A TTCN-3 Modeling Tool for Automated Testing on Cyber-Physical Systems

Nayun Cho, Janos Sztipanovits, Dugki Min

Abstract

Cyber-physical system is an integrated large-scale system that consists of computational and physical systems. Developing cyber-physical systems incur significant complexities in both computational and physical elements to provide reliable, available, and robust systems. Since computational systems control physical systems via networks, system developers are increasingly facing challenges to ensure that the system satisfies its functional requirements. Software testing is a solution for these challenges to reduce the risk of software failures. Traditional methods for software testing, however, typically rely on manual description of test cases, which is a tedious and error-prone effort. As a solution to this problem, a technique called model-based testing is used that provides a way to automate test generation by virtue of the abstract models of systems. Existing tools of model-based testing, however, have lack of modeling support to design test cases. This paper presents a tool, named AToM, to design test cases using intuitive visual representations for model-based TTCN-3 testing. AToM provides visual modeling capabilities for describing structure and behavior of models and generators for automatically synthesizing consistent test cases from the models. Employing the automated support of AToM, test designers can specify test cases without any manual modifications, thereby ensuring the quality of final results of the testing.

Keywords: Model-based testing; Model-driven engineering; Cyber-physical systems

1. Introduction

Cyber-physical System (CPS)[1] is an integrated large-scale system that consists of computational and physical systems. Since the computational systems control and monitor the physical environment through networks, faults from a single computational component may affect the physical world directly. To prevent faulty situations of life-critical CPSs, such as automobile, aerospace and healthcare, each computational component have to ensure its functionality via software testing process[2]. Currently, there are some activities in this direction, especially from the automotive standard community, called AUTOSAR[3].

The methods of software testing are categorized according to the testing goals or types of target system[4]. For example, black-box testing regards software as a black-box and check functionalities without knowing its internal structure. Unlike black-box testing, white-box testing concerns the internal structures of the software. Each of these methods involves testing processes to describe test cases[5]. A script-based testing, for instance, provides a way of test automation by describing a set of testing instructions that are executing one or more test cases. Also, a keyword-driven testing abstracts low-level test scripts as an action keyword in the test cases. Hence, a tester describes a set of action keywords to examine specific software modules rather than test scripts. However, traditional approaches to develop test cases typically use manual description mechanisms, which often generate error-prone test cases that affect the quality of software. In case of script-based testing or keyword-driven testing, a tester should manually specify a sequence of action keywords.

A solution to address the above problem is to use abstract models of the system, called model-based testing (MBT)[6]. MBT provides a mechanism of test automation by specifying structural models of the system. Especially, with regard to the language for MBT, Testing and Test Control Notation (TTCN-3) is one of standards for black-box testing. TTCN-3 is successfully adapted to the experienced CPS engineers by industry due to its graphical and textual presentation formats[7]. Since the models capture abstract syntax and semantics of the system, the tool that provides modeling of the structure
and behavior of a target system is much needed. There has been some research in developing a model-based tool for TTCN-3, but none of the existing tools fully supports requirements of MBT approach.

A suitable modeling tool for model-based TTCN-3 testing is one that can support (1) domain-specific modeling language (DSML) for human and machine presentations (i.e. graphical and textual presentations), (2) model transformation for bridging the gap of two DSMLs, and (3) automated test generation from the DSML. As we mentioned above, TTCN-3 consists of three presentation formats, graphical, tabular, and textual formats. In this paper, we focus on both graphical and textual formats of TTCN-3. Each presentation form is not separate from the test space, so that TTCN-3 combines presentation formats into textual format, named TTCN-3 core language. Graphical presentation format for TTCN-3 (GFT) presents notations for behavioral invocation of test cases using an extended version of message sequence chart (MSC). However, GFT has limited expressions to provide data aspects, such as declaration of types, templates, and so on. Owing to this limitation of GFT, a tool for model-based TTCN-3 testing requires an abstract model for TTCN-3 core language. Since the behavior and declaration models of TTCN-3 capture different domain semantics, a modeling environment for model-based TTCN-3 testing should provide transformation rules. Moreover, to using these models more efficiently, the tool is required to have the ability to convert the models into another format, such as executable test cases. Having these three qualities offer many advantages, such as a high degree of automation, development cost and time saving by eliminating error-prone test cases.

This paper describes a TTCN-3 tool of AToM (A Tool for Model-based TTCN-3 testing)[8], which is a DSML of TTCN-3, and a related TTCN-3 compiler that executes a generated test case from the TTCN-3 tool. Fundamental of the TTCN-3 tool is a metamodel of TTCN-3 core language, named TTCN-3 modeling language (T3CoreML) that captures backus normal form (BNF) grammar in TTCN-3 specification. Moreover, model-constraints of the TTCN-3 tool help designers to avoid semantic errors at the design time. Consequently, the automated support of the TTCN-3 tool of AToM allows modelers to express the declaration model of TTCN-3 using intuitive visual representations. Moreover, the model can be converted into an executable format to perform black-box testing using model interpreters.

This paper is organized as follows: Section 2 describes the design challenges in designing a tool for model-based TTCN-3 testing. Section 3 presents the design of the TTCN-3 tool of AToM for declaration of test procedures. Section 4 presents the steps to use TTCN-3 tool. Section 5 discusses related research efforts for model-based TTCN-3 testing, and finally, Section 6 presents concluding remarks.

2. Motivations and Design Challenges

Typically, MBT approach starts with describing test scenario using the models and takes it to generate test cases through an interpreter. Applying this approach to TTCN-3, we divide the testing process into three phases: test behavior design, test procedure declaration, and test execution. Test behavior design is based on graphical model of TTNC-3 to specify several statements of test-message invocations, interval logic, and evaluation of the results. GFT concerns not only behavioral sequence of a main test component (MTC), but also internal communication with test components of the target system. However, GFT contains limited expressions for data aspects, such as declaration of types, templates, and so on. Due to this limitation, the test procedure declaration is much needed as a second phase of model-based TTCN-3 testing. In this step, TTCN-3 core language is used to define data aspects of TTCN-3. Subsequently, as the last phase of testing process, test executor compiles the test case and executes a test using test scripts. There are inherent design challenges in providing the above-mentioned testing process:

- **Challenge 1:** To design the test behavior using GFT model, the model must be supported by the tool so that test logics are designed as test cases that can be verified according to semantic constraints of the model.
- **Challenge 2:** Since GFT model cannot cover the declaration of test procedure, the tool must be able to combine the behavioral model into the declaration model that accomplishes test automation.
- **Challenge 3:** In addition, the declaration model must be interpreted by the tool as an executable form that can be used by TTCN-3 compiler.
To address these challenges above, we have designed and developed a designing tool named AToM that provides a domain-specific modeling environment (DSME) for automated model-based TTCN-3 testing. AToM provides the following solutions for these challenges:

**Solution for Challenge 1: Using the model-integrated computing (MIC) paradigm to develop DSMEs for model-based TTCN-3 testing.** In order to support a modeling environment for model-based TTCN-3 testing, AToM offers DSMLs for designing the behavior model and the declaration model of test case. Moreover, we have designed metamodels of GFT and TTCN-3 core language that generates modeling environments for each domain via meta-programmable tool, called generic modeling environment (GME)[9]. To design metamodels of GFT and TTCN-3 core language, we refer to the specifications of TTCN-3. Moreover, the OCL-based graphical and textual constraints within the AToM help test designer to avoid semantic errors at the design time.

**Solution for Challenge 2: Translating the behavior model into the declaration model via model-transformation rule.** Once the behavior model has been described, the declaration field of the test case is needed to define data types, structures or variables. The declaration model of TTCN-3 can then be used to define such types and variables. As we described above, these two domain models capture different syntax, semantics and visualization rules of the target domain. For mapping the behavior model to the declaration model of TTCN-3, a transformation rule is required to convert the model of source domain into target domain automatically. The AToM tool provides a model-transformation rule, called GFT2T3Core, to mapping semantics of GFT model into TTCN-3 core language using graph rewriting and transformation (GReAT)[10]. Also, the GFT specification provides a relation between TTCN-3 core language and the corresponding GFT symbols. For example, a notation of “ActionBox” of GFT can be mapped to an expression of TTCN-3 core language, such as “map”, “unmap”, “connect”, “disconnect”, and so on. In fact, model transformation only concerns how the source model can be mapped to the target models. Obviously, it provides sequencing and conditional rules to translating two different domain models. However, the sequencing of rule is restricted to sequential execution of the rule which means that invocation sequence of the behavior model does not affect to the final result of the transformation. Thus, this model-transformation process can be a problem to generate the correct set of test cases. A solution to address this problem will be discussed in the following description. Using this transformation rule, a test designer can employ the test behavior model to describe declaration model without any manual modifications.

**Solution for Challenge 3: Generating an executable test case from the domain models using model interpreters.** In order to use these test models (i.e. GFT and TTCN-3 core language) more efficiently, such as synthesize test models to execute a black-box test, we need to translate the models into another format like executable TTCN-3 codes. The translation process of domain models into a useful form is called model interpretation that is achieved by a model interpreter. AToM consists of two model interpreters: (1) GFT interpreter, which converts a sequential feature of the GFT model to a text file (i.e. sequence data), and (2) TTCN-3 interpreter, which synthesizes a sequence data from GFT interpreter and the TTCN-3 model to generate an executable TTCN-3 code. Owing to the limitation of model transformation presented above, GFT interpreter captures the invocation order of a MTC object. Consequently, the results from TTCN-3 interpreter automate the manual process of translating the TTCN-3 model. Without this interpretation process of AToM, test designer needs to convert manually which is tedious and error-prone task.

### 3. Design of TTCN-3 Tool

**3.1. TTCN-3 Core Modeling Language (T3CoreML)**

T3CoreML is the core of model in the AToM. It allows defining declaration model of TTCN-3, such as data types, functions, and so on. In the specification of TTCN-3 core language[7], we categorize each expression to represent T3CoreML. The categories include: module definition, import of definitions, basic statements, simple definition, complex definition, type definitions, external actions, configuration operation, communication operations, alternative statements and operations, timer opera-
tions, execution of test case, and verdict operations. The categories include: module definition, import of definitions, basic statements, simple definition, complex definition, type definitions, external actions, configuration operation, communication operations, alternative statements and operations, timer operations, execution of test case, and verdict operations. Each category involves several expressions, which are applicable to define test procedures. The module definition is a fundamental unit of a test case that combines one or more diagrams of TTCN-3. Also, a module can reuse pre-defined functions or templates by importing other modules via import of definitions. Multiple data types, templates, functions, diagrams, and other expressions can define within a module. The expressions belonging to a module can be represented as diagrams, which is a composition of a set of expressions to sub-structure. It includes four types of diagram (i.e. test case, function, altstep, and control diagram) based on a particular purpose through complex definition. Based on a module, the basic statements allow to define branch and loop statements, such as if-else, for, do-while, and so on. Moreover, the simple definition includes expressions which represent grouping a logical structure to a module, signatures for procedure-based communication, communication ports for message and procedure-based communication and constant definition. Type definition contains types of data, variable, component, record, etc. Furthermore, external actions present an action to stimulate a SUT, and configuration operations provide a way to set up and control test components. Since TTCN-3 supports message and procedure-based communications (i.e. unicast, multicast, and broadcast), it can control the access to ports and examine incoming port queues by using communication operations. The set of alternatives (or altstep) specify default behavior or structures using alternative statements and operations. Also, TTCN-3 specifies expected duration of response by timer operations within a module. Consequently, controlling the test case execution and the result of a test case execution can be defined via execution of test case and verdict operations, respectively.

From now, we describe how T3CoreML supports declaration model for TTCN-3 that is useful for test procedure definition phase. A Metamodel of T3CoreML, corresponding to the categories above, involves three parts: (1) Module, (2) Definitions and (3) the Control. Figure 1 depicts a metamodel of T3CoreML. The hierarchical structure of design space can be composed from the TTCNModel, as a root model of T3CoreML.

- **Module**: The Module is a basic unit that uniquely identifies a test case and it includes one or more Definitions or Control models. As we describe above, each Module can be referred through Import connection. For example, if a test case requires to control the execution of test cases, then it should refer particular test cases (i.e. pre-defined test cases) using Import connection within a Control model.
- **Definitions**: This entity contains general definitions of TTCN-3 core language. It should be declared within a specific Module. The Definition class models diagrams (i.e. test case, function, and altstep diagram), types (i.e. data, variable, component, record, and so on), communication operations (i.e. send, reply, call, getreply, etc), verdict operations (i.e. setverdict, getverdict), timer operations (i.e. timeout, read, etc), configuration operations (i.e. create, connect, map, kill, running, etc), basic statements (i.e. if-else, for, do-while, etc), external actions (i.e. action) and alternative statements and operations (i.e. alt, repeat, interleave, etc).

![Figure 1. Metamodel for TTCN-3 Core Language](image-url)
Control: The control handles test execution which can define an order of test cases to be validated. It should be declared in a specific Module as same as the Definitions. Moreover, it must have more than one test-case diagram to execute sequentially. This requirement can be archived by DiagramReferences entity. Same as the other diagrams, types, timer, and external actions can be declared.

To prevent the risk of semantic errors and incorrect features of the models, we define several constraints in the metamodel of T3CoreML using OCL, which checks model constraints at modeling time. In the following section, we describe these model constraints of T3CoreML.

### 3.2. Model Constraints of T3CoreML

The constraints within the T3CoreML metamodel involve graphical and textual constraints to prevent semantic errors at the design time. (1) Graphical constraints are used to impose an invalid association between two entities in the design space. For example, a control diagram requires at least one test case diagram. The associations, such as Import, DefineDefinitions, and DefineControlLogics, should not have a recursive relationship. (2) Textual constraints, which examines invalid attributes, entities from the model. A TTCNModel, for instance, should contain at least one Module entity. Moreover, name of a Module and test-case diagram must be unique. These constraints allow the modeler to avoid building an incompatible and incorrect model to ensuring the quality of testing.

### 3.3. TTCN-3 Test Case Generator

To enable the black-box testing of target system, the designing environment provides a way to translate and generate test cases based on the model of T3CoreML. AToM enables a test-case designer to model the behavior and declaration model, as well as interpretation of T3CoreML. This interpreter consults a sequence data from the behavior model of TTCN-3 to keep internal order of the test scenario, see[8]. Using this information from the GFTML model, the TTCN-3 interpreter generates a text-based TTCN-3 file used by TTCN-3 compilers, like Go4IT[11] so that the TTCN-3 test case can be executed based on the declaration model of TTCN-3. Subsequently, the interpreter traverses the graphical hierarchy of T3Core model. Firstly, this interpreter checks a root model, TTCNModel entity, from the model to examine existence of Module model. As a second step of interpretation, all of the elements of T3CoreML model can be translated correspond to the sequence data. In this way, if the sequence data starts with a MTC instance with a unique identification, the interpreter translates the entity depending on whether it traverses conditional statements at the first. This second step will be invoked recursively until all elements of the graphical hierarchy of T3Core model are terminated. These interpreted texts are then automatically written to a TTCN-3 file with a name of Module entity by the model interpreter.

### 4. Case Study

In this section, we describe the flow of activities performed by a test designer using the TTCN-3 tool of AToM. We apply a simple domain name service (DNS) test to demonstrate in a concrete way our TTCN-3 tool. Basically, the testing process of AToM requires GFT2T3Core phase to translate the behavior model into the declaration model automatically. However, in this paper, we focus on the flow of activities from the TTCN-3 tool within this demonstration. In order to do that, we use results of the behavior model from our GFT tool[8]. This case study concerns a simple functionality of DNS server using message-based communication. In other words, it examines an IP address corresponding to the DNS name. In this case, we can get “pass” or “failed” results based on the following conditions.

- **Failure Condition:** (1) the response from DNS server differs from our expected IP address, or (2) the response time exceeds acceptable thresholds (i.e. 20 second).

**Step 1. Modeling the declaration model of TTCN-3:** A test designer uses the resulting GFT model as the behavior model of a test case to define test procedure, such as data, template, function, and so on. In this case study, we assume that an initial T3CoreML model is generated by a model-transformation rule, named GFT2T3Core. From this model, a test designer can design the declaration model for test execution. In this case study, we model the following entities of the DNS test:
• **Variable**: The data variable can be used to declare basic types with an initial value, user-defined variables, and so on. By using these variables, a test designer can define relevant variables to the behavior model of a test case as shown in Figure 2. In this case study, we model message identification (i.e., identification), enumerated messages (i.e., MessageKind), message types (i.e., Question and Answer), and timer (i.e., replytimer) by using T3CoreML.

• **Data Structure**: In TTCN-3, a tester can define a set of data structures by using record type. It can include either basic types or user-defined data types similar with traditional programming languages, such as C or C++. In this case study, we model a data structure of the message to query an IP address of the target DNS server using T3CoreML.

• **Template**: The template involves a set of instructions to construct a message for sending or to check an incoming message. Based on the pre-defined record type, a test designer can specify a message for sending and receiving. Also, it provides a way to initialize optional fields of record and template fields explicitly or implicitly. Within this case study, we design sending message (i.e., a_DNSQuestion) and receiving message (i.e., a_DNSAnswer), respectively.

• **Port**: Since TTCN-3 provides message and procedure based communication, each message needs to identify the direction of message sending or receiving by port types. The port types can be bidirectional using the in, out, and inout keywords. In the DNS test, we design an inout direction port (i.e., DNSPort) to communicate between a MTC and the target system.

• **Component**: Each test case composes of a MTC, test components of the target system, and their communications via port types. Thus, the direction of the port types should be seen from the component type that presents which ports is associated with a component. In this case study, we model a MTC (i.e., DNSClient) and a test component (i.e., DNSServer) with a port (i.e., DNSPort), respectively.

---

**Figure 2. The Designed Model using T3CoreML**

**Step 2. Checking the model-constraints of T3CoreML**: In order to minimize semantic violation of the declaration model at the design time, some of constraints should be checked using OCL-based constraints. As we described in Section 4, our TTCN-3 tool checks various constraints, graphical and textual constraints. Consequently, the checked declaration model is exported into the TTCN-3 interpreter and is used to generate a test case.

**Step 3. Generating TTCN-3 test case**: The model interpreter in TTCN-3 tool synthesizes the behavior and the declaration model of TTNC-3 for execute TTCN-3 test. As we mentioned in Section 3, this interpreter compares a sequence data from the behavior model of TTCN-3 and the traversed graph simultaneously. The TTCN-3 interpreter generates the declaration fields (i.e., data types, such as charstring, integer, enumerated message type, and so on) with the behavior fields (i.e., the test case and control part). In this case study, the interpreter concerns test behavior of a test case diagram, there by maintaining the sequential features of the test case.
Step 4. Executing TTCN-3 test: By using a TTCN-3 compiler, such as Go4IT, a test designer can draw the conclusion that whether the system satisfies required functional properties or not. Figure 3 illustrates a result of DNS test by using an open-source TTCN-3 compiler, named Go4IT. Figure 3 shows a result that a received IP address from the DNS server corresponds to the value of test case correctly.

5. Related Work

This section discusses some efforts in the area of model-based testing and TTCN-3 testing. There are several existing works to develop a tool for TTCN-3 testing. For example, [12] concerns an integrated tool that synthesizes different presentations in TTCN-3 as a platform independent model (PIM). The primary advantage of this approach is that it improves interoperability of the resulting tools. Moreover, their approach allows users to exchange or access a model data using a common interface. However, this research does not concern domain-specific semantics for each presentation formats as a platform specific model (PSM), like GFT. Lack of domain-specific semantics and its representative models make model-based testing difficult to achieve and use. Also, there are some related works regarding model-based testing for TTCN-3 using unified modeling language (UML). The OMG has adopted UML testing profile (UTP) for design, specify, analyze and visualize the artifacts involved in testing process. These UTP stereotypes provide a way to specify the test specific concepts, such as verdicts, defaults, test components, and so on. Previous researches, such as [13], generate a TTCN-3 code from the structure and behavior model of a test case using UTP models. At the metamodel-level, worked by [14], they investigate mapping rules between UTP and TTCN-3 to address model-to-model transformation that bridging the gap between two different domain models. Also, in [15], they provide an approach to automate TTCN-3 test using UML 2.0 models that depicts high-level abstraction models of the SUT. Unlike the above researches, they generate a test case from state machine diagram without any extension of the UML. These efforts can be used to express test cases using UML, but the lack of domain-specific semantics does not fully support the model-based TTCN-3 testing.

In contrast, our work is complementary to support the model-based TTCN-3 testing. AToM captures the behavior and structural semantics and transforms them into representative models. Moreover, AToM reduces the risk of semantically incorrect models using various constraints that allows test designers to avoid the generation of error-prone test cases, simultaneously. Consequently, after transforming the behavior model into the declaration model, AToM converts the model into an executable test code that significantly reduces the manual efforts.

6. Conclusion

With the growing complexity of CPSs, software testing becomes necessary to reduce the risk of fa-
ults or failures in the computational systems. Conventional techniques for software testing, however, are error-prone and tedious according to their manual nature. To address these challenges, we presented AToM, which is a modeling environment for TTCN-3 testing that resolves the challenges above. AToM provides DSMs for the behavior and declaration models of TTCN-3 that can be converted into an executable test code using model interpreter. Also, GFT2T3Core provides a rule for mapping two DSMs that resolves manual transformation between GFT and TTCN-3 core language. The automated support of AToM allows modelers to express a test case using behavior and declaration model of TTCN-3. Subsequently, the models are converted into executable TTCN-3 code and the output test code can be compiled via TTCN-3 compiler. Also, we presented a representative case study, DNS test, to check the functional requirements of the DNS server. The result shows that our TTCN-3 tool of AToM provides visual representation of TTCN-3 core language and generates a consistent test case taking into account the sequential procedures of a test simultaneously. Consequently, the output of model interpretation is executed by the TTCN-3 compiler that enables test designer to analyze the testing results without any manual interpretation process.

7. Acknowledgment

This research is supported by NRF International Research Collaboration Program (2011-0030918) from National Research Foundation (NRF) of the Ministry of Education, Science and Technology of Korea. This research was supported by MSIP (Ministry of Science, ICT&Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2013-(H0301-13-1012)) supervised by the NIPA (National IT Industry Promotion Agency).

8. References