Safety requirements-oriented interfaces environment scheme for safety-critical system

Y. Li, J. Guo, Y. Yang, G. Xie & Y. Su
School of Information Science and Technology,
Southwest Jiaotong University, China

Abstract

With the increasing complexity of safety-critical system and component-based development approach is widely used, focusing on the problem that the system safety is affected directly by the interfaces failure of its modules, the disadvantages of current safety scheme are analyzed, and an interfaces scheme is presented to ensure safety on system level. First, according to the interactions between safety-critical system and environment, an environment interfaces failure scenario is defined, then its effects to safety-critical system are analyzed. Second, to ensure the system safety requirements, composing the module and its environment, a safety requirements-oriented interfaces environment scheme is presented, which aims to avoid the interfaces faults that could cause a system failure and provide maintenance information when system violates safety requirements. Third, an identification algorithm to generate the safety requirements-oriented interfaces environment is presented based on model checking technology. Finally, taking the urban rail transit computer based interlocking system as an example, the safety requirements-oriented interfaces environment of a signal module is analyzed with the algorithm implemented by SCADE. The result is completely consistent with the field practical experience, which shows the feasibility and effectiveness of this scheme.
Keywords: safety-critical system, safety requirements, environment interfaces failure, safety requirements-oriented interfaces environment, computer based interlocking.

1 Introduction

Safety-critical system (SCS) is a system whose failure could result in human death, significant property loss or environment damage [1]. Component-based development which composing smaller independently developed components into larger components assemblies is widely used in the complex SCS development. Modules is the basic notion of component and components assemblies, which affects the system safety directly. The interfaces fault of a module can easily cause a system failure. For example, one important cause of the Three Miles Island nuclear event in 1979, is an interface which gathered environment state mistakenly [2]. The identification of relation of module interfaces and system safety can benefit the SCS safety analysis, but since the concurrency, real-time, complexity and variety of interactions with environment, there are many challenges facing this task.

The traditional safety analysis methods, like FTA, ETA, FEAM, OSHA, can be used to deduce the fault modes of modules, but are incapable of analyzing the interfaces behavior. Elmqvist J and Nadjm-Tehrani S presented a safety interfaces scheme (SIS) to guarantee SCS safety on component level. SIS presents a safety condition that SCS should assure in a specific fault scenario, depending on faults mode library. This scheme has been adopted well in Swedish strategic research foundation project and the national aerospace research program NFFP [3, 4]. Elmqvist and Nadjm-Tehrani [5] improved the scheme to analyze residual risks quantitatively. Ying and Xu [6] extended the scheme from single and double faults mode to multiple faults mode. Since faults modes are always accumulated during system operation phase, the exhaustiveness of identification of faults modes will affect the completeness of safety interfaces, and SIS cannot be applied well to guide interface design or system integration before system put into use.

This paper presents a safety requirements-oriented interfaces environment (SRoIE) scheme to ensure SCS safety on system level. SRoIE identifies the interfaces that should be ensured for a specific safety constraints, based on system functional model and safety properties. A algorithm to generate SRoIE is presented, including a refinement algorithm for keeping the interfaces set of SRoIE minimal and critical. At last, a module of urban rail transit computer based interlocking system is taken as an example to show the feasibility of this scheme.

2 Module and environment interfaces fault scenario

Modules gathers environment states through input interfaces, based on desired functional requirements, provides data or control commands to environment as outputs through output interfaces. The correctness of input interfaces will directly affects the compute result and system safety of SCS.

Definition 1. *A module is a tuple:*

$$\Sigma = (I, Q, Q_0, \Delta, T, O) \tag{1}$$

where: I is input interfaces, and the input domain is denoted by V_I; Q is finite states set; $Q_0 \subseteq Q$ is initial states set; Δ is finite labels set; $T \subseteq Q \times \Delta \times Q$ is transition set; O is output interfaces set, and the output domain is denoted by V_O.

Modules keep uninterrupted interaction with environment, and perform its functionality under transition constraints. This process is defined as a composition of module and its environment.

Definition 2. *Let $\Sigma' = (I', Q', Q_0', \Delta', T', O')$ be a module, $\Sigma'' = (I'', Q'', Q_0'', \Delta'', T'', O'')$ be an environment, the composition of module and its environment is denoted:*

$$\Sigma^C = \Sigma' \parallel \Sigma'' \tag{2}$$

where: $I^C = I''$, $Q^C = Q' \cup Q'', Q_0^C = Q_0' \times Q_0''$, $T^C = T' \times T''$, $O^C = O' \cup O'' - O' \cap O''$, $V_I' \subseteq V_O''$.

In general, the failure caused by invalid inputs can be avoid by taking design methods, such as fault tolerance, but the consistency of interfaces and environment states is difficult to detect. The inconsistency is an environment interfaces fault scenario and could cause a system failure.

Definition 3. *Let v_i be an environment state of a module, v_i^f be the input result of v_i, environment interface fault (EIF) scenario is denoted by $\Sigma\left(v_i/v_i^f\right)$, where $v_i, v_i^f \in V_I$, $v_i \neq v_i^f$.*

The EIF represents an inconsistency of inputs and environment states. Two-tuples $(I_i, v_i) \in I \times V_I$ indicates the input of interface I_i is v_i, then an interaction between SCS and environment is denoted by $\mu_i = \bigcup_{k=1}^{num} (I_k, v_k)$, where num is the total interfaces numbers of module. The interfaces whose failure could cause system violates a safety constraint ψ are called key interfaces, denoted by KI^{ψ}.

Definition 4. *safety verification model of SCS is a function:*

$$f : (\Sigma \parallel \Sigma'', v_i, v_o, \psi) \longrightarrow (true, false) \tag{3}$$

where: Σ is a module; Σ'' is the environment of Σ; $v_i \in V_I$ is environment inputs; $v_i \to v_o$ and $v_o \in V_o$; $\psi \in \Psi$ is a safety constraint that SCS needed to be verified.

As different environments employ different safety requirements, safety analysis must consider the concrete environment that the system placed in, Safety verification model verifies that, in Σ'', Σ whether satisfy ψ with v_i and its output v_o or not. $\forall v_i$, $f\,(\Sigma \parallel \Sigma'', v_i, v_o, \psi) = true$ means Σ satisfies ψ in Σ'', denoted by $(\Sigma \parallel \Sigma'') \models \psi$.

3 SRoIE scheme

When SCS is in $\Sigma\left(v_i/v_i^f\right)$, for v_i, v_i^f may both be in input domain, SCS fails to identify these consistency, then performs its function with v_i^f. The safety constraint which is satisfied in normal environment may be violated in EIF scenario.

3.1 Safety in EIF scenario

The environment of SCS includes gathering environment Σ^G and working environment Σ^W, and Σ^W determines the computation result of SCS is safe or not.When input interfaces get data properly, $\Sigma^G = \Sigma^W$. But if the interfaces malfunction: $\Sigma\left(v_i/v_i^f\right)$, $\Sigma^G \neq \Sigma^W$. In this situation, SCS performs its functional requirements based on v_i^f, but Σ^W verifies the safety based on v_i, then equation (3) is changed into:

$$f : \left(\Sigma \parallel \Sigma^W, v_i, v_i^f \to v_o, \psi\right) \longrightarrow (true, false) \tag{4}$$

An underlying risk is posed that SCS performed its functional requirements successfully, and ensured the safety in Σ^G, but Σ^W cannot tolerate SCS outputs, then violates the safety requirements, i.e. if $\exists v_i, v_i^f \in V_I$, $\exists\psi \in \Psi$, $\Sigma\left(v_i/v_i^f\right)$, then $f\left(\Sigma \parallel \Sigma^G, v_i^f, v_i^f \to v_o, \psi\right) = true$, but $f\left(\Sigma \parallel \Sigma^W, v_i, v_i^f \to v_o, \psi\right) = false$.

For this problem, this paper presents a safety requirements-oriented interfaces environment (SRoIE) scheme: not only do SCS need to ensure the functional requirements, but also modules interfaces.

Definition 5. *A SRoIE of a module is a tuple $\langle\psi, CI^\psi\rangle$, where: ψ is a safety constraint; CI^ψ is the core interfaces set of ψ, where $CI^\psi = \bigcup B_i$, $B_i \in KI^\psi$, and B_i is the minimal set in KI^ψ, i.e. $\forall A_j \in KI^\psi$, if $B_i \bigcap A_j \neq \phi$, then $A_j \nsubseteq B_i$, $i, j \in N^+$.*

The core interfaces set is monotone with respect to set inclusion, i.e. $\forall\Lambda_1, \Lambda_2 \in KI^\psi : \Lambda_1 \subseteq \Lambda_2 \Rightarrow (\Lambda_1\ is\ core\ interfaces\ set \Rightarrow \Lambda_2\ is\ core\ interfaces\ set)$. SRoIE presents a minimal interfaces combination which could affect a specific safety constraint, and indicates implicitly a relation between interfaces and system safety. For example, if the key interfaces set of ψ is $\{\{I_1, I_2, I_3\}$, $\{I_1, I_2, I_3, I_5\}$, $\{I_2, I_3\}$, $\{I_4, I_5, I_6\}$, $\{I_5, I_6\}\}$, then the SRoIE is $(\psi, ((I_2, I_3), (I_5, I_6)))$, which implies that if $\{I_2, I_3\}$ or $\{I_5, I_6\}$ fails, ψ will be violated.

Through ensuring the SRoIE, safety in working environment can be ensured, and when a safety constraint is violated, maintenance information is provided that the states of interfaces corresponding to the constraint should be checked to prevent interfaces fault.

3.2 SRoIE algorithm

SRoIE can be deduced through model checking which is widely used in SCS to verify whether system satisfies a specific constraint or not [7, 8]. In model checking framework, the gathering environment and working environment can be constructed in SCS functional model, then the fault tolerant ability of the two environments can be analyzed respectively by equations (5) and (6).

$$R_1 = f\left(\Sigma \parallel \Sigma^G, v_i^f, v_i^f \longrightarrow v_o, \psi\right) \tag{5}$$

$$R_2 = f\left(\Sigma \parallel \Sigma^W, v_i^f, v_i^f \longrightarrow v_o, \psi\right) \tag{6}$$

$\neg R_1 \vee R_2 = false$ means ψ is satisfied in E^G, but violated in E^W. The counterexample μ_n returned by model checking can be divided into gathering environment case μ_n^G and working environment case μ_n^W, $\mu_n^G \cup \mu_n^W = \mu_n$. Meanwhile, $\exists i, j \in N, 1 \leq i, j \leq num, \forall (I_i, v_i) \in \mu_n^W, \exists_1 (I_j, v_j) \in \mu_n^G, I_i = I_j$, and $\forall k \in N, 1 \leq k \leq num$, if $k \neq j$, then $I_k \neq I_j$.

$\exists i, j \in N, 1 \leq i, j \leq num$, $\left(I_i^G, v_i^G\right) \in \mu_n^G$, $\left(I_i^W, v_i^W\right) \in \mu_i^W$, if $I_i^G = I_i^W$, $v_i^G \neq v_i^W$, then $\cup I_i^G$ is the key interfaces set of the safety constraint. SRoIE can be refined through the key interfaces set, and the SRoIE generation algorithm is shown in algorithm 1. Utilizing system functional model and safety constraint set, algorithm identifies the key interfaces set for every safety constraint by $KIGeneration$ algorithm, then SRoIE is refined by $CIRefinement$ algorithm.

Algorithm 1 SRoIE generation algorithm

```
Input: system model $\Sigma$; system safety constraint set $\Psi$.
Output: SRoIE of $\Sigma$: $C$.
 1: Initialize $KI = \phi$, $\Lambda = \phi$, $C = (\phi, \phi)$
 2: while $\Psi \neq \phi$ do
 3:     choose a constraint $\psi_i$ from $\Psi$
 4:     $CI^{\psi_i} = \phi$, $KI^{\psi_i} = \phi$
 5:     while $\left(\left(ki = KIGeneration\left(KI^{\psi_i}, \Sigma, \psi_i\right)\right) \neq \phi\right)$ do
 6:         $KI^{\psi_i} = KI^{\psi_i} \cup \{ki\}$ //key interfaces set of $\psi_i$
 7:     end while
 8:     $CI^{\psi_i} = CIRefinement\left(KI^{\psi_i}\right)$ //core interfaces set of $\psi_i$
 9:     $C = C \cup \left(\psi_i, CI^{\psi_i}\right)$
10:     $\Psi = \Psi / \psi_i$
11: end while
12: return $C$
```

$KIGeneration$ identifies the rest key interfaces based on key interfaces have been identified, which ensures the completeness of identification process for a specific constraint and is shown in algorithm 2.

Since the key interfaces space is huge and redundant, a refinement for SRoIE is necessary, which is algorithm 3 $CIRefinement$. To reduce the computation efforts, the monotone characteristic of core interfaces set has been utilized.

4 Case study

Computer based interlocking (CBI) [9] is a key subsystem in railway signal domain. CBI undertakes a responsibility of arranging routes for trains movements in safety, and needs a strict fail-safe requirement [10, 11]. As the effects of concurrency, conflict, and competition, EIF may easily cause a accident and affect

Algorithm 2 Key interfaces set generation algorithm: KIGeneration

```
Input: identified interfaces set $\Xi$; system model $\Sigma$; safety constraint $\psi$.
Output: key interfaces set $KI^{\psi}$.
 1: Initialize $KI^{\psi} = \phi, i = 1$
 2: Model Checking$(\Sigma, \psi)$ with $\Xi$ work properly
 3: if $\neg(\neg R_1 \vee R_2)$ then
 4:     get the counterexample $\mu_n$
 5:     while $i < num$ do
 6:         if $\eta_i^G \neq \eta_i^W$ then
 7:             $KI^{\psi} = KI^{\psi} \cup \{I_i\}$//update key interfaces set
 8:         end if
 9:         $i++$
10:     end while
11: end if
12: return $KI^{\psi}$
```

Algorithm 3 Core interfaces set refinement algorithm: CIRefinement

```
Input: key interfaces set $KI$.
Output: core interfaces set $CI$.
 1: Initialize $CI = \phi, k = 1$
 2: while $k \leq |KI|$ do
 3:     $ki$ =choose the $k^{th}$ element of $KI$
 4:     $i = 1, tem = ki, temp = KI/ki$
 5:     while $(i \leq |temp|)$ do
 6:         $ti$ =choose the $i^{th}$ element of $temp$
 7:         if $ki \subseteq ti$ then
 8:             $temp = temp/ti$
 9:             $tem = ki$
10:         else if $ti \subseteq ki$ then
11:             $ki = ti$ //$ti$ is a core interface
12:             $tem = ti$
13:             $temp = temp/ti$
14:             $i = 1$
15:         else
16:             $i++$
17:         end if
18:     end while
19:     $SI = SI \cup \{tem\}$
20:     $k++$
21: end while
22: return $CI$
```

the safety and availability of signal system. The identification of SRoIE has a significance for CBI safety design and maintenance.

4.1 Signal opening module

Signal opening is a core functionality of CBI, and its safety directly affects the safety of movements of trains. For example, a line of a subway is shown in figure 1. There is a point $P04$ within the protected section of a downlink signal $X0404$, so the straight route $X0404SR$ and diverging route $X0404DR$ can be set respectively depending on the point position.

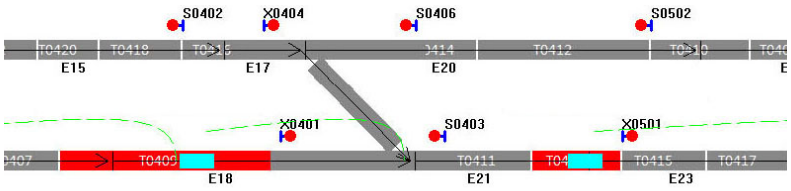

Figure 1: A subway line.

The signal opening module of X0404 Σ_{X0404} involves 27 input interfaces, including section, platform screen door, and point. Σ_{X0404} has four states: (SR_false, SR_true, DR_false, DR_true), and two mutually exclusive processes: $X0404SR$ and $X0404DR$. $X0404SR$ has a initial state SR_false, while $X0404DR$ has DR_false. When corresponding sections are clear ($*P$), platforms has no stop commands($*STOP$), signal has no call-on command($*CR$), platform screen door is safe($*SD$), signal aspect is in safe state ($*SGYA$), depending on the position of $P04$ ($P04SwNC$ for $X0404SR$, $P04SwRC$ for $X0404DR$), Σ_{X0404} transits to state SR_true or DR_true.

4.2 SRoIE identification and result analysis

Σ_{X0404} needs to satisfy the constraint: the straight route and diverging route cannot be set simultaneously, denoted by: $\psi_1 := not\ (X0404SR\ and\ X0404DR)$. This paper utilized SCADE [12] to implement the SRoIE algorithm, and 109 key interfaces have been identified, for example ($ST0402P$,$P04SwNC$,$P04SwRC$), ($P04SwNC$,$X0404SD$,$P04SwRC$), ($P04SwRC$,$X0404SGYA$,$P04SwNC$, $042SD$). Then the SRoIE is refined: (ψ_1, ($P04SwNC$, $P04SwRC$)), which indicates:

1. $\left(\Sigma_{X0404} \parallel \Sigma^W\right) \models \psi_1$, if the interfaces $P04SwNC$ and $P04SwRC$ have been assured;
2. When $\left(\Sigma_{X0404} \parallel \Sigma^W\right) \not\models \psi_1$, the states of $P04SwNC$ and $P04SwRC$ should be checked.

The safety interfaces identified through field experiences includes $ST0402P$, $T0414P$, $P0414.0402SwNC$, $09SD$, $P04SwRC$, etc, including the two core interfaces identified by SRoIE for ψ_1. Meanwhile, the rest of experience interfaces are identified at other safety constraints. The SRoIE result is consistent with the practical engineering experience, and the identification comprehensiveness is ensured. Moreover, the CBI interfaces have been associated with specific safety constraints respectively, and the faults modes that CBI should be ensured are suggested implicitly.

5 Conclusion

This paper divided the SCS environment into gathering environment and working environment. To ensure the safety in the two environment on system level, a SRoIE scheme is presented. Based on the system model and safety requirements, SRoIE scheme presents interfaces environment that a specific safety constraint should be ensured. The interfaces information can be used to improve the design or integration of modules, and guide system maintenance. SRoIE scheme has a guiding and practice significance for the design, development and maintenance of SCS.

Acknowledgements

This work is partially supported by the China Railway Corporation under Grant 2015X009-D and 2014X008-A. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

References

[1] Knight J. C. Safety critical systems: challenges and directions. Software Engineering, 2002. ICSE 2002. Proceedings of the 24th International Conference on, pp. 547–550, 2002.
[2] U.S. NRC. Backgrounder on the Three Mile Island Accident. http://www.nrc.gov/reading-rm/doc-collections/fact-sheets/3mile-isle.html
[3] Elmqvist J. & Nadjm-Tehrani S. Safety-oriented design of component assemblies using safety interfaces. Electronic Notes in Theoretical Computer Science, 182: pp. 57–72, 2007.
[4] Elmqvist J., Nadjm-Tehrani S. & Minea M. Safety interfaces for component-based systems. Computer Safety, Reliability, and Security. Springer Berlin Heidelberg, pp. 246–260, 2005.
[5] Elmqvist J. & Nadjm-Tehrani S. Formal support for quantitative analysis of residual risks in safety-critical systems. High Assurance Systems Engineering Symposium, 2008. HASE 2008. 11th IEEE, pp. 154–164, 2008.
[6] Ying Liu & Xu Zhongwei. Safety interface scheme for component-based safety critical software. Journal of Computer Applications, 28(11): pp. 2933–2935, 2008.

[7] Clarke E. M., Grumberg O. & Peled D. Model checking. MIT press, 1999.
[8] Baier C. & Katoen J. P. Principles of model checking. Cambridge: MIT press, 2008.
[9] Hui M. A. Formal Modeling and Verification of Urban Rail CBTC Interlocking Based on CSP. Beijing Jiaotong University, 2013.
[10] Haifeng Wang, T. H. Xu *et al.* Novel Online Safety Observer for Railway Interlocking System. The Journal of Transportation Engineering ASCE, 139(7): pp. 719–727, 2013.
[11] Yang Yang & Shaowen Zou. Design and Development of Computer-based Interlocking System SWJTU-II. Journal of the China Railway Society, 27(3): pp. 118–123, 2005.
[12] ESTEREL Technologies. SCADE Suite. http://www.esterel-technologies.com/products/scade-suite/