Test Case Generation from Formal Models of Cyber Physical System

Lichen Zhang*, Jifeng He and Wensheng Yu

Shanghai Key Laboratory of Trustworthy Computing
East China Normal University, Shanghai 200062, China
**Corresponding author: zhanglichen1962@163.com*

Abstract

Formal methods and testing are two important approaches that assist in the development of cyber physical systems. Formal specification can be used to assist testing and Formal methods and testing are seen as complementary. In this paper, we address the problem of generating test cases for cyber physical systems from formal specifications using differential dynamic logic(DL), a logic for specifying and verifying hybrid systems.

Keywords*: Cyber Physical Systems, Formal Methods, test, differential dynamic logic(DL)*

1. Introduction

Cyber-physical systems (CPS) [1], frequently characterized as smart systems include digital cyber technologies, software, and physical components and are intelligently interacting with other systems across information and physical interfaces. They are expressing an emerging behavior and so, create even more *live* functionalities during the deployment. CPS are sensing the external world and they immediately react on the state of the surrounding. Thus, in order to successfully develop such complex systems and to remain competitive on top of that, early and continuous consideration and assurance of system quality is becoming of *vital importance*. Cyber physical systems, due to their increased size and complexity relative to traditional embedded systems, present numerous developmental challenges. The long-term viability of requires addressing these challenges through the development of new design, composition, verification, and validation techniques. These present new opportunities for researchers in cyber physical systems. It is natural to advocate the use of formal techniques in this application area in order to cope with these challenges and indeed a large body of knowledge exists on their use. Formal specifications contain a great deal of information that can be exploited in the testing of an implementation, either for the generation of test-cases, for sequencing the tests, or as an oracle in verifying the tests. In this paper, we address the problem of generating test cases for cyber physical systems from formal specifications using differential dynamic logic (DL), a logic for specifying and verifying hybrid systems.

2. Related Work

A test is a description of which input to apply and when to apply it to the implementation under test, and of which output can be observed from the implementation under test, in order

to decide whether the implementation conforms to its design. A test typically consists of the following steps:

- apply an input action and continue the test;
- observe an output action from the implementation and if an output action is observed that was allowed then continue the test, if an output action is observed that was not allowed then terminate the test with verdict fail, if no output action is observed and no output action is supposed to be observed then continue the test, and if no output action is observed but an output action should be observed then terminate with the verdict fail; or
- terminate the test with the verdict pass.

In [2], Lee, *et al.*, present embedded system software testing based on SOA to overcome mobile restriction. In [3], Tchamgoue, *et al.*, classified the event-driven program models into low and high level based on the type of event they handle, categorized concurrency bug patterns and surveyed existing detection techniques for concurrency bugs in event-driven programs. They believe that such taxonomy can help the developer to understand the causes of concurrency bugs and to avoid introducing them. It can also ease the debugging process, and help developing heuristics for more precise detection tools. In [4], design considerations to realize a cyber-physical testing language that involves software-based and human evaluation agents as test oracles are outlined and discussed.

Quantitative conformance testing of cyber-physical system (CPS) exploits time series of measurements, such as temperature or energy, for validating the correctness of deployed systems. In [5], Matthias Woehrle, *et al.*, presents the foundations of segmented state space traversal in the setting of quantitative conformance testing of a CPS. It is demonstrated how this strategy together with domain-specific adaptations remedies state space explosion inherent to formal (state-based) verification.

In [6], Praveen Ranjan Srivastava and Tai-hoon Kim presents a method for optimizing software testing efficiency by identifying the most critical path clusters in a program. They do this by developing variable length Genetic Algorithms that optimize and select the software path clusters which are weighted in accordance with the criticality of the path. Exhaustive software testing is rarely possible because it becomes intractable for even medium sized software. Typically only parts of a program can be tested, but these parts are not necessarily the most error prone. Therefore, they proposed a more selective approach to testing by focusing on those parts that are most critical so that these paths can be tested first. By identifying the most critical paths, the testing efficiency can be increased.

In [7], Jae-Hee Lim, *et al.*, presents a hierarchical test model and automated test framework for robot software components of RTC (Robot Technology Component) combined with hardware module. The hierarchical test model consists of three levels of testing based on V-model: unit test, integration test, and system test. The automated test framework incorporates four components of test data generation, test manager, test execution, and test monitoring.

Most of the national critical key infrastructure, such as power, piped gas and water supply facilities, or the high-speed railroad, is run on the SCADA (Supervisory Control and Data Acquisition) system. Recently, concerns have been raised about the possibility of these facilities being attacked by cyber terrorists, hacking, or viruses. Thus, it is time to adopt the relevant security management techniques. Sungmo Jung, Jae-gu Song, Seoksoo Kim analyze

the vulnerabilities of SCADA systems through scenarios, designs a test-bed to prove such vulnerabilities, and suggests security devices [8].

In [9], Farkhod Alisherov A., and Feruza Sattarova Y. present a methodology for penetration testing .Penetration testing is one of the oldest methods for assessing the security of a computer system. The idea behind penetration testing methodologies is that the penetration tester should follow a pre-scripted format during test as dictated by the methodology. A penetration testing methodology was proposed in this research.

In [10], Dino Mandrioli, Sandro Morasca, and Angelo Morzent address the problem of automated derivation of functional test cases for real-time systems, by introducing techniques for generating test cases from formal specifications written in TRIO, a language that extends classical temporal logic to deal explicitly with time measures. They describe an interactive tool that has been built to implement these techniques, based on interpretation algorithms of the TRIO language. Several heuristic criteria are suggested to reduce drastically the size of the test cases that are generated. Experience in the use of the tool on real-life cases js reported.

In [11], Rachel Cardell-Olive presents a test generation method for real-time systems. A standard timed transition system model of a network of Uppaal timed automata is transformed under a given test view into a testable timed transition system (TTTS). The test view captures information about a particular set of tests: the selection of relevant events to be observed; the mapping between implementation and specification events; the granularity of the observer's clock; a partition of test events into inputs and outputs. By choosing different test views, the tester can control the number of tests required: more detailed tests can be used for critical test purposes, and less detailed tests elsewhere. A test generation algorithm is presented which constructs a set of test cases from a TTTS. The resulting test suite is input to a test execution harness for the implementation under test. It is assumed that the implementation can be viewed as a TTTS and that it has no more states than the specification TTTS. The final output of the test method is a verdict: yes, the implementation conforms to its specification or no, the implementation fails to conform to the specification because it fails a particular experiment.

In [12], Bahareh Badban, *et al.*, present an automated test generation method for hybrid control systems, which involves the generation of both discrete and real-valued, potentially time continuous, input data to the system under test.

In [13] Moez Krichen & Stavros Tripakis propose a new framework for black-box conformance testing of real-time systems. The framework is based on the model of partially-observable, non-deterministic timed automata. They argue that partial observability and non-determinism are essential features for ease of modeling, expressiveness and implementability. The framework allows the user to define, through appropriate modeling, assumptions on the environment of the system under test (SUT) as well as on the interface between the tester and the SUT.

3. Generating Test Cases from Formal Specification of Cyber Physical Systems Using Differential Dynamic Logic

Much of the process of test execution and monitoring is automated in modern software development practice. But the generation of test cases has remained a labor-intensive manual task. Methods are now becoming available that can automate this process [10–12]. A simple test-generation goal is to find an input that will drive execution of a (deterministic, loop-free) program along a particular path in its control flow graph. By performing symbolic execution

along the desired path and conjoining the predicates that guard its branch points, we can calculate the condition that the desired test input must satisfy. Then, by constraint satisfaction, we can find a specific input that provides the desired test case. This method generalizes to find tests for other structural coverage criteria, and for programs with loops, and for those that are reactive systems (*i.e.*, that take an input at each step). A major impetus for practical application of this approach was the realization that (for finite state systems) it can be performed by an off-the-shelf model checker: we simply check the property "always not P," where P is a formula that specifies the desired structural criterion, and the counterexample produced by the model checker is then the test case desired . Different kinds of structural or specification-based tests can be generated by choosing suitable P. What makes testing hybrid systems difficult is that continuous input and continuous output always occur, that they occur in synchrony and that they depend on each other. If the brake system is stimulated with a steadily decreasing distance, then the brake pressure should increases steadily. It is even possible that the continuous input depends on the continuous output. For instance, if the car brakes, the amount of brake pressure influences how rapidly the distance with the car in front decreases (or increases). Furthermore, discrete output behavior also depends on continuous input behavior and discrete output behavior may have time constraints. For instance, if the distance measurement with the car in front makes a jump (which means a new car is detected), then a "New Car" output has to occur within 0.5 seconds. A test is passed if only expected output is observed (given the input applied) and it fails if an unexpected output is observed. If the brake pressure does not increase while the distance with the car in front decreases (as expected), then the test fails.

The goal of model-based testing, in the form we consider it, is to automate test generation and execution. The behavior of the system is specified by a formal model and tests are automatically generated from this specification. The specification can be a transition system, an automaton, a process algebra term, or a (formal) specification language. Tests are generated by selecting discrete or continuous input from the specification and enumerating the possible observations. A verdict pass or fail is attached to each possible observation in accordance with the specification. Tests are executed by automatically stimulating the implementation with the input described by the test and simultaneously observing the output from the implementation.

The conformance relation formally defines if an implementation conforms to the specification [13]. The test generation procedure defines how tests are generated from a specification. With a formal definition of how a test is constructed and a formal conformance relation it is possible to prove whether our tests are sound and exhaustive with respect to the conformance relation. That is, if the implementation conforms to the specification then the implementation will pass all tests that can be generated from the specification and if the implementation is not conform the specification then it is possible to generate a test which fails.

In this paper, we use differential dynamic logic (DL) to specify cyber physical systems, and generate test cases from formal specifications by differential dynamic logic(DL) [14], and translate the test cases from formal specifications in to the models of Modelica [16] as shown in Figure 1, and finally test cyber physical systems using the simulation through Modelica as shown in Figure 2.

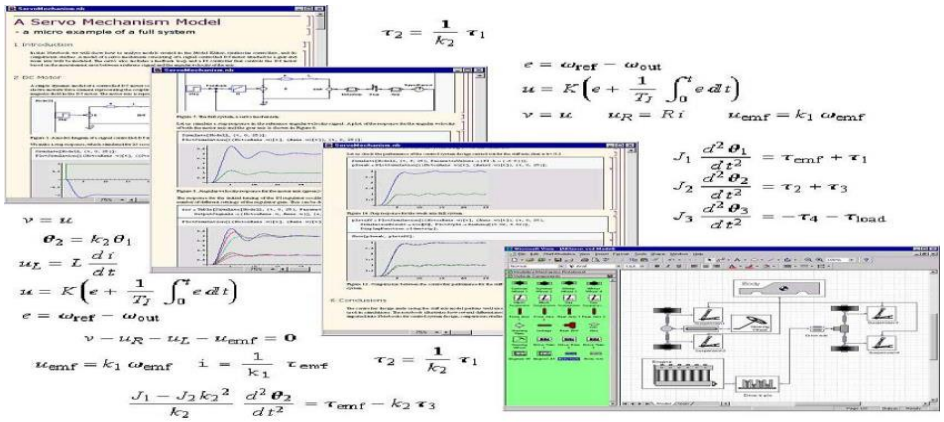

Figure 1. Modelica

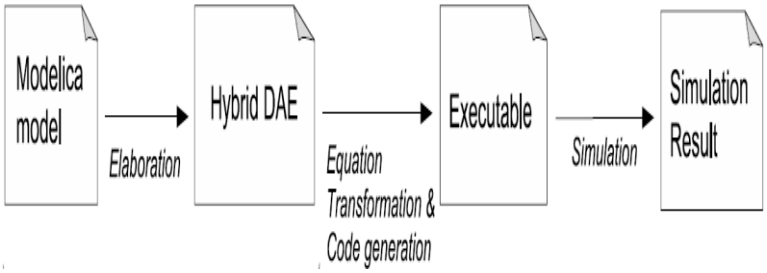

Figure 2. Test steps of Modelica

Modelica [16] is a new language for hierarchical object oriented physical modeling which is developed through an international effort. The language unifies and generalizes previous object-oriented modeling languages. The language has been designed to allow tools to generate efficient simulation code automatically with the main objective to facilitate exchange of models, model libraries and simulation specifications. Modelica is primarily a modeling language, sometimes called hardware description language that allows the user to specify mathematical models of complex physical systems, *e.g.*, for the purpose of computer simulation of dynamic systems where behavior evolves as a function of time. Modelica is also an object-oriented equation based programming language, oriented towards computational applications with high complexity requiring high performance.

Differential Dynamic Logic [14] combines descriptions of system behaviour and correctness statements about the system state within a single specification language. By permitting arbitrary system operationsαas actions of a labelled multi-modal logic, DL provides formulas of the form[α]φand <α>φ.The formula [α]φexpresses that all (terminating) runs of system α lead to states in which condition φ holds, whereas<α>φ expresses that there is at least one (terminating) run of αafter whichφholds. differential logic dL is a first-order dynamic logic with three basic characteristics to meet the requirements of cyber physical systems:

1) Discrete jumps. Projections in state space are represented as instantaneous assignments of values to state variables.

2) Continuous evolution. Continuous variation in system dynamics is represented with differential equations as evolution constraints.

3) Regular combinations. Discrete and continuous evolutions can be combined to form hybrid programs using regular expression operators

For example [15], The car has state variables describing its current position (xc), velocity (vc), and acceleration (ac). The continuous dynamics of the car is described by the differential equation system of ideal-world dynamics for longitudinal position changes ($x_c' = v_c$, $v_c' = a_c$)We assume bounds for acceleration ac in terms of a maximum acceleration $A \geq 0$ and a minimum positive braking power b>0 . We introduce a constant ε that provides an upper bound for sensor and actuator delay, communication between the traffic center or traffic sign detector and the car controller, and computation in both. The car controller1 and the traffic center may react and exchange messages as quickly as they want, but they can take no longer than ε . The Variable speed limit control is modeled by DL as follows:

$$vsl \equiv (ctrl; dyn)^* \quad (1)$$

$$ctrl \equiv ctrl_{car} \parallel ctrl_{ctr} \quad (2)$$

$$ctrl_{car} \equiv (a_c := -b) \quad (3)$$

$$\cup (?Safe_{x_{sl}}; \ a_c := *; \ ?(-b \leq a_c \leq A)) \quad (4)$$

$$\cup (?x_c \geq x_{sl}; \ a_c := *; \ ?(-b \leq a_c \leq A \wedge a_c \leq \frac{v_{sl} - v_c}{\varepsilon})) \quad (5)$$

$$\cup (?v_c = 0; \ a_c := 0) \quad (6)$$

$$Safe_{\underline{sl}} \equiv x_c + \frac{v_c^2 - v_{sl}^2}{2 \cdot b} \quad (7)$$

$$+ \left(\frac{A}{b} + 1\right) \cdot \left(\frac{A}{2} \cdot \varepsilon^2 + \varepsilon \cdot v_c\right) \leq x_{sl} \quad (8)$$

$$ctrl_{ctr} \equiv (x_{sl} := x_{sl}; \ v_{sl} := v_{sl}) \quad (9)$$

$$\cup (x_{sl} := *; \ v_{sl} := *; \ ?(v_{sl} \geq 0 \wedge Safe_{\underline{sl}})) \quad (10)$$

$$dyn \equiv (t := 0; \ x_c' := v_c, \ v_c' = a_c, t' = 1) \quad (11)$$

$$\& \ v_c \geq 0 \wedge t \leq \varepsilon) \quad (12)$$

The continuous dynamics (11) of the model describe the evolution of the car's position and velocity according to the current acceleration. We use a variable t that evolves with constant slope (*i.e.*, a clock) for measuring time within the upper bound ε , and constrain the evolution of velocity vc to non-negative values, see (12).

In the following model, we provide a model for variable speed limit control in the presence of an incident moving towards a car. Cars in this model follow the same control as in the previous section. They take care to comply with speed limits and potentially satisfy or optimize secondary objectives. Accordingly, the lower bound Safesl of the speed limit remains unchanged. We introduce state variables describing an incident's position (xi) and its velocity of movement (vi) towards cars. The system dynamics, are extended with motion of an incident. We also introduce a minimum velocity (vmin), which is often mandatory on freeways and highways, to exclude unreasonable car behavior from the model (*e.g.*, avoid having a car brake to a complete stand still, wait for the incident to arrive at the car's position, just to finally accelerate with maximum acceleration and rush beyond the incident).

Variable speed limit control in presence of static and moving incidents (vsli) is modeled by DL as follows:

$$vsli \equiv (ctrl;\ dyn)^* \quad (1)$$

$$ctrl \equiv ctrl_{car} \parallel ctrl_{ctr} \quad (2)$$

$$ctrl_{ctr} \equiv \text{if } (\neg Alert_c) \text{ then} \quad (3)$$

$$(x_{sl} := x_{sl};\ v_{sl} := v_{sl}) \quad (4)$$

$$\cup (x_{sl} := *;\ v_{sl} := *;$$

$$?(v_{sl} \geq 0 \wedge Safe_{\underline{sl}}) \quad (5)$$

$$\text{else} \quad (6)$$

$$x_{sl} := *;\ v_{sl} := *; \quad (7)$$

$$?(v_{sl} \geq v_{min} \wedge Safe_{\overline{sl}} \wedge Safe_{\underline{sl}}) \quad (8)$$

$$\text{fi}; \quad (9)$$

$$Alert_c \equiv x_f - D \leq x_c + \left(\frac{v_c^2 - v_{min}^2}{2 \cdot b} + \left(\frac{A}{b} + 1\right) \cdot \left(\frac{A}{2} \cdot \varepsilon^2 + \varepsilon \cdot v_c\right)\right) \cdot \left(1 + \frac{v_f}{v_{min}}\right) \quad (10)$$

$$\wedge x_c \leq x_f \quad (11)$$

$$Safe_{\overline{sl}} \equiv (v_f = 0 \wedge x_{sl} \leq x_f) \quad (12)$$

$$\vee \left(v_f > 0 \wedge x_{sl} \leq \frac{x_f \cdot v_{min} + x_c \cdot v_f}{v_f + v_{min}}\right) \quad (13)$$

$$dyn \equiv (t := 0;\ x_c' = v_c,\ v_c' = a_c,\ x_f' = -v_f,\ t' = 1 \quad (14)$$

$$\&\ v_c \geq v_{min} \wedge t \leq \varepsilon) \quad (15)$$

Once DL specification has been created, it is analysed according to D L rules. For the function that the user has indicated, the pre-condition and post-condition are examined, and a partition representing them is generated. Subsequently, the partition is used to produce predicates which need to be passed to an external solver to find out whether they are satisfiable, and, if they are, to choose a random sample from each partition. We assume that the system state is uniquely determined by the value of *n* variables, *{x1; x2,... , xn}*, where each *xi* takes its value from its domain *Di*. Thus, the reachable state space of the system is a subset of $D = D1 \times D \times \ldots \times Dn$. The system may move from one state to another subject to the constraints imposed by its transition relation. The transition relation is a subset of $D \times D$, specified as a Boolean predicate on the values of the variables defining the state-space. We assume that there is a transition relation for each variable, *xi*, specifying how the variable may change its value as the system moves from one state to another. Informally speaking, the transition relation for *xi* can be thought of as specifying three components: a set of *pre-state* values of *xi*, a set of *post-state* values for *xi* and the condition which *guards* when *xi* may change from a pre-state value to a post-state value. We adopt the usual convention that primed versions of variables refer to their post-state values while the unprimed versions refer to their pre-state values.

For Example, Safe bounds test generation: From the viewpoint of a traffic center and an in-car driver assistance system (*e.g.*, obstacle or pedestrian detection), a combination $Safe_{\overline{sl}} \geq Safe_{\underline{sl}}$ is most interesting, since it allows us to derive a minimum distance between an incident and a car that is still safe for braking before meeting the incident. Analogously to above, we define such a safe operating distance in, which indicates the latest distance at

which a traffic center or an in-car driver assistance system must start processing in order to warn about an (moving) incident in time so that the system can react safely.

$$x_i - x_c \geq \left(\frac{v_c^2 - v_{sl}^2}{2 \cdot b} + \left(\frac{A}{b} + 1 \right) \cdot \left(\frac{A}{2} \cdot \varepsilon^2 + \varepsilon \cdot v_c \right) \right) \cdot \left(1 + \frac{v_i}{v_{\min}} \right)$$

Figure 3 shows the test results of the relation of distance between car and event with the car's velocity

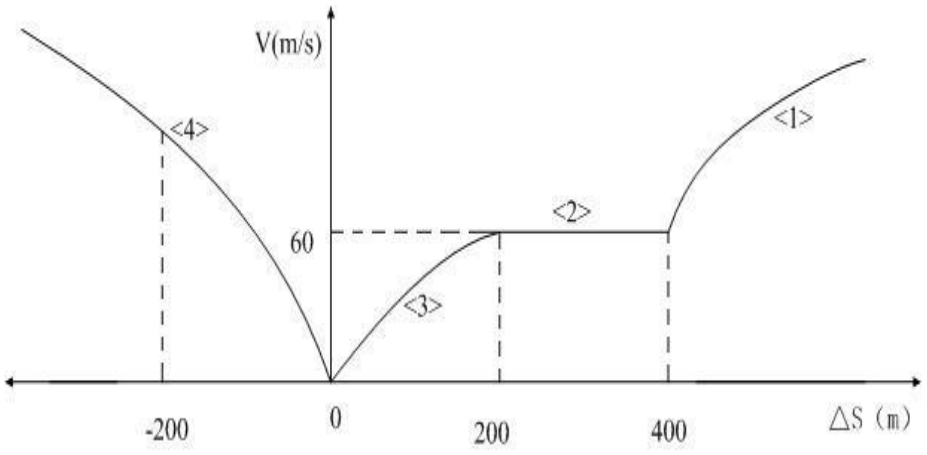

Figure 3. the Relation of Distance between Car and Event with the Car's Velocity

Figure 4 shows the test results of the relation of the car's velocity with the time.

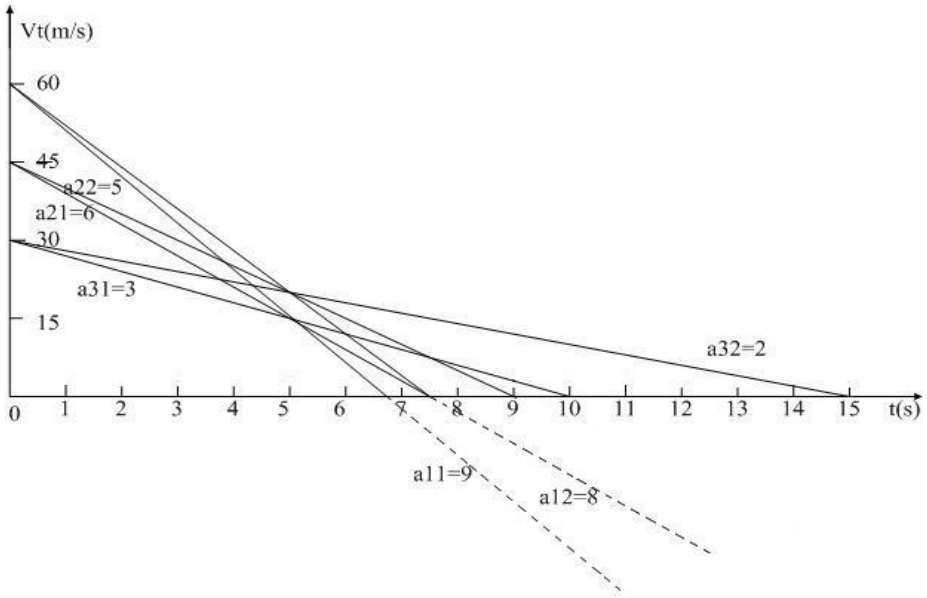

Figure 4. the Relation of the Car's Velocity with the Time

The Table 1 shows the test result of car's safety.

Table 1. The Test Result of Car's Safety

region	distance△S(m)	Velocity Vt(m/s)	braking power (m/s^2)	Car state
〈1〉	400<=△S	∃	∃	safe
〈2〉	200<=△S<400	Vt<60	a=Vt/t	safe
		Vt>60	a=Vt/t	unsafe
〈3〉	0<=△S<200	Known Vo, select suitable Velocity and acceleration	a>=6m/s^2	safe
			a<6m/s^2	unsafe

4. Conclusion

In this paper, we address the problem of generating test cases for cyber physical systems from formal specifications and reduce an infinite set of testing parameters into a finite set.

The further work is devoted to develop the test case generating methods and tools for the verification of the dynamic continuous features of cyber physical systems.

Acknowledgments

This work is supported by Shanghai 085 Project for Municipal Universities and the Innovation Program of Shanghai Municipal Education Commission under grant No. ZF1213, national high technology research and development program of China (No.2011AA010101), national basic research program of China (No.2011CB302904), the national science foundation of China under grant No.61173046, No.61021004, No.61061130541, No.91118008), doctoral program foundation of institutions of higher education of China (No. 20120076130003), national; science foundation of Guangdong province under grant (No.S2011010004905).

References

[1] E. A. Lee and S. A. Seshia, "Introduction to Embedded Systems – A Cyber-Physical Systems Approach", Berkeley, CA: LeeSeshia.org, **(2011)**.
[2] M. -H. Lee, C. -J. Yoo and O. -B. Jang, "Embedded System Software Testing Using Mobile Service Based On SOA", International Journal of Advanced Science and Technology, vol. 1, **(2008)** December, pp. 55-64.
[3] G. M. Tchamgoue, K. -H. Kim and Y. -K. Jun, "Testing and Debugging Concurrency Bugs in Event-Driven Programs", International Journal of Advanced Science and Technology, vol. 40, **(2012)** March, pp. 55-68.
[4] C. Berger, "Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles", Shaker Verlag, Aachener Informatik-Berichte, Software Engineering Band 6, Aachen, Germany, **(2010)**.
[5] M. Woehrle, K. Lampka and L. Thiele, "Conformance testing for cyber-physical systems", ACM Transactions on Embedded Computing Systems, **(2011)**.
[6] P. R. Srivastava and T. -h. Kim, "Application of Genetic Algorithm in Software Testing", International Journal of Software Engineering and Its Applications, vol. 3, no. 4, **(2009)** October, pp. 87-96.

[7] J. -H. Lim, S. -H. Song, J. -R. Son, T. -Y. Kuc, H. -S. Park, H. -S. Kim, "An Automated Test Method for Robot Platform and Its Components", International Journal of Software Engineering and Its Applications, vol. 4, no. 3, **(2010)** July, pp. 9-18.
[8] S. Jung, J. -g. Song, S. S. Kim, "Design on SCADA Test-bed and Security Device", International Journal of Multimedia and Ubiquitous Engineering, vol. 3, no. 4, **(2008)** October, pp. 75-86.
[9] F. Alisherov A. and F. Sattarova Y., "Methodology for Penetration Testing", International Journal of Grid and Distributed Computing, vol. 2, no. 2, **(2009)** June, pp. 43-50.
[10] D. Mandrioli, S. Morasca and A. Morzenti, "Generating Test Cases for Real-Time Systems from Logic Specifications", ACM Transactions on Computer Systems, vol. 13, no. 4, **(1995)**, pp. 365–398.
[11] R. Cardell-Oliver, "Conformance Tests for Real-Time Systems with Timed Automata Specifications", Formal Aspects of Computing, vol. 12, **(2000)**, pp. 350-371.
[12] B. Badban, M. Franzle, J. Peleska and T. Teige, "Test Automation for Hybrid Systems", In Proceedings of the Third International Workshop on Software Quality Assurance (SOQUA 2006), ACM, New York, NY, USA, **(2006)**, pp. 14–21.
[13] M. Krichen and S. Tripakis, "Conformance testing for real-time systems", Formal Methods in System Design, vol. 34, no. 3, **(2009)**, pp. 238–304.
[14] A. Platzer, "Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics", Springer, **(2010)**, pp. 426, ISBN 978-3-642-14508.
[15] S. Mitsch, S. M. Loos and A. Platzer, "Towards formal verification of freeway traffic control", In Chenyang Lu, editor, ACM/IEEE Third International Conference on Cyber-Physical Systems, Beijing, China, **(2012)** April 17-19.
[16] P. Fritzson, "Principles of Object-Oriented Modeling and Simulation with Modelica 2.1", New York, NY: Wiley-IEEE Press, **(2004)**.