Fault Injection for SCADE Models

Xiaolin Shen[1,a], Shaoyin Wang[2], Daqing Wang[3], Xiaoping Xue[1]

[1]School of Electronic and Information Engineering, Tongji University, Shanghai, China

[2]National Maglev Engineering Center, Tongji University, Shanghai, China

[3]Shanghai Metro Technology Center, Shanghai, China

[a]shenxiaolinwufan@126.com

Keywords: Fault injection; Fault-tolerant; ATP; SCADE

Abstract. A fault injection method based on SCADE was designed in this paper, which can simulate hardware failure in the form of software fault. Injecting faults into SCADE models brings forward fault tolerance testing to development phase avoiding the cost of changing fault-tolerant design by modifying the code. Development and fault injection test are in the same environment, which improves the efficiency of the project management. Speed and location measuring subsystem of ATP was selected as the target model in this paper and faults were injected into it in SCADE. The effect on the system of the injected fault can be observed in the simulation process. The weakness of the fault-tolerant design of the system can be found with this method.

Introduction

With the rapid development of urban transportation, the popularity of rail transit has become an important issue to solve the increasingly serious traffic congestion. Communications Based Train Control (CBTC), which has advantages of high speed and high density, has gradually become the first choice of the signal system in the field of rail transit.

Automatic Train Protection system (ATP), which is one of the core subsystems of CBTC, is crucial to ensure the safety of train operation. ATP must satisfy strict requirements of the safety of its software because of the possible disastrous consequences of its failure, including significant financial losses and threat to human lives. To improve the safety of the system, formal software method based on SCADE is used in the development of the ATP software and the development process combined with the corresponding testing process forms the V&V model, which is recommended by EN50128 standards.

ATP is required to have the fault-tolerant ability, which should at least guarantee that the system would not transfer to the failed unsafe state [1] in any situation. However, normal testing cannot test the fault-tolerant ability for hardware failure, as it only aims to test whether the software system can properly execute its function to the requirements in software level. Therefore, we can simulate the hardware failure in the form of software faults and inject the faults into the software system to test the fault-tolerant ability of the system.

The work of this paper is designing such a method to inject faults simulating hardware failure into our ATP system using SCADE. We simulate exceptions of variables and operators caused by various hardware failures such as bit flipping, set types of these exceptions as fault types, and establish fault models called fault nodes in this paper of these fault types using SCADE.

In order to maintain the original structure of the system, fault injection can be implemented by inserting fault nodes into the copy of the system model or replacing the original nodes in the copy of the system model with fault nodes. We run the system with the injected faults in SCADE and set the frequency of the occurrence of the faults, which can simulate permanent, intermittent and transient faults, to test the fault tolerance of the system in the development phase by observing and analyzing the behavior of the system.

The remainder of this paper is organized as follows. The first section is the introduction. The second section briefly discusses some related works. The third section presents the fault injection method based on SCADE models. Simulation and analysis of the fault injection are presented in the fourth section. Conclusions and future work are given in the fifth section.

Related Works

In this section, we mainly discuss the Software Implementation Fault Injection (SWIFI). SWIFI can be classified into high layer and low layer according to the different layers of faults to simulate.

Fault injection in the high layer mainly simulates human errors in software design and implementation, such as code error, data error, variable error, program environment error, etc. [2]. Development of our ATP system makes use of the SCADE suit platform, which is developed by Esterel Technologies providing support of formal modeling and verification. SCADE has integrated code generator, which can automatically generate codes complying with the class A certification of DO-178B. As the code generator can generate codes of high safety level, we need not to consider human errors in coding in our fault injection method.

Fault injection in the low layer mainly simulates hardware failure in the system load, which can inject faults into the bottom of the software such as CPU registers and memory locations and so on. The earliest research in this aspect is FIAT [3] proposed by Siewiorek et al., FIAT can select the physical location of the memory image to inject fault, and disrupt the memory image of the task. Soon, Kanawati proposed a new fault injection tool FERRARI [4], using the Unix system call ptrace to set breakpoints in the target process, can modify the process memory image and CPU registers. Xception [5], proposed by Carreira, using modern CPU registers to trigger the fault injection process, can inject faults simulating faults in memory and registers into the distributed PowerPC system with small intrusion. However, the traditional low layer fault injection is applicable to the fault tolerance testing of the off-the-shelf system.

For safety critical system, if the test result shows that the fault tolerance of the system is not complete, it is necessary to change the fault-tolerant design by modifying the code, which is costly and time-consuming. In our development process, all of design, implementation, testing and verification of the software can be completed using SCADE, so we proposed a method to inject faults simulating hardware failure into the system model in SCADE which can test the improper fault-tolerant design of the system and improve it directly in the development phase using SCADE, avoiding the cost of modifying the code. Using SCADE models to simulate faults makes our fault injection more intuitive. Moreover, development and fault injection test are in the same environment, which will obviously improve the efficiency of the project management.

Fault Injection Using SCADE

Formal model editor, simulator, verifier, code generator, report generator, etc. with graphic interface are provided in SCADE. In the phase of software development, SCADE editor was adopted to establish model of the system, testing in the corresponding phases were implemented by SCADE simulator. Therefore, fault injection in this phase should be implemented with SCADE editor and SCADE simulator.

As many calculations in SCADE model can be completed by variables and operators, we tried to import exceptions of variables and operators as fault types into the system model to test the fault-tolerant ability of the system. These faults are caused by hardware failures such as bit flipping in CPU registers, so we can implement these fault types by simulating corresponding kinds of hardware failures using fault nodes, and the fault nodes can be established using SCADE editor. Time properties of the occurrence of the faults also can be set in SCADE simulator to simulate permanent, intermittent and transient faults.

Fault Injection Location and Rules**.** The normal test procedure operated by simulator in SCADE can be controlled only by external input of the system model. Such test mode cannot be exposed to the internal side of the system and cannot simulate internal fault of the system either. Therefore, faults should be injected into the internal side of the system model in our method. According to our target fault types to inject, faults can be inserted into the location of variables and operators in SCADE model.

In order to increase efficiency of fault injection and independently analyze the influence of the fault at later time, the location of the fault injection should be controlled avoiding injecting equivalent fault to the greatest extent. In addition, fault injection is implemented in SCADE, so the system model with fault should observe the syntax and logic rules in SCADE to pass the examination of SCADE, and thus the model can be established and operated. By combining with the above two requirements, fault injection should be restricted in the following ways:

Rule 1: Fault location: In order to avoid injecting equivalent fault, when the injection location of fault is chosen, attentions should be paid to: the injectable variables should be those which resulted from the predefined SCADE operators, operators imported by SCADE or constant; faults cannot be injected into other indirect variables that can trace back to such variables.

For instance, the SCADE model contains the following relation, where L_1, L_2, L_3, L_4, L_5, L_6 are variables in this model:

$$L_3=L_1*L_2, \tag{1}$$
$$L_4=L3, \tag{2}$$
$$L_6=L_5+L_3, \tag{3}$$
$$L_7=-L_3. \tag{4}$$

Fault that simulates L_3 exception or operator exception can be injected into Eq. 1; it's meaningless to inject fault into Eq. 2, as it only represents the transitive relation of values. It is meaningless to import L_3 exception into Eq. 3, but L_6 exception or operator exception can be injected into it. Fault can be injected into Eq. 4, as it indicates the calculation relation rather than transitive relation of values.

Rule 2: Operator predefined: In order to observe the syntax and logic rules in SCADE, operator predefined in SCADE should be utilized to simulate the fault type of operator exception. Moreover, characteristics of interfaces should be maintained after fault nodes replacing the original nodes; in other words, data types and numbers of the original input and output variables should remain unchanged. For example, if the operator "+" in Eq. 5 to be replaced by "*", Eq. 5 can be directly substituted by Eq. 6, where L_1, L_2, L_3 are variables in the SCADE model.

$$L_3=L_1+L_2, \tag{5}$$
$$L_3=L_1*L_2. \tag{6}$$

According to operators predefined in SCADE and the frequently used data types, substitutable operators and the corresponding types of interface variables are listed in Table 1.

Table 1 Available substitutable operator

In1 type	In2 type	Out type	Available operators
Int	Int	Int	+, -, *, **, DIV, MOD
Int	Int	Bool	>, >=, <, <=, =, <>
Real	Real	Real	+, -, *, /
Real	Real	Bool	>, >=, <, <=, =, <>
Char	Char	Bool	>, >=, <, <=, =, <>
Bool	Bool	Bool	AND, OR, XOR, =, <>

Note that: "=" in SCADE is the operator to judge equivalence.

Fault Injection Model**.** A fault module with fault nodes simulating the target fault types was established in SCADE. And it will be convenient for the management of the fault injection process if a fault injection model could be constructed and fault nodes in the fault module could satisfy the fault injection model.

For fault injection process, a complete model should contain fault location, fault type, fault injection time and fault duration time. Therefore, our fault injection model can be described with the following parameters:

Fault_Injection_Model (<Loc>, <Fault_Type>, <Par1>, <Fault_Freq>, <Par2>, <Par3>). Where <Loc> represents the location of the fault, and each Loc is unique in the target system; <Fault_Type> and <Par1> are used to describe the fault types to inject; <Fault_Freq>, <Par 2> and <Par 3> are used to describe time properties of the fault injection.

Fault types described by <Fault_Type> and <Par1> can be implemented by establishing fault nodes using SCADE editor, where Fault_Type refers to the type of the fault node and Par1 is the parameter used to control the fault node which is input by the tester. Fault types described by Fault_Type and Par1 are shown in Table 2.

Table 2 Fault types and description

Fault_Type	Available variable type	Par1 type	Par1 description
Sw_Operator	Int, Real, Char, Bool	Enum	Represents the operator to replace with.
Flip_Bool	Bool	Null	Parameter not needed.
Flip_Bit	Int, Real	Int	Binary representation of Par1 representing a bit-vector, which is to bitwise "XOR" with the variable.
Set_Bit	Int, Real	Int	Binary representation of Par1 representing a bit-vector, which is to bitwise "OR" with the variable.
Clear_Bit	Int, Real	Int	Binary representation of Par1 representing a bit-vector, all bits of which are flipped and then it is to bitwise "AND" with the variable.
Sw_Value	Int, Real, Char, Bool	Same type as the available variable.	Represents the variable value to replace with.

Parameter <Loc> is appointed by Tri_loc of the fault node like Fig.1 shows, Tri_loc is the parameter in the simulation execution process, _loc points out <Loc>, and the value of Tri_loc is controlled by the fault injection time.

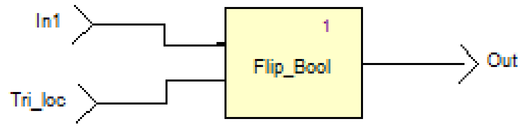


Fig. 1 Flip_Bool node

There is no internal concept of time in SCADE, so time properties can be simulated by the cycle of the simulation process. A cycle is equal to an execution of the simulation; in each execution process, a series of inputs in the test scenario file can be read step by step under the control of the simulator, and a series of outputs can be generated. According to the time properties of true faults, faults can be classified into permanent fault, intermittent fault and transient fault [6].

(1) Permanent fault will always exist in the system once it occurs until the fault is recovered. During the simulation process, extending the fault injection time from the cycle in which the fault first occurs to each cycle after the fault occurs can simulate permanent fault.

(2) Intermittent fault is the transitory fault that occurs occasionally until the fault is recovered. During the simulation process, intermittent fault can be simulated by setting the cycle in which the fault first occurs and the interval period of the occurrence of the fault.

(3) Transient fault is the one that happens temporarily, it occurs just once. During the simulation process, setting the fault occurrence in a cycle can simulate transient fault.

Time properties of the fault injection model described by <Fault_Freq>, <Par 2> and <Par 3> are implemented by the set command *SSM::* in the test scenario file supported by SCADE simulator. Fault_Freq refers to the occurrence frequency of fault, Par2 and Par3 are input by the tester to control the duration time of fault. Table 3 presents the relation among these three parameters.

Process of Fault Injection**.** Fault module with fault nodes was added into the system project in the form of a library, so the testers are able to insert the fault nodes into the location where fault is expected to happen. To simulate different kinds of faults in the software system, fault nodes cover different kinds of variable exceptions and operator exceptions in this paper.

Table 3 Fault frequency and description

Fault_Freq	Par2 type	Par2 description	Par3 type	Par3 description
Permanent	Int	The cycle in which the injected fault occurs.	Null	Parameter not needed.
Intermittent	Int	The cycle in which the injected fault first occurs.	Int	Interval period of the occurrence frequency of the fault.
Transient	Int	The cycle in which the injected fault occurs.	Null	Parameter not needed.

Using test scenario files supported by SCADE simulator, the system model with fault can be driven. Test scenario files here are Tcl scripts. The system model can operate automatically at single step or multiple steps under the control of the simulator. All outputs of the system can be monitored via observer provided in the simulator; besides, internal behavior of the system at each step of execution can be observed which is convenient for analyzing the fault-tolerant ability of the system.

If the system behavior during the simulation satisfies the safety requirements with the injected faults, we consider that the system is fault-tolerant for these faults. Otherwise, we need to modify the fault-tolerant design according to the error behavior of the system to improve the fault tolerance of these kinds of faults until the system become fault-tolerant for all kinds of faults.

For the description of a safety system, there are three states [1]. Operational state refers to the state in which the system executes its functions correctly; failed safe state refers to the state in which the system can no longer execute its functions correctly, but this will not harm to the safety of the system or human lives; failed unsafe state refers to the state in which the system can no longer execute its functions correctly and may hazard the safety of the system and human lives. The safety property requires that fault-tolerant ability of the safety critical system should guarantee that the system would not transfer from operational state to failed unsafe state when fault occurs.

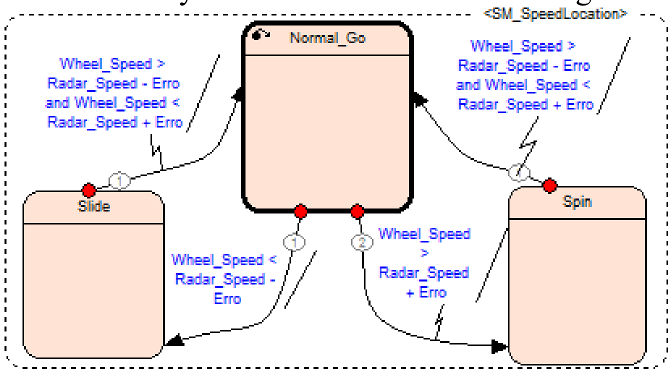
Fault Injection Examples

There are various fault nodes and different combination of parameters available for different system models. All of the fault injection process should comply with the injection rules described in this paper. Here gives two simulation processes and result analysis using Set_Value and Sw_Operator in Table 2 as examples. Simulations have been implemented by injecting fault nodes into a system model established in SCADE to prove the validity of the proposed fault injection method. Illustration of the simulation is also presented in this section.

Speed and location measuring subsystem of ATP is selected as the target system in the simulation. The result of the speed and location measuring system is the input of the core control subsystem of ATP, so it will directly affect the safety of ATP. Speed and location measuring system works with the wheel speed sensor and the radar device, which are two calculation parts of the system. When the train is running, we need to judge the spin or slide operation state of the train and calculate the train speed and travel distance according to the output of the two calculation parts.

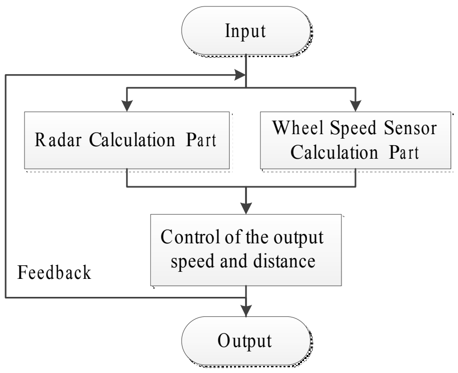
The system model established by SCADE editor is shown in Fig. 2, where Wheel_Speed is the result of the wheel speed sensor calculation part, Radar_Speed is the result of the wheel radar calculation part, Erro is the common calculation error of the radar device, Normal_Go, Slide, Spin are the operation states of the system. We injected fault into wheel speed sensor calculation part and radar calculation part shown in Fig.3 respectively and ran the simulation twice, then we analyzed the system behaviors according to the simulation results.

Failed Safe Situation of the System**.** Assuming that fault occurred in the wheel speed sensor calculation part. To simulate this scenario, we injected a variable exception into this part. In SCADE, we inserted the Set_Value node in our fault nodes library into an output variable, using Par1 to set the output variable value and setting the fault to be intermittent in the test scenario file. When the fault was injected into the system model, we ran the system with and without fault respectively using the same test cases. The operation situation of the fault free system is shown in Fig. 4, and the operation situation of the system with fault is shown in Fig. 5.



<SM_SpeedLocation>
Normal_Go
Wheel_Speed > Radar_Speed - Erro and Wheel_Speed < Radar_Speed + Erro
Wheel_Speed > Radar_Speed - Erro and Wheel_Speed < Radar_Speed + Erro
Slide
Spin
Wheel_Speed < Radar_Speed - Erro
Wheel_Speed > Radar_Speed + Erro
1
1
2
1

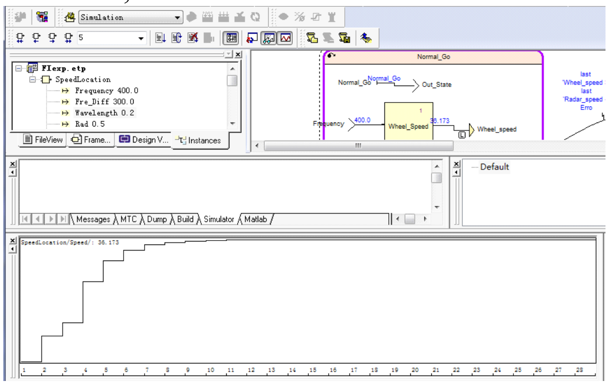
Fig. 2 Speed and location measuring system model



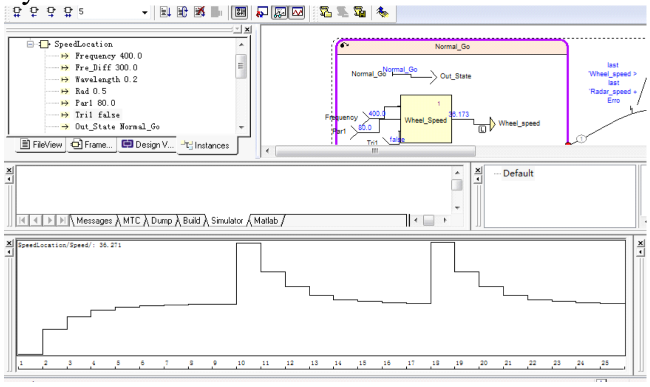
Input
Radar Calculation Part
Wheel Speed Sensor Calculation Part
Control of the output speed and distance
Feedback
Output

Fig. 3 Function description of the model

According to the operation process observed in SCADE simulator, in the cycle with fault, the train's operation state transfers from Normal_Go to Spin and the speed measuring results come from the radar device part. The results are not very accurate compared with the results of the fault free system. In the fault free cycle, despite that the injected fault has interference on the output of the system, the train's operation state can recover to Normal_Go and the output error converges fast. Therefore, we consider that failed safe state of the system is satisfied.


FIexp.etp
SpeedLocation
Frequency 400.0
Fre_Diff 300.0
Wavelength 0.2
Rad 0.5

SpeedLocation/Speed/: 36.173


SpeedLocation
Frequency 400.0
Fre_Diff 300.0
Wavelength 0.2
Rad 0.5
Par1 80.0
Tri1 false
Out_State Normal_Go

SpeedLocation/Speed/: 36.271

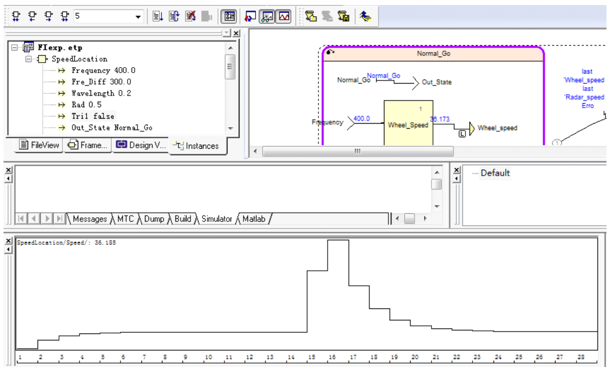
Fig. 4 Operation situation of the fault free system

Fig. 5 Operation situation of the system with variable exception

Failed Unsafe Situation of the System. Assuming that fault occurred in the radar calculation part. To simulate this scenario, we injected operator exception into this part. In SCADE, we replaced the original node with the Sw_Operator node, using Par1 to switch "*" to "+" and setting the fault to be transient in the test scenario file. When the fault was injected into the system model, we ran the system with fault using the same test cases as the fault free system. The operation situation of the system is shown in Fig. 6.

According to the operation process observed in SCADE simulator, in the cycle with fault, the output speed of the wheel speed sensor part is accurate compared with the fault free system. However, as the miscalculation of the radar device part, the train's operation state transfers from Normal_Go to Slide wrongly and the speed measuring results has notable error compared with the results from the fault free system. And the error converges slowly in the fault free cycle. The output error of the speed and location measuring system will lead to the failed unsafe state of the train. Obviously, the fault-tolerant design has not taken faults in the radar device part into consideration. So it is necessary to improve the fault-tolerant design for the radar calculation part.

To conduct a comprehensive analysis of the fault-tolerant ability of the system, fault injection using the proposed method need to combine with the analysis of the functional requirements and safety requirements. Choosing appropriate fault type and injection point will make fault injection test more effective.



```
FIexp.etp
  SpeedLocation
    Frequency 400.0
    Fre_Diff 300.0
    Wavelength 0.2
    Rad 0.5
    Tril false
    Out_State Normal_Go
```

```
SpeedLocation/Speed/: 36.185
```

Fig. 6 Operation situation of the system with operator exception

Conclusions and Future Work

A fault injection method based on SCADE models was proposed in this paper, which can simulate hardware failure in the form of software fault. Then speed and location measuring subsystem of ATP was selected as the target model in this paper and faults were injected into it in SCADE. The effect on the system of the injected fault was observed in the simulation process. Then the weakness of the fault-tolerant design of the system was found by analyzing the system behavior.

Compared with traditional SWIFI method, our proposed method for fault tolerance testing can be used in early development phase to avoid the cost of modifying codes. Development and fault injection test are in the same environment, which obviously improves the efficiency of the project management. All of permanent, intermittent and transient faults can be simulated in our method.

Fault injection in SCADE is restricted in some extent as fault models in SCADE must observe the syntax and logic rules in SCADE which would lead to lacking of the fault types. Research on fault injection in state machine in SCADE models in the future is important. Moreover, it will greatly increase the usable range of the proposed fault injection method to develop more fault types that can be imported into SCADE or supported by SCADE in the future work.

References

[1] E. Cutright, T. DeLong, Johnson B, Generic processor fault model, Technical Report UVA-CSCS-NSE-004 Revision 00. (2002).

[2] J.F. Chen, Y.S. Lu, X.D Xie, Research on software fault injection testing, Journal of Software. 2009, 20(6): 1425−1443.

[3] J.H. Barton, E.W. Czeck, Z.Z. Segall, D.P. Siewiorek, Fault injection experiments using FIAT, IEEE Trans. on Computers. 1990, 39(4): 575−582.

[4] A.K. Ghani, A.K. Nasser, A.A. Jacob, FERRARI: A flexible software-based fault and error injection system. IEEE Trans. on Computers. 1995, 44(2): 248−260.

[5] J. Carreira, H. Madeira, J.G. Silva, Xception: A technique for the experimental evaluation of dependability in modern computers, IEEE Trans. on Software Engineering, 1998, 24(2): 125−136.

[6] X.Z. Yang, Fault and its Form of Expression, in: Fault-tolerant Technique and STRATUS Fault-tolerant Computer, Harbin, 1993, pp. 28-29.

Acknowledgment

This work is supported by the Open foundation of the State Key Laboratory of Rail Traffic Control and Safety (Contract No.RCS2010K007), Beijing Jiaotong University.