms of the ability of SCSPM to generate a command to the smart card that is listed in the default GSC-IS command set (called APDU set where APDU stands for Application Protocol Data Unit) as a consequence of invoking a BSI method or a sequence of BSI methods from a client application. Since it is not realistic to expect all smart cards to support this GSC-IS APDU set, practical interoperability at the VCEI layer is defined in terms of the smart card carrying a data container (e.g., an elementary file in a file system card) called by a special name – Card Capability Container (CCC) that contains the translation data for mapping commands in the default GSC-IS APDU set to the card's native APDU set as well as the correct program logic within the SCSPM implementation that performs this mapping to generate the appropriate card specific APDUs. Figure 1 is the schematic diagram showing the position of the two layers (BSI and VCEI) in the GSC-IS architecture as well as the list of 23 methods (functions) in the BSI layer.

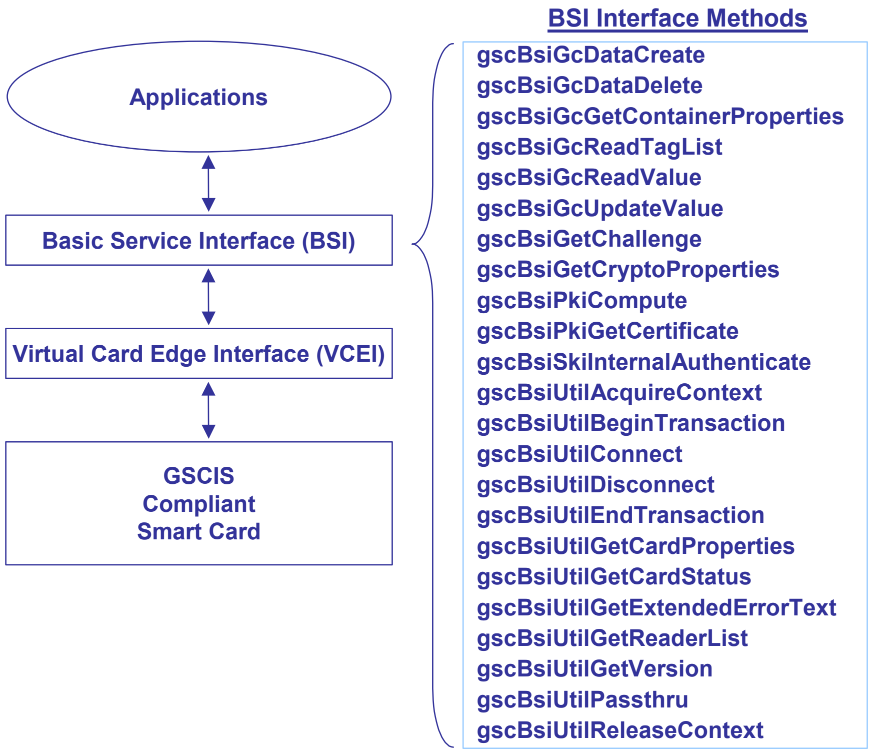


Figure 1 - GSC-IS Conceptual Architecture and the List of BSI Methods

The primary goals in developing a conformance test methodology for testing a SCSPM and a card implementation for conformance to BSI and VCEI layers respectively of GSC-IS are:

- To validate the GSC-IS for any inconsistencies and non-realizable conditions or outcomes.
- To identify difficulties encountered with other external components (like Smart Card Reader Driver, the Card Reader etc) in the smart-card application environment that are outside the scope of the GSC-IS and which may have a bearing on ensuring GSC-IS compliance.

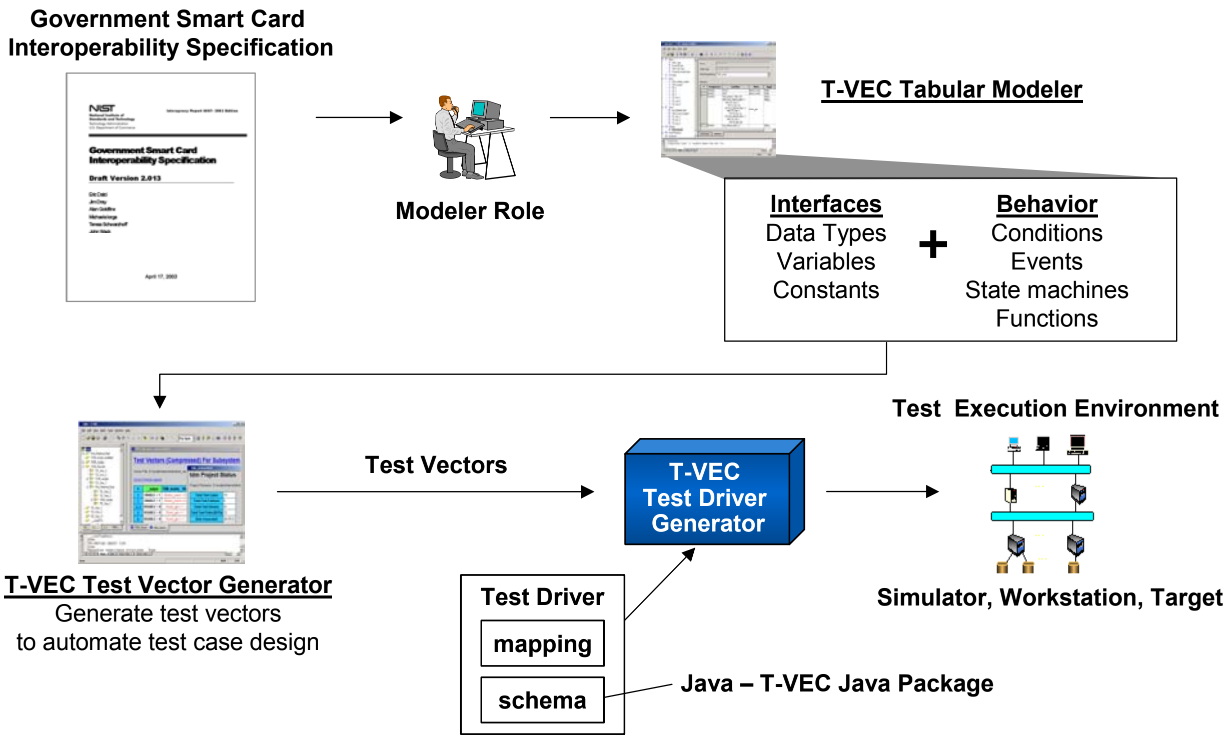


Figure 2. Test Generation for GSC-IS Conformance - Process Overview

3. SALIENT FEATURES OF MODEL-BASED TEST GENERATION

The process steps (shown in Figure 2) involved in developing and generating conformance tests for the BSI methods of GSC-IS are:

1. PS1: Building a Software Cost Reduction (SCR) model for the BSI methods using the T-VEC Tabular Modeler (TTM[1])
2. PS2: Translating the SCR model into T-VEC linear model [5].
3. PS3: Generating the test vectors from the model for testing conformance to BSI layer specifications of GSC-IS.
4. PS4: Generating a test driver to execute the test vectors against an SCSPM implementation
5. PS5: Execute the test driver
6. PS6: Evaluating test results and producing test results report

In the above sequence of process steps, the generation of the test driver itself involves several activities. The terms in the in the behavioral model (Process Step 2) have to be mapped to the variables in the interface commands that will exercise the product (in our case SCSPM for BSI layer testing) under test. This is the information that is shown with the title "mapping" in the diagram above. The syntax of the mapping information depends upon the language binding used. Here in our toolkit, the executable test code is Java and hence the "mapping" information consists of Java language variables. Next, we need an algorithmic pattern (e.g., initialization, loading of test vectors etc) for conducting conformance test for a BSI method. This information is labeled as "schema" in the above diagram.

4. MODELING AND TEST GENERATION – AN ILLUSTRATIVE EXAMPLE

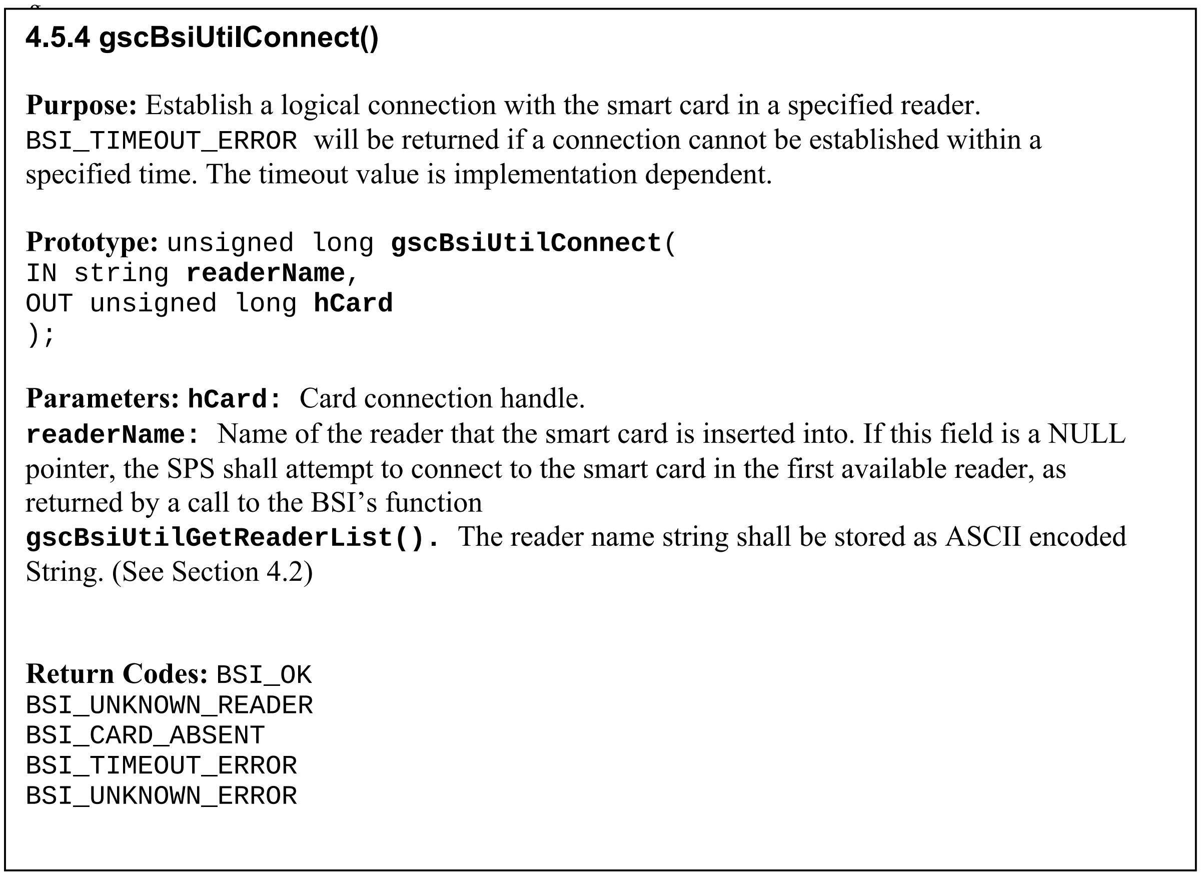
A meaningful and useful interaction in a smart card-based application occurs only through transactions. A transaction in turn is nothing but an ordered sequence of method calls. Consequently our GSC-IS conformance testing should consist of generating tests for verifying transaction behavior. Hence our overall tasks consists of modeling the individual methods, generating test vectors and executable test code for exercising each of those methods,

[1] T-VEC Tabular Modeler (TTM) is based on the Software Cost Reduction (SCR) method.

identifying the sequence of method calls for a particular transaction and then running the generated test codes for methods in that sequence. For example the transaction – read the tag value in a container in a smart card consists of the following sequence of method calls.

- gcsBsiUtilConnect
- gcsBsiGetContainerProperties (optionally a card may also be involved in multiple transactions)
- gcsBsiUtilAcquireContext
- gcsBsiGcReadTagList
- g

BsiGcReadValue



4.5.4 gscBsiUtilConnect()

Purpose: Establish a logical connection with the smart card in a specified reader. `BSI_TIMEOUT_ERROR` will be returned if a connection cannot be established within a specified time. The timeout value is implementation dependent.

Prototype:
```
unsigned long gscBsiUtilConnect(
IN string readerName,
OUT unsigned long hCard
);
```

Parameters: **`hCard:`** Card connection handle.
`readerName:` Name of the reader that the smart card is inserted into. If this field is a NULL pointer, the SPS shall attempt to connect to the smart card in the first available reader, as returned by a call to the BSI's function **`gscBsiUtilGetReaderList().`** The reader name string shall be stored as ASCII encoded String. (See Section 4.2)

Return Codes:
```
BSI_OK
BSI_UNKNOWN_READER
BSI_CARD_ABSENT
BSI_TIMEOUT_ERROR
BSI_UNKNOWN_ERROR
```

Figure 3 – Function Signature for gscBsiUtilConnect ( ) Method

Let us now look at the process involved in modeling the first method – gscBsiUtilConnect. In order to model this method we need to look at the definition of the method (called function signature) in the GSC-IS document (shown in Figure 3). The function signature specifies the name of the method, the set of input and output parameters and a several of return codes. The GSC-IS also gives textual description (not shown here) of the conditions (input parameter values and environmental settings (e.g., Card not in the Reader)) that determine each of the return code values. It is this textual description that provides the behavior of the methods and hence forms the building blocks for modeling the method in SCR. Specifically in SCR, a method is modeled as a condition table that has the same name as the method name and consisting of the following columns:

- A column named *condition* that contains predicates describing the combination of input values & environmental settings
- A column named *assignment* that gives the associated return code corresponding to the condition.

Since there are 5 return codes in the function signature for the gscBsiUtilConnect ( ) method, the SCR condition table for this Let us now analyze the input parameter values and environmental settings that determine a particular value of return code (i.e., BSI_OK).

BSI_OK when

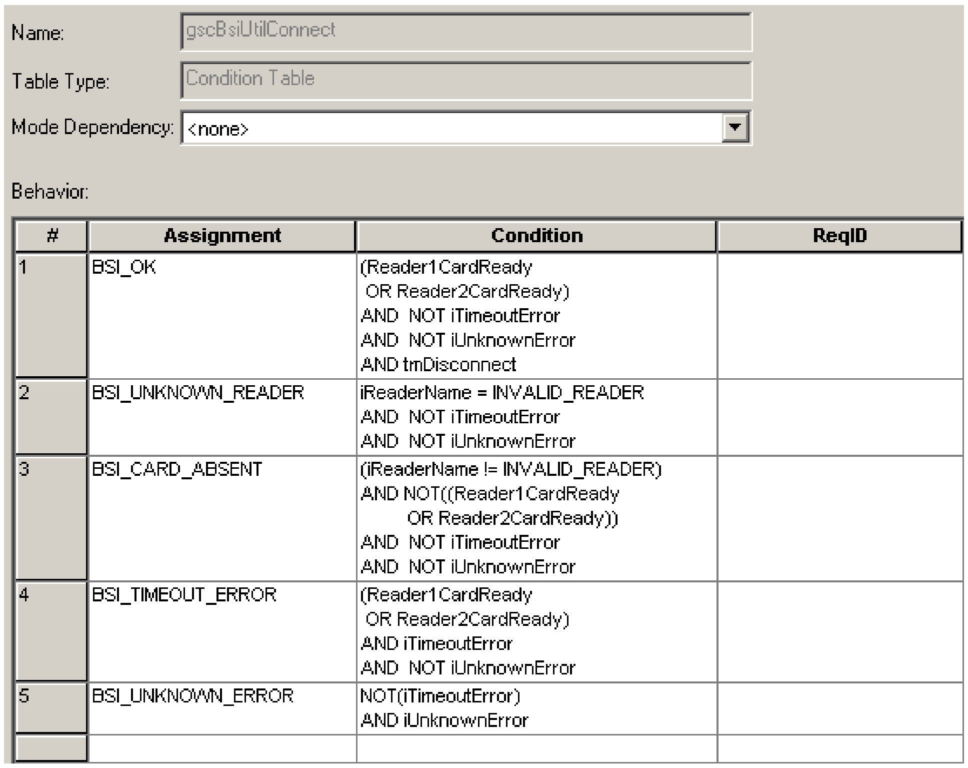
(Reader1CardReady OR Reader2CardReady)
AND NOT iTimeoutError
AND NOT iUnknownError
AND tmDisconnect

Where Reader1CardReady is a term that is TRUE when
(iReaderName = READER1 OR iReaderName = NULL)
AND

method consists of 5 rows as shown in Figure 4. Each different output (corresponding to a return code in the method signature) for the model of any method defines different conditions on the inputs or terms. Thus formulating conditions for all of the return codes of a method constitute a complete model of the method.

(iCardLocation = READER1 OR iCardLocation = BOTH)

This models the situation where there are one or more card readers, and there is a card location. If there is a valid card reader ready, and there is not a time-out or an unknown error, the method should return BSI_OK, and then disconnect. In addition, various error conditions are often established to represent an input to the subsystem. This is done because part of the model must represent elements of the test environment, such as the availability of a card, or a card in, or not in, the reader.



Name: gscBsiUtilConnect

Table Type: Condition Table

Mode Dependency: <none>

Behavior:

#	Assignment	Condition	ReqID
1	BSI_OK	(Reader1CardReady OR Reader2CardReady) AND NOT iTimeoutError AND NOT iUnknownError AND tmDisconnect	
2	BSI_UNKNOWN_READER	iReaderName = INVALID_READER AND NOT iTimeoutError AND NOT iUnknownError	
3	BSI_CARD_ABSENT	(iReaderName != INVALID_READER) AND NOT((Reader1CardReady OR Reader2CardReady)) AND NOT iTimeoutError AND NOT iUnknownError	
4	BSI_TIMEOUT_ERROR	(Reader1CardReady OR Reader2CardReady) AND iTimeoutError AND NOT iUnknownError	
5	BSI_UNKNOWN_ERROR	NOT(iTimeoutError) AND iUnknownError	

Figure 4 – Condition Table for the Method gscBsiUtilConnect ( )

From the above model definition, it should be clear that while modeling a method we take into account not only just the input parameter values as defined in the method signature but also external inputs as well so that the method's model can generate the data needed for conducting a self-contained test (with a complete test environment) on the method's behavior. Yet another feature of the SCR model for a BSI method behavior is that the conditions for the successful execution of the method also include predicates that indicate successful execution of any of the other methods that are pre-requisites to the method being modeled.

For example let us examine the method **gscBsiUtilAcquireContext**. Since this method is invoked after **gscBsiUtilConnect** and **gscBsiGetContainerProperties**, a success return code BSI_OK for this **gscBsiUtilAcquireContext** should include conditions where a successful connection has occurred [tmConnect] through invocation of **gscBsiUtilConnect**, all container properties have been retrieved [tmGetContainerProperties] through invocation of **gscBsiGetContainerProperties** and more specifically the access control rules (ACRset) have been set (values BSI_ACR_ALWAYS or BSI_ACR_XAUTH). In addition, there are a number of other conditions that must also hold for **gscBsiUtilAquireContext** to return BSI_OK: the handle to the card must be good [NOT tmBadHandle], there must be a valid application identifier [NOT tmBadAID], the access control rules must be available [NOT iACRNotAvailable], there is valid authentication data [NOT iBadAuth], the card is not removed from the reader [NOT tmCardRemoved], the pin is not blocked [NOT iPinBlocked], the card reader has performed a successful authentication exchange with the smart card [NOT iTerminalAuth], and no method has returned an unknown error code [tmUnknownError].

So far what we have described is just the process step 1 (PS1 in section 3). The SCR model of the BSI layer of GSC-IS now consists of a set of tables one for each BSI method. This model is not amenable for generation of test vectors. To facilitate this operation (i.e., generation of test vectors) the SCR model is linearized (made one-dimensional) into a disjunctive set of input-out relations (called functional relationships (FRs)) through process step 2 (PS2). The resulting model is called the T-VEC linear model.

Now each of the functional relationships (FRs) of the T-VEC linear model becomes a model path since a disjunction of the individual FRs make up the entire model. If at least one vector of values that satisfies a given FR can be found, we have obtained test data to cover that path. If we can generate a set of test vectors that satisfies all FRs in the T-VEC model then we have obtained complete model coverage. This is exactly the logic behind the test vector generation process step (PS3).

At this stage in our automated conformance test generation effort for GSC-IS, we have the T-VEC behavioral model and a set of test vectors. This information alone is not sufficient for generating executable test code. We need to map every input term defined in the behavioral model (i.e., iReaderName, iCardLocation etc) 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 what 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 from the Smart Card Middleware (SCSPM) and the card itself. 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. In addition there are some additional operations that have to be performed in a smart-card based application environment that have nothing to do with interoperability specifications. These include: Inserting/Removing smart cards from the reader, Attaching/Detaching Readers etc. All these house-keeping activities are encoded into an algorithmic pattern of sequence of steps called "test schema." Now 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 T-VEC's Java test driver generator to execute test code in Java (PS4). The test driver generator also generates and Expected Output file (EOT) based on the test vectors (that form sets of input/output values) in the test vector suite.

Now the test driver code has to be executed against a SCSPM implementation (PS5) to verify whether its behavior conforms to BSI layer specification of GSC-IS. Since we did not have a full-fledged SCSPM implementation to test against, we ran the generated test code against a SCSPM simulator. This test execution generates the Actual Output file (AOT). A cross comparator tool then performs the process step 6 (PS6) – i.e., generating the test results report by comparing the EOT and AOT.

5. CONCLUSIONS

This paper describes the model-based test generation methodology used for generating conformance tests for testing the Basic Services Interface (BSI) layer of the Government Smart Card Interoperability Specification (GSC-IS). The model for 23 BSI methods resulted in 306 model paths and the test vector generator generated 692 test vectors to cover those model paths. Similar modeling, model transformation, test generation, object mapping , test schema and test driver generation efforts are underway for developing conformance tests for the Virtual Card Edge Interface (VCEI) layer of GSC-IS.

6. REFERENCES

[1] NIST IR 6887 – 2003 EDITION, "Government Smart Card Interoperability Specification", Version 2.1, July 16, 2003.

[2] Eric Dalci et al, " Setup and Process Guide for Conformance Testing of Government Smart Card Systems, Basic Service Interface, Java Binding, Version 1, Dec 1, 2003.

[3] E.L.Safford. "Key applications of Test Automation Framework (TAF)", Proc 12th Annual Software Technology Conference, April 30 - May 5, 2000.

[4] C.Heitmeyer, J.Kirby,B.Labaw and R.Bharadwaj. "SCR: A toolset for specifying and analyzing software requirements",Proc. 10th Annual Conference on Computer-Aided Verification, Vancouver, Canada, 1998

[5] M.R.Blackburn., R.D. Busser, J.S. Fontaine. "Automatic Generation of Test Vectors for SCR-Style Specifications", Proc. 12th Annual Conference on Computer Assurance, Gaithersburg, Maryland, pages 54-67, June, 1997.