# Safety Perspective for Supporting Architectural Design of Safety-Critical Systems

Havva Gülay Gürbüz, Bedir Tekinerdogan, and Nagehan Pala Er

Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey
havva.gurbuz@bilkent.edu.tr,
{bedir,nagehan}@cs.bilkent.edu.tr

**Abstract.** Various software architecture viewpoint approaches have been introduced to model the architecture views for stakeholder concerns. To address quality concerns in software architecture views, an important approach is to define *architectural perspectives* that include a collection of activities, tactics and guidelines that require consideration across a number of the architectural views. Several architectural perspectives have been defined for selected quality concerns. In this paper we propose the *Safety Perspective* that is dedicated to ensure that the safety concern is properly addressed in the architecture views. The proposed safety perspective can assist the system and software architects in designing, analyzing and communicating the decisions regarding safety concerns. We illustrate the safety perspective for a real industrial case study and discuss the lessons learned.

**Keywords:** Software architecture design, software architecture modeling, software architecture analysis, safety-critical systems.

## 1 Introduction

To address quality concerns in software architecture views, an important approach is to define *architectural perspectives* that include a collection of activities, tactics and guidelines that require consideration across a number of the architectural views [6]. In this context, Rozanski and Wood define several architectural perspectives for selected quality concerns such as security, performance, scalability, availability and evolution. In order to capture the system-wide quality concerns, each relevant perspective is applied to some or all views. In this way, the architectural views provide the description of the architecture, while the architectural perspectives can help to analyze and modify the architecture to ensure that system exhibits the desired quality properties.

An important concern for designing safety-critical systems is safety since a failure or malfunction may result in death or serious injury to people, or loss or severe damage to equipment or environmental harm. It is generally agreed that quality concerns need to be evaluated early on in the life cycle before the implementation to mitigate risks. For safety-critical systems this seems to be an even more serious requirement due to the dramatic consequences of potential failures. For coping with safety several standard and implementation approaches have been defined but this has not been directly considered at the architecture modeling level. Hence, we propose the *Safety*

*Perspective* that is dedicated to ensure that the safety concern is properly addressed in the architecture views. The proposed safety perspective is defined according to the guidelines as described by Rozanski and Woods [6]. The safety perspective can assist the system and software architects in designing, analyzing and communicating the design decisions regarding safety concerns. We illustrate the safety perspective for a real industrial case study and discuss the lessons learned.

The remainder of the paper is organized as follows. Section 2 presents the proposed safety perspective. Section 3 illustrates the safety perspective for an industrial case study. Finally, section 4 presents the conclusion.

## 2 Safety Perspective

Rozanski&Woods provide the following guidelines [6] to define a new perspective:

- The perspective description in brief in *desired quality*
- The perspective's *applicability to views*
- The *concerns* which are addressed by the perspective
- An explanation of *activities for applying the perspective* to the architectural design.
- The *architectural tactics* as possible solutions when the architecture doesn't exhibit the desired quality properties the perspective addresses
- Some *problems and pitfalls* to be aware of and risk-reduction techniques
- *Checklist* of things to consider when applying and reviewing the perspective to help make sure correctness, completeness, and accuracy

Table 1 shows the proposed safety perspective description including the above points. In the following we shortly discuss the each point.

**Table 1.** Brief Description of Safety Perspective

| | |
|---|---|
| **Desired Quality** | The ability of the system to provide an information about safety-related decisions and ability to control and monitor the hazardous operations in the system |
| **Applicability** | Any systems which include hazardous or safety-critical operations |
| **Concerns** | Failures, Hazard, Risks, Fault Tolerance, Availability, Reliability, Accuracy, Performance |
| **Activities** | Identify hazards, Define risks, Identify safety requirements, Design safety model, Assess against safety requirements |
| **Architectural Tactics** | Avoid from failures and hazards, Define failure detection mechanisms, Mitigate the failure consequences |
| **Problems and Pitfalls** | Describing the fault tolerance, No clear requirements or safety model, Underestimated safety problems |

Table 2 shows how the safety perspective affects each of the architectural views as defined by Rozanski and Woods [6]. For all the seven views the safety perspective seems to be useful and can reshape the corresponding view. The activity diagram in Fig. 1 shows the activities for applying the safety perspective. The first step includes the identification of the hazards followed by the definition of risks. This is followed by identifying and detailing the safety requirements. After the safety requirements safety models are designed and the safety requirements are assessed. In the following section we explain each activity using an industrial case study.

**Table 2.** Applicability of Safety Perspective to Architectural Views

| View | Applicability |
|---|---|
| Functional View | The functional view allows determining which of the system's functional elements considered as safety critical. |
| Information View | The information view helps to see the safety-critical data in the system |
| Concurrency View | While designing the safety-critical systems, some elements need to be isolated or integrated in runtime. Therefore this will affect the system's concurrency structure. |
| Development View | Applying this view can help to provide a guideline or constraints to developers in order to raise awareness for the system's safety critical elements. |
| Deployment View | Applying this view can help to determine the required hardware, third-party software requirements and some constraints for safety. |
| Operational View | Safety implementation includes critical and complex operations. Therefore, operational view needs to consider safety critical elements to describe system's operation properly. |
| Context View | Applying this view can help to understand which types of users will use the system and which external systems are necessary to make sure the system operates correctly. |

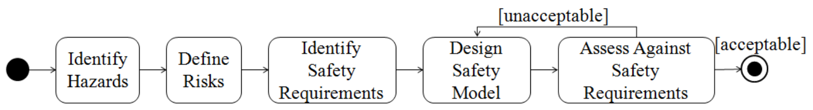

**Fig. 1.** Applying the Safety Perspective

## 3 Case Study

In this section we show the application of proposed safety perspective approach by using an avionics control system project of a company. To illustrate the application of the proposed safety perspective we have selected *"displaying aircraft altitude data"* as an example requirement for our case study. Altitude is defined as the height of the aircraft above sea level. Pilots depend on the displayed altitude information especially when landing.

### 3.1 Activities for Safety Perspective

This section explains how the activities given in Fig. 1 are applied to our case.

**Identify Hazards**
In order to identify and classify hazards, preliminary hazard analysis can be conducted which should include the list of all hazards, their probable causes and consequences, and the severity. Hazard severity levels are defined as *catastrophic, critical, marginal* or *negligible* in [2]. Hazard identification activity is performed with domain experts (avionics engineers and pilots), system engineers and safety engineers.

We have selected *"displaying wrong altitude data"* hazard related to selected requirement as an example hazard to illustrate the remaining activities. The possible causes of this hazard are loss of/error in altimeter device, loss of/error in communication with altimeter device and error in display device. Aircraft crash is identified as the possible consequence of this hazard. Severity of this hazard is identified as catastrophic since possible consequence of the hazard is aircraft crash.

**Define Risks**

To define risks, estimation of probability of hazard occurrence for each hazard should be carried out. In [2], occurrence definitions are defined as *frequent, probable, occasional, remote* or *improbable*. Based on the hazard severity and hazard occurrence class identification, risks should be assessed and categorized as *high, serious, medium* or *low* [2]. After the risk definition, risk assessment should be conducted by methods such as fault tree analysis, event tree analysis, simulation etc. For our case study, our design criterion is to design the system such that the probability of occurrence of all catastrophic failures should be improbable. Since the selected hazard is catastrophic hazard, the probability of occurrence is improbable. According to severity category and probability of occurrence, the risk category of the selected hazard is medium.

**Identify Safety Requirements**

After the hazard identification and risk assessment, software safety requirements should be determined to construct a safety model. Safety requirements can be identified by using different methods such as *preliminary hazard analysis* [7], top-down analysis of system requirements and specifications [7] and *fault tree analysis* [5]. Additionally, there are some other methods which combine the several existing techniques to derive safety requirements. To illustrate this step, we produce *"Probability of displaying wrong altitude should be improbable"* as a high-level safety requirement related to selected hazard. Many low-level safety requirements can be generated from this high-level safety requirement. Examples of the generated low-level safety requirements are (1)*"Altimeter data should be received at least two independent altimeter devices."*, (2) *"If the difference between two altimeter values received from two altimeter devices is more than a given threshold, the altimeter data should not be displayed and a warning should be generated."*, (3)*"Altimeter data should be shown on at least two independent display devices ".*

**Design Safety Model**

To present the safety-critical elements or components in the system a safety model is needed that can be derived from safety requirements. One way to create a safety model of the system is defining an extension mechanism to UML models [3]. UML extension can be achieved by adding stereotype to UML diagrams. Another approach to design a safety model is defining a domain-specific language [12]. Another way to express safety model is using automata [14].

This activity is an iterative process. The models are created first and then they are checked against the safety requirements. The models can be changed according to these checks. We prefer to show two versions of the architecture for our case study.

The first version is designed without considering the safety requirements. It is modified after safety requirements are identified, that is, after safety perspective is applied, which results in the second version. The reasons of the modifications will be explained in the next section (assessment section). The left part of the Fig. 2 shows the deployment diagram of the first version, which includes one avionics control computer (AvionicsComputer), one altimeter device (Altimeter), and one display device (GR_Display). The deployment diagram of the second version, after applying the safety perspective, is shown in the right part of the Fig. 2. The second version includes two avionics control computers (AvionicsComputer1 and AvionicsComputer2), two altimeter devices (Altimeter_1 and Altimeter_2), and two display devices (GR_1_Display and GR_2_Display). Avionics control computer contains following modules: M1153 Manager (M1553), A429 Manager (A429), Navigation Manager (NAV), Graphics 1 Manager (GR_1), Graphics 2 Manager (GR_2), Health Monitor (Health_Monitor).

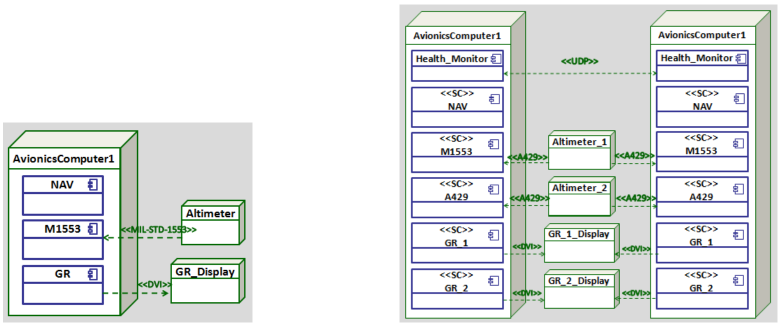

**Fig. 2.** Deployment View for the First Version (left) and for the Second Version(right)

M1553 Manager receives data from the devices connected to MIL-STD-1553 communication channels. Similarly, A429 Manager receives data from the devices on the ARINC-429 communication channels. MIL-STD-1553 and ARINC-429 are two widely known communication standards used in avionics systems. These two managers just receive the data and send it to the required modules. They do not make any calculations on the data. Navigation Manager receives the altimeter data from M1553 Manager and A429 Manager and makes the range check and difference check calculations on the altimeter data. If the difference between two altimeter values received from two altimeter devices is more than a given threshold, a warning data is produced. The altimeter data and warning data are sent to Graphics Managers. Graphics Managers drive two graphical displays according to the received data. A well-known standard called DVI is used to drive graphical displays. *SC* (Safety Critical) stereotype is defined to tag the safety-critical modules in the second version of the deployment diagram. *SC* stereotype differentiates the safety-critical modules from the rest of the modules.

**Assess Against Safety Requirements**

After designing the system's safety model, it should be assessed to check whether it is consistent with identified safety requirements. There is only one altimeter device and one display device in the first version of the architecture so low-level safety requirements 1 and 3 are not satisfied. We adapted the first version and included one additional altimeter device and one additional display device in the second version of the architecture. There are two different altimeter devices and two different display devices in the second version so low-level safety requirements 1 and 3 are satisfied.

Redundancy is also accomplished for the avionics control computer in the second version of the architecture. There are two avionics computers which can communicate to each other for heartbeat messages (through UDP protocol). They run according to master/slave paradigm. Only one of the avionics computers can be master at a given time. If slave avionics computer cannot receive heartbeat messages, it can become master. Both of them can receive altimeter data and can display it on graphical display devices but only the master computer does it.

Safety requirement 2 is also satisfied in the second version of the architecture. Navigation Manager checks the altitude data and produces either the altitude data or a warning for altitude. If altitude data is produced, it is displayed on both graphical devices by Graphics Managers. If a warning is generated, a warning symbol is displayed on the graphical devices instead of altitude. Health monitoring is another tactic which is applied in order to increase the safety of the system. Health monitor checks the status of the modules. If there is a problem related with a module, it can restart the module. Health monitors are also used to determine master/slave condition. Heartbeat messages are sent and received by health monitors.

## 3.2 Architectural Tactics

Architectural tactics can be considered as possible solutions when the architecture does not exhibit the required quality properties addressed by the perspective. In order to avoid from failures and hazards, one way is making the system as simple as possible. Another way is applying *redundancy* [13] by replicating the components in the system. The other way is N-version programming proposed by Chen and Avizienis [1]. By using N-version programming technique, different designs can be created for each version of the system in order to determine design faults from safety perspective. If hazards and failures occur, system should be able to detect them. In order to detect the failures, failure detection mechanisms can be derived from safety requirements [8]. Another tactic for failure detection is *heartbeat* [ref] which offers a mechanism for periodically monitoring the aliveness and arrival rate of independent runnables. At the architecture design level, based on the hazard identification and risk definition, consequences of failures can be predicted and reduced/prevented. *Redundancy* and *replication* also can be used in order to mitigate from the failure consequences.

Several architectural tactics are utilized for our case study. The first architectural technique is redundancy. Several parts of the system are designed as redundant in order to satisfy both safety requirements and high availability needs. This technique is applied to avoid from failures and mitigate the failure consequences. Health monitoring technique is applied for failure detection of the safety-critical modules. Table 3 summarizes the applied tactics. Similar tactics can be applied for other identified catastrophic hazards.

## 3.3 Checklist

In this section, we provide checklists in Table 4 for requirements capture and architecture definition to consider when applying and reviewing the perspective to help make sure correctness, completeness, and accuracy. We have applied the checklist to our case study. Results are presented in third column of Table 4. All items in the checklist are answered as yes except for the item 9. Since our case study doesn't include any safe state, this question is answered as not applicable.

**Table 3.** Architectural Tactics for the Case Study

| **Tactic** | Avoid. | Detect. | Mitigate |
|---|---|---|---|
| If one of the altimeter devices produces wrong altimeter output, this fault is detected by Navigation Manager and a warning is generated | ✓ | ✓ | ✓ |
| If one of the display devices crashes and cannot display altitude data, the other one continue to display it. | ✓ |  | ✓ |
| If master avionics computer is not available, the slave avionics computer becomes master and starts to operate. | ✓ |  | ✓ |
| If a safety-critical module fails, this failure is detected by health monitor. The module is re-started. |  | ✓ | ✓ |

**Table 4.** Checklist Table

| **No** | **Explanation** | **Y/N/NA** |
|---|---|---|
| 1 | Have you identified safety-critical operations in the system? | Yes |
| 2 | Have you identified possible failures and hazards including causes and consequences of them? | Yes |
| 3 | Have you worked through the hazard severity and occurrence information to define the risks? | Yes |
| 4 | Have you identified availability needs for safety of the system? | Yes |
| 5 | Have you worked through example scenarios with your stakeholders so that they understand the planned safety risks the system runs? | Yes |
| 6 | Have you reviewed your safety requirements with external domain experts? | Yes |
| 7 | Have you addressed each hazard and risk in the designed safety model? | Yes |
| 8 | Is the design of safety model as simple as possible and highly modular? | Yes |
| 9 | Have you identified safe states and fully checked and verified them for completeness and correctness? | NA |
| 10 | Have you produced an integrated overall safety design of the system? | Yes |
| 11 | Have you defined the fault tolerance of the system? | Yes |
| 12 | Have you applied the results of the safety perspective to all effected views? | Yes |
| 13 | Have domain experts reviewed the safety design? | Yes |

### 3.4 Applicability to Views

Table 5 lists the application of safety perspective to the views for our case study.

**Table 5.** Safety Perspective Application for the Case Study

| View | Applicability to the case study |
|---|---|
| Functional | Safety-critical modules are determined (see right part of the Fig. 2) |
| Information | Safety-critical data is determined (altitude data) |
| Concurrency | Not applicable |
| Development | Requirement Standard, Coding Standard, Design Decisions, Reviews / Checklists and common processing required are defined. |
| Deployment | There are two avionics control computers, two altimeter devices and two display devic-es. (see right part of the Fig. 2) |
| Operational | Check the correctness of the loaded binaries, Software Configuration Management and Software Problem Reporting for safety-critical defects are defined, maintenance and user training are provided. |
| Context | External devices related with safety-critical features are determined. |

## 4 Conclusion

Safety-critical systems need to be carefully designed and analyzed because a failure may result in death or serious injury to people, or severe damage to equipment. Hereby, the architecture design plays a crucial role to support the overall design and realization of the system and ensure the required level of safety. Addressing quality concerns at the architecture view level has been actually based on either defining a new viewpoint [2] or using architecture perspectives [7], each with their own merits. In our earlier work we have considered the explicit modeling of viewpoints for quality concerns [9][10][11]. Unfortunately, so far no architectural perspective has been defined for the safety concern. Based on the guidelines by Rozanski and Woods [7] we have proposed a safety perspective that can be used in the design of safety-critical systems. We have applied the safety perspective in a real industrial context. The safety perspective helps the designers to explicitly reason about and document the design decisions regarding the safety concern. In this respect, the safety perspective appeared not only to be useful as a guidance tool for assisting the safety engineer and the architect, but it also helped in the early analysis of the architecture. In our future work we aim to apply the safety perspective for several other domains and consider the trade-off analysis with the perspectives for other quality concerns. Further we also aim to define a viewpoint for safety.

## References

[1] Chen, L., Avizienis, A.: N-Version Programming:A Fault-Tolerance Approach to Reliability of Software Operation. In: Fault Tolerant Computing, FTCS-8, pp. 3–9 (1978)

[2] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J.: Documenting Software Architectures: Views and Beyond, 1st edn. Addison-Wesley (October 2002)

[3] MIL-STD-882D, Standard Practice for System Safety, Department of Defense (2000) (retrieved January 22, 2014)
[4] Pataricza, A., Majzik, I., Huszerl, G., Várnai, G.: UML-based design and formal analysis of a safety-critical railway control software module. In: Proc. of Symposium Formal Methods for Railway Operation and ControlSystems (FORMS 2003), Budapest, pp. 125–132 (2003)
[5] Ramezani, R., Sedaghat, Y.: An Overview of Fault Tolerance Techniques for Real-Time Operating Systems. In: 3th International Conference on Computer and Knowledge Engineering, pp. 1–6 (2013)
[6] Rausand, M., Hoylan, A.: System Reliability Theory, Models, Statistical Methods, and Applications. Wiley, USA (2004)
[7] Rozanski, N., Woods, E.: Software Architecture Systems Working with Stakeholders Using Viewpoints and Perspectives, 1st edn. Addison-Wesley (2005)
[8] Software Safety Guide Book, NASA Technical Standard (2004)
[9] Sojer, D., Christian, B., Knoll, A.: Deriving Fault-Detection Mechanisms from Safety Requirements. In: Computer Science- Research and Development, pp. 1–14. Springer (2011)
[10] Sözer, H., Tekinerdogan, B.: Introducing Recovery Style for Modeling and Analyzing System Recovery. In: 7th IEEE/IFIP Working Conference on Software Architecture, Vancouver, Canada, February 18-22, pp. 167–176 (2008)
[11] Sözer, H., Tekinerdogan, B., Aksit, M.: Optimizing Decomposition of Software Architecture for Local Recovery. Software Quality Journal 21(2), 203–240 (2013)
[12] Tekinerdogan, B., Sözer, H.: Defining Architectural Viewpoints for Quality Concerns. In: Crnkovic, I., Gruhn, V., Book, M. (eds.) ECSA 2011. LNCS, vol. 6903, pp. 26–34. Springer, Heidelberg (2011)
[13] Wasilewski, M., Hasselbring, W., Nowotka, D.: Defining requirements on domain-specific languages in model-driven software engineering of safety-critical systems. In: Lecture Notes in Informatics Software Engineering Workshopband, pp. 467–482 (2013)
[14] Wu, W., Kelly, T.: Safety Tactics for Software Architecture Design. In: 28th Annual International Computer Software and Applications Conference, Hong Kong, pp. 368–375 (2004)
[15] Yu, G., Wei Xu, Z.: Model-Based Safety Test Automation of Safety-Critical Software. In: International Conference on Computational Intelligence and Software Engineering, pp. 1–3 (2010)