Safety-Oriented Software Architecture Design Approach

Yuling Huang

School of Electronic and Information Engineering, Tongji University, Shanghai, China

halinhuang@163.com

Keywords: Software Architecture; Software Safety; Safety Requirements; Safety Tactics

Abstract. With the increasing attention of software safety, how to improve software safety has already become a more important concerned issue, especially for the safety-critical systems. Currently, the influence of architecture in assurance of software safety is being increasingly recognized. Safety design at architecture level can effectively improve software or system safety. This paper focus on how to consider safety in software architecture design phase and proposed a safety-oriented software architecture design approach. Through the system hazard analysis, this design approach uses the selected combination of safety tactics to effectively improve the software or system safety, providing a new way of thinking for software safety architecture design.

Introduction

Safety Critical System (SCS) refers to the system which has potentially destructive power. Once such a system produced a failure, many serious consequences may be caused, such as casualties, property loss and environmental damage etc. In recent years, software application in SCS is more and more extensive, and the scale also increasingly grows. From railway transit field to the aerospace field and from the power system to the medical system, this type of software plays a key role in command and control aspect for software safety. The core research in SCS safety is how to reduce the probability of unsafe system conditions that various SCS elements lead to, or weaken the SCS's consequences that failures produce, through using a variety of management, organization, technical measures [1]. This is also the focus of this paper.

Since software architecture (SA) was proposed, it has been favored by many researchers and practitioners, and become an important research field of software engineering. SA determines a system's main structure, macroscopic properties, its basic functions and characteristics. SA is the basis and the key to success of the whole software design. Currently, SA design methods mostly focus on system's non-functional requirements (NFR), for instance safety or reliability, into consideration. SA design combining with SA analysis method, researchers hope to find the potential risk in early software life cycle [2]. The research on safe SA not only can ensure the safety of SA-based software development components, but also can ensure the safety of the final software product. Software safety and reliability have different characteristics. The former pay more attention to software defect that lead to enter the unsafe state and how to reduce the risk of system failure which can led to the catastrophic accidents. So far, however, the work combined safety and SA design is very scarce. So, there is little guidance on how to address safety concerns in shaping a safe SA and how to improve SA by using basic safety tactics. In order to improve this situation, this paper proposed a safety-oriented software architecture design approach.

Related Works

Currently, many studies have showed that through explicitly presenting SA and its elements, such as components and connector, SA can handle high abstract level's design problems, such as

the global organization and control structures, functions allocated to the calculating unit, high-level interactions between calculating units or other design problems [3]. According to safety standards of software elements under design, IEC 61508 provide some general guidelines on the selection of these techniques [4]. However, this guidance fails to illustrate further how to utilize these techniques to maximize the protection against failures. Reference [5] classifies the existing conventional SA design methods into five categories, which are Artifact-Driven, User-Case-Driven, Pattern-Driven, Domain-Driven and Requirement-Driven, and respectively describes each kind of design method's design process, design advantages and defects. For SCS architecture design, its safety assurance needs to be satisfied as well as its functional requirements. The five categories design methods are too general because of lack of practicability and concreteness.

Reference [7] tells us how to combine the SA with safety model so as to improve the consistency between safety analysis and software development. By using UML modeling tools for SA design, it elaborates how to convert software architecture model into safety analysis model through using FTA (Fault Tree Analysis) and FMECA (Failure Mode Effects and Criticality Analysis) methods, which are two well-known safety analysis methods with tool support. Therefore, on one hand, it is feasible to take safety into account in the SA design phase. On the other hand, the reuse and traceability of safety analysis is very important for both architects and safety engineers.

Reference [8] defines an analytic safety-attribute model. The focus of this model is the relationship between the safety attribute and SA with respect to failures. There are four key elements which need identifying in this model: failure classification, failure cause, failure behavior and failure property. The contribution is to propose a description of safety tactics framework and summarize the existing safety tactics, as shown in Fig. 1.

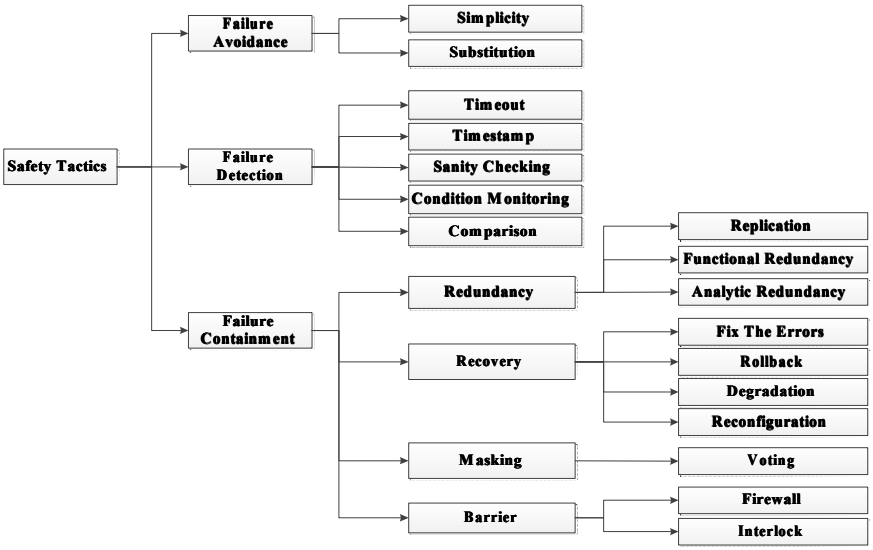

Fig. 1 A hierarchy of safety tactics

Reference [9] proposed a decision-oriented architecture design approach that based on the decision-abstraction and issue-decomposition principles specific to the architecture level design of software. By using the architecture design decisions' metamodel which consists of issues, solutions, decisions and rational, the design decisions' model can be established. This approach achieves to implement the semi-automated candidate architecture solutions synthesis, and the decision-making rational in SA design phase can be a reference for the acquisition of decision rational in implementation phase. Reference [10] proposed an architecture trade-off analysis method (ATAM),which a kind of evaluating software or system architecture method from the perspective of assessment for quality attributes such as safety, reliability, modifiability and usability etc. This

method takes system target and user's view for the input, in view of the architecture analyzing quality attributes that may threaten a system function, utilizing the risk, sensitivity and equilibrium existing in the structure of scene technical analysis. Repeat the design process "Design-Assessment-Modify" until a balance between multiple quality attributes.

In this paper, based on the basic idea of reference [8, 9, 10], we proposed a new safety-oriented software architecture design approach for safety-critical system. This method provides practical guidance on how to use the safety tactics. Combined with FTA method, the safety analysis for safety-critical system can be done in a top-down way in the architecture design phase.

Safety-Oriented SA Design Approach

Fundamental Idea**.** The safety-oriented SA design approach proposed in this paper focuses on safety by using combination of safety tactics in Fig.1. Some conventional software architecture design methods almost directly associate the crosscut form of components and connectors with system safety design which added to the basic structure. Because there is no simple and standard consideration of design arrangement that can meet system safety, it makes design process becomes more complicated. In addition, it blurs the roles and boundaries of components and connectors because of adding non-standardly safety design solutions, which makes design results are very difficult to understand. Therefore, it is the key to design safe software that using the divide-and-conquer strategy when constructing safe architecture of software system.

This approach is based on a hierarchical system model. The first step is to analyze the software requirement belong to one layer, and then according to the requirements specification, we should elicit functional requirements and safety requirements. However, not all the safety requirements can be individually extracted, some safety requirements are often reflected in the functional requirements. After obtaining the safety requirements through preliminary hazard identification, the second step is to conduct preliminary hazard analysis. At the same time, deriving and preliminarily selecting the feasible safety tactics. The third step is to further refine and classify these safety tactics. By analyzing the severity caused by the system or architecture elements failure, safety metrics should be determined. The fourth step is through analyzing the fundamental protection mechanism of safety tactics and the safety metrics, the feasible safety tactics will be filtered again, and the architecture should be constructed by choosing suitable safety tactics combination. At last, we should check whether the safety tactics combination can meet the safety requirements to control the software failure within an acceptable level of risk. If it cannot meet the above condition, then we should redesign the selection of safety tactics. Through repeating the "design- assessment-modify" design process, we can make the architecture meeting every layer's safety metrics for the safety requirements. The safety-oriented SA design approach is shown in Fig. 2.

Safety Tactics**.** The safety tactics which can be combined with the existing architecture or design pattern are fundamental design decisions. It plays a good supplement and guiding role for the choice of architecture design. Tactic is a new design concept, and the existing tactics are far from comprehensive. It is still in the development phase.

The safety tactics are divided into three groups in reference [7], which are respectively failure avoidance, failure detection and failure containment. At first, like other quality attributes tactics, safety tactics can exist at various levels of abstraction. For instance, redundancy can be refined into three different forms: replication, functional redundancy and analytic redundancy. Safety tactics in each group are divided in detail, as shown in Fig.1. Secondly, safety tactics can be implemented by either software or hardware. In many cases, an architect has a choice between software and hardware tactic schemes. For an example of voting, it can be implemented either in hardware or in

software. For a hardware-implemented voter, it may only require a combination of simple single-bit circuits. However, for a software-implemented voter, it has its own costs in terms of development efforts, speed of operation or an increase in system complexity. The primary considerations are always based on a trade-off among cost, performance and complexity. Thirdly, safety tactics can be combined. Generally, each safety tactics can only address one specific aspect of safety. Safety tactics, however, can be combined to address multiple aspects of safety.

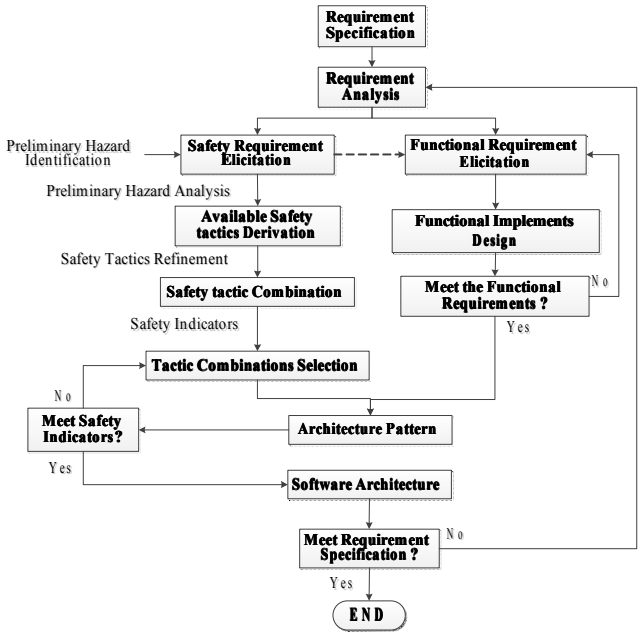

Fig. 2 Safety-oriented SA design approach

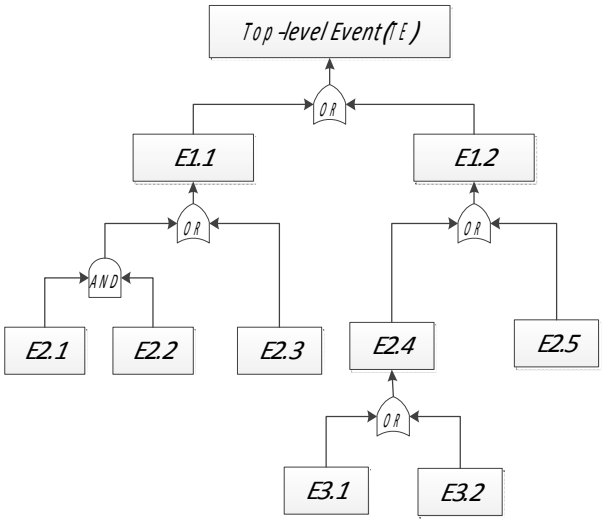

Fig. 3 FTA Model

The safety design of SCS can mainly be divided into structural design optimization and fault-tolerant design. The former aims to reduce software defects, while the later aims to prevent a number of the known software failure. Safety tactics can combine with the existing safety design technique to effectively guide the selection of protective mechanisms to improve the safety.

The derivation of safety tactics requires certain rules. For Safety Critical System, architect should make the safety objectives first, and then have a strong persuasive evidence that the system can achieve high safety rather than describing how it will achieve the objectives. Thus, the first step to extract the available safety tactics is to determine the safety objectives. Secondly, according to these safety objectives, we choose preliminarily from the three tactic groups. For example, some failures which cannot be avoided, we shouldn't choose the tactics from the group of failure avoidance. Thirdly, every tactics of the selected groups shall be further analyzed in detail based on these safety objectives combined with structural design optimization and fault-tolerant design techniques to derive the available tactics. It should accord to the concrete design requirements to judge whether the safety tactic is available. The real-time system, for instance, cannot choose the rollback tactic but can choose the degradation tactic from the failure containment group.

Safety Analysis**.** In the architecture design process, safety analysis is a very important part. Currently, FTA (Failure Tree Analysis) and FMEA (Failure Mode and Effect Analysis) are two examples of well-known safety analysis methods with tool support. In this paper, we use FTA to analyze safety in our proposed SA design approach.

FTA, which is a top-down system analysis process, is a deductive method of analysis. The goal of FTA is to find the minimal combinations of basic events leading to the top level event. In other words, one wants to compute the minimal cut sets of the fault tree. Minimal cut sets are combinations of events leading to a system-level failure when happening together. The root of the

tree is called top-level event, and represents an event that should not occur in a safe system. Analysts begin to analyze safety from a top-level event. Then a single failure event or failure events' combinations, which may cause the top-level event, should be determined in the lower level hierarchy. We don't continue down the analysis until the primary event or events combinations have been fully affirmed. These primary events are leaves in a failure tree. They can be internal or external coming from system, and could be due to hardware failure or software error.

There is an example shown in Fig.3. In this example, TE is a top-level event. It may be triggered to happen when event E1.1 or E1.2 happens. But the event E1.1 and E1.2 are not leaves of this failure tree. They can also be triggered to happen by other events. In this example, the primary events are E2.1、E2.2、E2.3、E2.5、 E3.1 and E3.2. It is important to note that event E2.1 and E2.2 need to occur at the same time, which may cause the top-level event. Assume that among these leaves, the event E2.1, E2.2 and E2.5 occurs easily, and they are also likely to cause the top-level event to occur. And the other failure events occurrence probability is very small. Then we need to take different mitigation means to reduce their occurrence probability so as to improve the system safety. Because event E2.1 and E2.2 need to occur synchronously to trigger the top-level event, on one hand, we can consider how to reduce the synchronous occurrence probability. On other hand, we should consider the consequences when the two events occur separately. Which consequences are more serious, we need to pay more attention on the related event. The specific mitigation means to reduce failure rate can refer to the safety tactics in Fig.1. By using FTA, architects find out the primary event that easily causes the top-level event to improve the architecture design, and the safety of the whole system will be enhanced.

Application Example

Software safety design process always starts with the system or platform hazards identified by preliminary hazard analysis. We take railway transit ATP system as an example to illustrate the derivation of the safety tactics.

ATP System. According to the definition of the CBTC system [11], ATP is Automatic Train Protection system, which includes the follow aspects for the safety protection function: Train self-checking, Speed measure and locate, Speed supervision, Train safe stopping, Safe direction and door control, CBTC operation mode and Runaway protection. Fig.4 shows the ATP system function level division. Fig.5 is a logical relationship between the functional modules. Assuming that is the initial system architecture design.

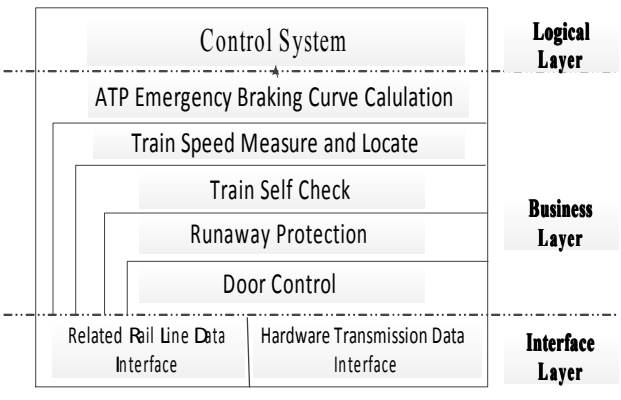

Fig. 4 ATP system hierarchy model

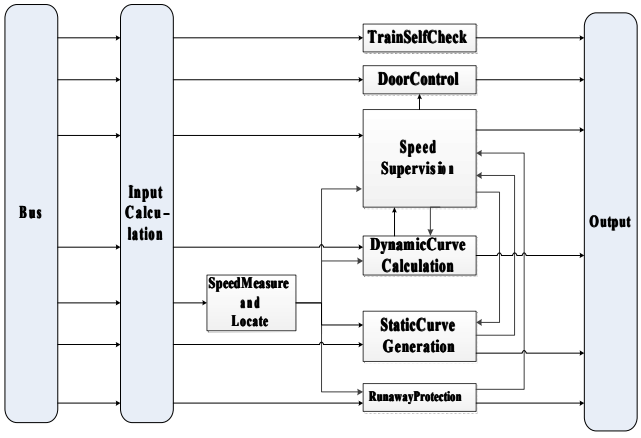

Fig. 5 The control module of ATP system

Safety Tactics Derivation**.** This approach is based on hierarchy mode to analyze and derive

one layer's safety attributes. We take the logical layer of ATP system as an example.

First, we assume that the train are taking unmanned automatic driving mode, the logical layer are the control system that consists of monitor module and control module. We start from preliminary hazard identification through requirements analysis. Here we considered only one of the main hazards arising from ATP system: Train obstacle detection and/or braking cannot operate correctly. The design process should precede by defining the safety of requirements into a lower level. That is ATP system shall detect obstacle and stop in time. Further refinement of the system safety function requires the braking function's allocation to hardware or software. As it takes unmanned automatic driving mode and train itself constraints don't allow to modify the hardware design, we can't use simple hardware protection mechanism. Therefore, we extract the software safety requirements from the design process: control module should be able to output commands in time in response to obstacle detection. After weighing the software design and hardware design, Fig.6 shows the top-down process of safety requirements refinement.

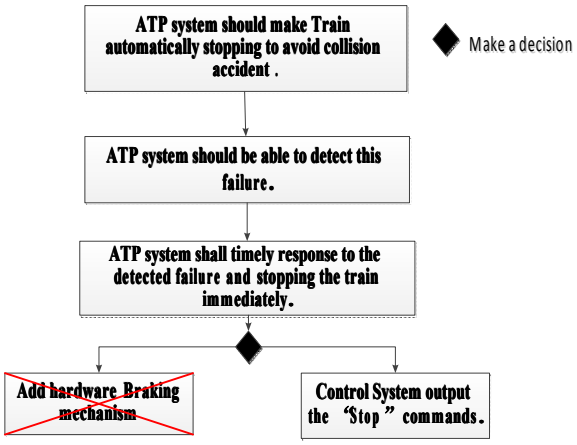

Fig. 6 Safety requirements refinement

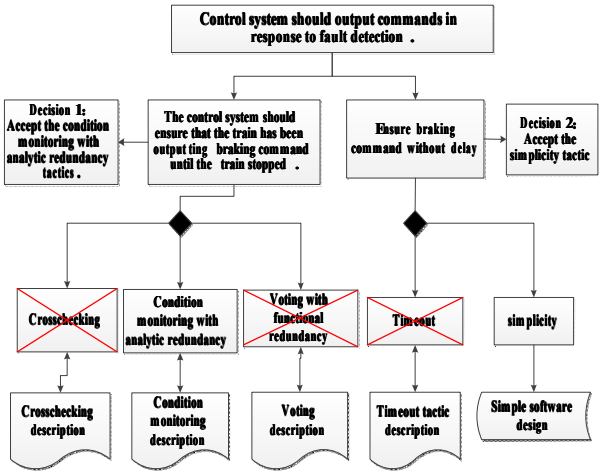

Fig. 7 One solution of safety tactic combinations choice

Then we analyze further to refine the safety requirements. The control system should always output braking commands in response to detection. Since the final state of train is stopping, tactics must be selected form the categories of failure detection and failure containment. Obviously, time-out and timestamp tactics are irrelevant in this case. The sanity-checking tactic is also not suitable because there's nothing to act to make the train stopping. Moreover, the recovery tactics are also rejected simply because any recovery action can't immediately stop the train. Finally, the barrier tactics are also inappropriate candidates since they are used as passive protection mechanisms. Thus, possible safety tactics would be condition monitoring, comparison, voting and redundancy. It is worth noting that redundancy can't be applied alone. Actually, the implementation of the previous three tactics requires some form of redundancy tactics. In other words, the possible tactic choices would be: 1) Condition monitoring with analytic redundancy. 2) Comparison with functional redundancy (crosschecking). 3) Voting with functional redundancy. In order to meet the timing requirement of stopping function, we identified two possible tactics: simplicity and timeout. After identified these possible tactics, we need to choose the most suitable safety tactic combinations, and the reasons of the choice should also be recorded so that it can be understood easily by development engineer in the implement phase. In addition, before making a choice, architects should make descriptions of these safety tactic combinations, so as to help the designer to refer to these descriptions when making decisions. Fig.7 shows one solution of architecture design by choosing the condition monitoring with analytic redundancy. Fig.8

shows the description of the condition monitoring tactics.

Finally, according to the selected safety tactics, architecture pattern could be constructed, analyzing whether the architecture meet the expected safety requirements. If it fails to meet, then architects repeat the "design –assessment- modification" process, until the safety of architecture achieving an acceptable risk level. Here to explain, the architecture design shown in the Fig. 9 is only one scheme and not necessarily the most perfect solution. Through comparing the evaluation results of several schemes, designer can get the optimal architecture design.

In addition, the InputCalculation module of Fig. 4 is responsible for all the calculations and analysis of the input data received from the bus, which verify whether the data is required and effective or not. Obviously, this module is very important in the whole system. However, from the initial architecture design, it can be seen that once the module is failure, it will bring serious influence to the whole system. Thus, this module is the primary event in the FTA model, and needs designers to focus on. Therefore, we modified the module design still according to the above selection process. By taking the comparison and functional redundancy tactics, Fig. 10 is a solution after modified.

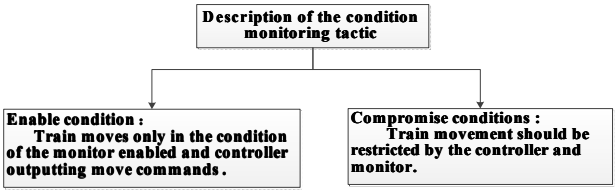

Fig. 8 Description of the condition monitoring tactic

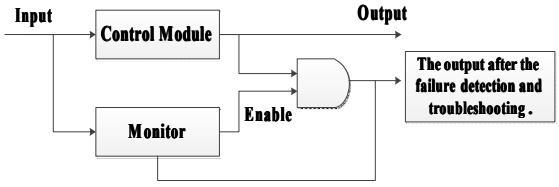

Fig. 9 The initial design of ATP Monitoring System

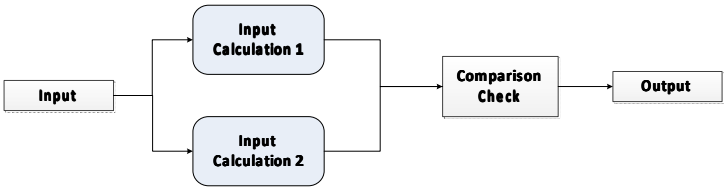

Fig. 10 The modified InputCalculation module.

Conclusions and Prospects

Software safety assurance refers to a series of quality assurance activities during software development life cycle, which aims to eliminate the potential dangers. It is very significant to study how the non-functional attribute "safety" to be described, analyzed and verified during the architecture construction process. This paper proposed a new software architecture design approach, which is on the basis of the hierarchical model and takes safety tactics' combinations to improve the safety of SA solution. This design approach provides a practical guidance and a new way of thinking for safe software architecture design. Through carrying out appropriate safety analysis, software defects that can reduce the system safety would be found out and eliminated or reduce its risk to be an acceptable level. However, the analysis and extraction of requirements relies on the architects' experience and technology. From this point, if the requirement analysis is imperfect, then the architecture design is likely to exist design defects. Moreover, we repeat the "Design-Assessment-Modify" process during the whole architecture design process in this approach, however, the safety assessment indicators' allocation are not elaborated too much in this paper, because this is not the focus of this study. Future research can be extended here.

References

[1] X. G. Fan, W. K. Zhu and F. M. Zhang, “Software Safety Research Review,” Computer Science, Vol. 38, No. 5, 2011, pp. 8-13, 27.

[2] M. Hong and J. R. Shen, “Research Progress on Software Architecture,” Journal of Software, Vol. 17, June 2006, pp. 1257-1275.

[3] D. Garlan and M. Shaw, “An Introduction to Software Architecture,” World Scientific, Vol. 1, 1993.

[4] IEC, “61508-Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems”, International Electrotechnical Commission, 1998.

[5] Y. B. Wang and X. K. Li, “Research on Software Architecture Design Method,” Computer Engineering and Design, Vol. 26, No. 3, 2005, pp. 604-607.

[6] M.A. de Miguel, J.F. Briones, J.P. Silva and A. Alonso, “Integration of Safety Analysis in Model-driven Software Development,” Software IET, 2008, pp. 260 – 280..

[7] W.H. Wu and T. Kelly, “Safety Tactics for Software Architecture Design,” Proceedings of the 28th Annual International Conference of Computer Software and Applications, 2004, Page(s):368-375.

[8] X.F. Cui, Y. C. Sun, X. Sai and M. Hong, “Architecture Design for the Large-Scale Software-Intensive Systems: A Decision-Oriented Approach and the Experience,” Proceedings of 14th IEEE International Conference on Engineering of Complex Computer Systems, 2009, pp. 30-39.

[9] R. Kazman, M. Klein and M. Barbacci, “The architecture tradeoff analysis method[C],” 4th International Conference on Engineering of Complex Computer Systems (ICECCS’98), 1998, pp. 35-45.

[10] C. Schifers and G. Hans, “IEEE standard for communications-based train control(CBTC) performance and functional requirements,” F,2000[C].

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.