# Two safety patterns: Safety Assertion and Safety Assertion Enforcer.

Eduardo B. Fernandez, Brahim Hamid

# Two safety patterns: Safety Assertion and Safety Assertion Enforcer

EDUARDO B. FERNANDEZ, Florida Atlantic University
BRAHIM HAMID, CNRS/IRIT Laboratory

Safety is the avoidance of unacceptable hazards, including threats to human lives, the environment, or to costly facilities. Safety constraints are expressed using assertions that define system states that should not occur because they may lead to mishaps. We present here two safety patterns. The Safety Assertion pattern describes the contents of an assertion that indicates a state of the system that must not happen. The Safety Assertion Enforcer pattern evaluates safety assertions when there is an incoming event that can change the state of the system and prevents the change if it violates an assertion. These patterns can be useful for designing or certifying safe systems.

Categories and Subject Descriptors: **D.2.11 [Software Engineering]**: Software Architectures - Patterns

General Terms: Design

Additional Key Words and Phrases: safety, architecture patterns, safety patterns, safety assertions.

## 1. INTRODUCTION

Safety is the avoidance of unacceptable hazards, including threats to human lives, the environment, or to costly facilities. Hazards are states or conditions of the system, that when combined with some state of the environment, lead to mishaps. A mishap is the occurrence of a situation that leads to human or environmental harm; that is, a mishap is a violation of safety. Safety is often expressed with assertions that describe situations (states) or actions that must be avoided, e.g. an elevator must not open its doors when moving. These assertions are usually related to specific states of control systems. Typically, security constraints are needed as well. Control systems manage, command, monitor, and regulate the behavior of other devices or systems. For realizing their control functions these systems have sensors, actuators, and a decision logic. Based on observed events and their decision logic they affect their environment and their controlled physical entities via actuators. In their present form, they have become industrial control systems (ICSs), which manage a variety of subsystems, and where safety constraints are even more necessary (Krotofil and Gollmann 2013). ICSs are usually cyber-physical systems (CPSs), which integrate computing and communication capabilities with the monitoring and control of entities in the physical world (Lee 2008). Because of their larger scope and heterogeneity, CPSs have an increased need for safety restrictions.

Author's address: Eduardo B. Fernandez, email: fernande@fau.edu; Dept. of Computer and Elect. Eng. and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA. Brahim Hamid, CNRS/IRIT Laboratory, Institut de Recherche en Informatique (Computer Science Research Institute) of Toulouse, 31062 Toulouse Cedex 9, France

The static and dynamic conceptual models of control systems, including safety constraints, should be defined at the application level where the semantics of the application are well understood (Fernandez2013, J. Nicolas et al., 2006). Because of the explicit semantics in this level we can also apply here institutional policies as well as regulations, which normally establish semantic constraints. Other non-functional aspects can also be specified at this level, e.g., the required degree of security and reliability for each use case of the system (Choi 2007, Fernandez2013). The lower levels enforce the restrictions defined at the application level. Each level has its own safety or security mechanism and should participate in enforcing the constraints.

We present here two patterns that describe safety assertions and their enforcement:

- Safety Assertion. Describe the contents of an assertion that indicates a state of the system that must not happen because it may lead to a mishap.
- Safety Assertion Enforcer. Evaluate safety assertions when there is an incoming event that can change the state of the system and prevent the change if it violates an assertion.

Our intended audience includes designers and system administrators of control systems which must have safety constraints, including most cyber-physical systems. We assume our readers to be familiar with UML.

Section 2 describes the Safety Assertion pattern, while Section 3 presents the Safety Enforcement pattern. We finish with some conclusions and ideas for future work in Section 4.

## 2. SAFETY ASSERTION PATTERN

### 2.1 Intent

Describe the contents of an assertion that indicates a state of the system that must not happen because it may lead to a mishap.

### 2.2 Context

.Any control system that controls operations that may be harmful to humans or the environment.

### 2.3 Problem

Mishaps happen because a dangerous situation was not considered in the requirements, or because there was a failure in some unit of the system, or because there was an intentional action taken against vulnerable elements of the system. Complex systems such as ICSs and CPSs can have a variety of potential mishaps. Currently, different type of professionals perform these designs and their terminology may not be precise which may result in hazards being overlooked.

Leveson categorizes hazards into three groups [Lev95]: The system is not available. The system generates an incorrect output. The system misses a hard deadline. Any of these types of hazards can produce mishaps, which should be avoided regardless of their cause. The solution to this problem is affected by the following forces:

Precision—Assertions should define precisely the concepts used in safety designs. Imprecise assertions may be misleading.

Contents—All the elements involved in a safety assertion must be explicitly included; otherwise, the assertion could be ambiguous or incomplete and lead to mishaps.

Boolean decisions—Assertions must be evaluated to true or false only; otherwise, it would not be clear what the system should do.

Regulations—The assertion must indicate relevant regulations about type of safety required; e.g. avoidance of harm to people takes priority over harm to infrastructure, maintenance must be performed within specific time periods, or similar.

Risk—There should be a way to specify the severity level and the implied risk in case the assertion is not followed.

### 2.4 Solution

Hazards must be prevented in the design of the software for the system. Safety is dependent on the correctness, reliability, and security of the system.

Describe a Boolean assertion that if true forbids a state considered unsafe. The assertion should indicate a condition and the effect of an operation or event that may change the current state. It should also indicate the risk for not following it. For example, an elevator is moving (condition) with closed doors (current state) when an event occurs that attempts to open the doors (new state). The event should have no effect if the Assertion indicates that this new state is not acceptable. If the assertion is not followed there should be a measure of the resulting risk. A regulation may indicate that the elevator must be tested at specific intervals (which will reduce the probability of an error event happening), or similar constraints.

#### Structure

Figure 1 shows the class diagram for this pattern. **Assertions** apply to the whole **System** or to some of its **Components**. Assertions refer to the changes from an old to a new **State**. An assertion includes a Boolean **Condition** (time, date, operational context) that must be true for the assertion to apply[1]. The **Event** would produce a change of state. **Risk** is some measure of the impact of not avoiding the hazard. Assertions must comply with existing **Regulations**. The **Assertion Catalog** collects all the assertions that apply to the system or its components.

#### Dynamics

Possible use cases include Add/Delete an assertion and Modify an existing assertion. These are very simple, they only require modifying the Assertion Catalog, and we do not illustrate them with sequence diagrams.

### 2.5 Implementation

Assertions can be expressed in any suitable language, including formal languages such as OCL (Warmer and Kleppe 2003). Assertions should be checked for possible inconsistencies. The system administrator would manage them using some user interface connected to a repository, similar to a Policy Definition Point (see the Assertion Catalog of Figure 1).

This pattern could be used in a methodology such as the one described in (Hauge et al. 2011, 2012, 2013, Huang 2013, Leveson 1994, and Wu/Kelly 2004) for safe systems, and in new methodologies to build CPSs. It could also be useful to add to the MARTE dependability profile defined in [Bernardi et al. 2011]. Finally, it can fit well a model-driven certification method (R. Panesar-Walawege et al. 2011)

[1] An expression (predicate) involving time, place, or anything, can be true if it matches the current situation.

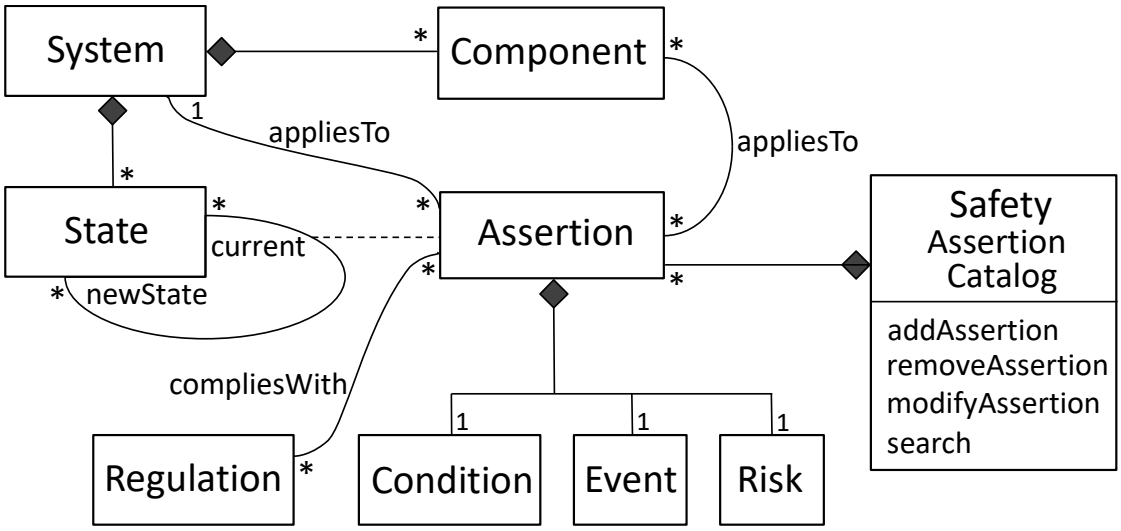


Figure 1. UML class diagram of the Safety Assertion pattern

2.6 Known Uses

- (J. Nicolas et al. 2006) describe a quality attribute template for the management of commands in tele-operated systems where they describe the conditions to prevent the execution of unsafe commands.

- (Thramboulidis and Scholz 2010) present a metamodel for safety concepts which can describe the effects of not enforcing the safety assertions, that is, resulting hazards.

- Industrial systems, such as elevators, trains, and robot systems use similar concepts to avoid mishaps.

2.7 Consequences

This pattern provides the following advantages:

- Precision—Assertions can define precisely the terms used in safety designs, which can avoid misunderstandings in implementing industrial systems.

- Contents—All the elements that are considered necessary according to the forces are now included in the assertion.

- Boolean decisions—Assertions are logical statements that can be evaluated to true or false.

- Regulations—The assertion can be related to relevant regulations that may specify the type of safety control required, maintenance periods, or required certification.

- Risk—There is now a way to specify the severity level and the implied risk in case the assertion is not followed

Possible liabilities include overhead to evaluate each assertion and unnecessary complexity for simple applications. There may be a need to override assertions in cases of emergency, e.g. opening the elevator doors in between floors because movement is blocked; this case is not considered in the pattern.

2.8 Related patterns

- Authorization (Fernandez 2013). Describe who is authorized to access specific resources in a system, in an environment in which we have resources whose access needs to be controlled. Rules (policies) indicate for each active entity, which resources it can access, and what it can do with them. This pattern can be used to determine if a command to the system (event) is authorized or not.
- Regulation patterns, e.g. Health Insurance Portability and Accountability Act, HIPAA (Fernandez and Mujica 2014). This paper describes patterns for the Privacy and Security Rules of HIPAA.
- Safety Assertion Enforcer. Evaluate safety assertions when there is an incoming event that can change the state of the system and prevent the change if it violates an assertion.

## 2. SAFETY ASSERTION ENFORCER PATTERN

3.1 Intent

Evaluate safety assertions when there is a new event that may change the state of the system and prevent the change if it violates an assertion.

3.2 Context

.Any control system that controls operations that may be harmful to humans or the environment. This pattern assumes that we have a catalog of safety assertions that describe all the situations that may produce mishaps.

3.3 Problem

Mishaps happen because a dangerous situation was not considered in the requirements, or because there was a failure in some unit of the system, or because there was an intentional action taken against vulnerable elements of the system. Complex systems such as ICSs and CPSs can have a variety of potential mishaps. Currently, different type of professionals perform these designs and their terminology may not be precise which may result in hazards being overlooked.

Leveson categorizes hazards into three groups [Lev95]: The system is not available. The system generates an incorrect output. The system misses a hard deadline. Any of these types of hazards can produce mishaps, which should be avoided regardless of their cause. If we are in a safe state and there is a new event that may change this state, before we perform the state transfer we must make sure that the transition will not take the system to an state which may produce a mishap. The solution to this problem is affected by the following forces:

Full mediation—Every attempt to change state must be mediated by a process that decides if the change is safe.

Reified decisions--Decisions can be sometimes more complex than a Boolean response, e.g. some action may be partially executed. Also, many times there is a need to store decisions for possible reuse and to manipulate decisions as objects (for undo or redo purposes).

3.4 Solution

Use a concept analogous to the Security Reference Monitor (Fernandez 2013) to evaluate the effect of commands on assertions and decide if they will produce a safe change of state. State changes must not happen if they lead to mishaps.

*Structure*

Figure 2 shows the UML class diagram for this pattern. **Events** may come from **Users** or **Systems** (**Subjects**). An event that may change the state of the system is intercepted by the **Assertion Enforcer** which decides if this change would result in an unsafe state. To make this decision the Assertion Enforcer uses the **Safety Assertion Catalog** that stores the assertions that apply to the **System** or its **Hardware** or **Software** Components; if a specific **Safety Assertion** is true the **Event** will not result in any change. The **Decision** indicates the result of evaluating the effect of the event. There must be a decision for each event. Note that the class Safety Assertion Catalog together with pattern Safety Assertion form a Policy Definition Point pattern (see Related Patterns). Events and Decisions are reified so they can be manipulated as classes, which is useful for a better analysis or to undo or redo them.

*Dynamics*

Figure 3 shows the sequence diagram for the use case "Receive Event". An event coming from a subject is evaluated by the Safety Enforcer by searching in the Safety Assertion Catalog for assertions affected by this event. If any of those assertions evaluates to True the Decision is "disallow (event)" and the state change does not occur.

3.5 Implementation

Decisions and assertions can be expressed in any suitable language, including using formal languages such as OCL (Warmer and Kleppe 2003). The enforcer can be a demon or agent that intercepts attempts to change state. Instead of events, we may receive operations sent by users or other systems, those should be handled similarly.

This pattern could be used in a methodology such as the one described in (Hauge et al. 2011, 2012, 2013, Huang 2013, Leveson 1994, and Wu/Kelly 2004) for safe systems, and in new methodologies to build CPSs. It could also be useful to add to the MARTE dependability profile defined in [Bernardi et al. 2011]. Finally, it can fit well a model-driven certification method (R. Panesar-Walawege et al. 2011)

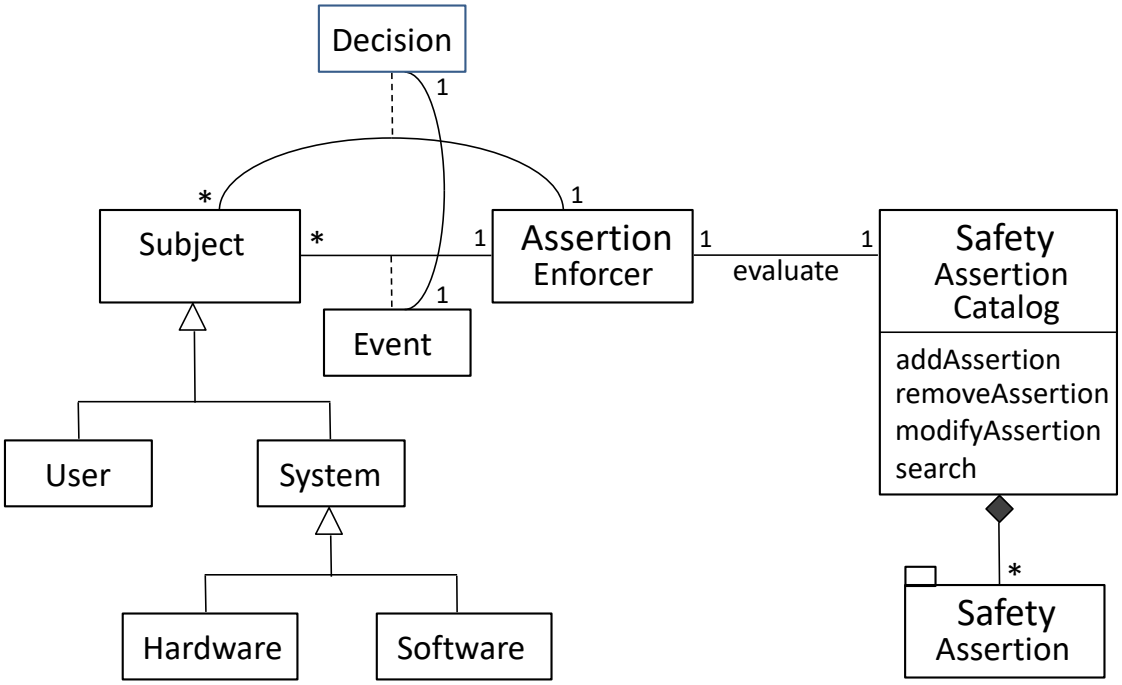


Figure 2. UML class diagram of the Safety Assertion Enforcer pattern

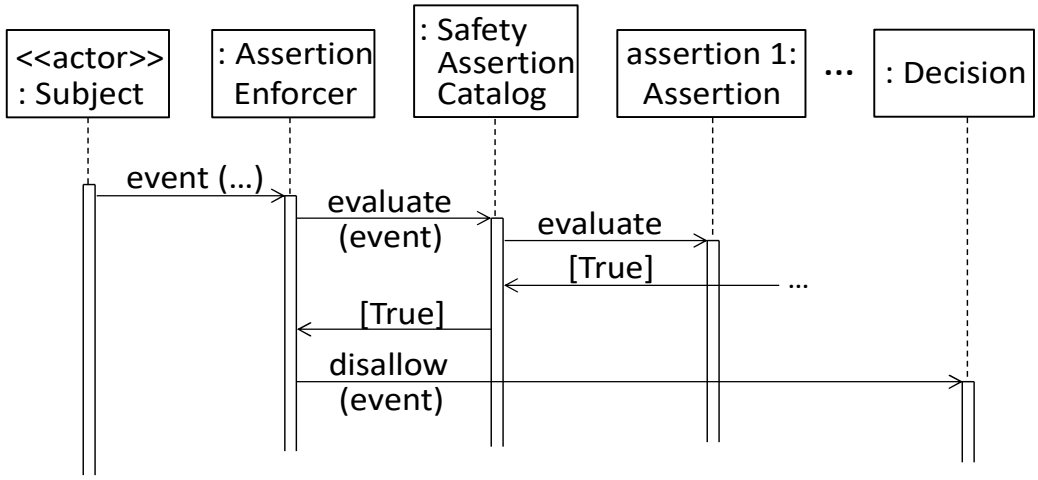


Figure 3. Sequence diagram for use case "Receive event"

3.6 Known Uses

- (J. Nicolas et al. 2006) describe a quality attribute template for the management of commands in tele-operated systems where they describe the conditions to prevent the execution of unsafe commands.

- (Thramboulidis and Scholz 2010) present a metamodel for safety concepts which can describe the effects of not enforcing the safety assertions, e.g. resulting hazards.

- Industrial systems, such as elevators, trains, and robot systems use similar concepts to avoid mishaps.

3.7 Consequences

- Full mediation—Every attempt to change state is mediated by a process (the Assertion Enforcer) that decides if the change is safe.

- Reified decisions--Decisions can be now more complex than a Boolean response, e.g., a change of state may be allowed if some condition is satisfied. . Also, many times there is a need to store decisions for possible reuse and to manipulate decisions as objects.

- Possible liabilities include evaluation overhead and unnecessary complexity for simple cases.

3.8 Related patterns

- Safety Assertion. Describe the contents of an assertion that indicates a state of the system that must not happen because it may lead to a mishap.

- Regulation patterns, e.g. Health Insurance Portability and Accountability Act, HIPAA (Fernandez and Mujica 2014). This paper describes patterns for the Privacy and Security Rules of HIPAA.

- Authorization (Fernandez 2013). Describe who is authorized to access specific resources in a system, in an environment in which we have resources whose access needs to be controlled. Rules (policies) indicate for each active entity, which resources it can access, and what it can do with them. This pattern can be used to determine if a command to the system (event) is authorized or not.

- Policy Definition Point (PDP). The pattern on Policy-Based Access Control uses a Policy Definition Point, which is a pattern in itself (Fernandez 2013).

- (Hauge and Stolen 2011) proposes a pattern based approach called Safe Control Systems (SaCS) that includes a pattern language and provides guidance on the development of design concepts for safety critical systems. The authors provide three types of basic patterns: Requirements, Design, and Safety, with an example of each one: State Space Subset, Trusted Backup, and Adapting within a Constrained State Space, respectively. The safety pattern is justified using arguments. These patterns are used in an example of a railway interlock. This work is extended in (Hauge and Stolen 2012, and Hauge and Stolen 2013).

- Reference Monitor (Fernandez 2013). Enforce authorization when a subject requests a protection object and provide the subject with a decision. In a computational environment in which users or processes make requests for data or resources, this pattern enforces declared access restrictions when an active entity requests resources. It describes how to define an abstract process that intercepts all requests for resources and checks them for compliance with authorizations

- A Failure pattern describes how a failure happens; we introduced them in (Buckley and Fernandez 2012).

## 4. CONCLUSIONS

Although a variety of patterns and methodologies for safety have been published (Choi 2007, Hauge and Stolen 2011, 2012, 2013, Wu and Kelly 2004), we could not find a pattern to describe safety assertions. We had written patterns for security rules and this work parallels that study. A safety assertion is similar to an authorization rule except that it is valid for any attempted change while authorization rules apply to actions by specific users or systems (subjects).

An important result of a systematic lifecycle for safety-oriented designs is a procedure for certification (Heimdahl 2007). We have explored how patterns can help for reliability certification (Buckley et al. 2011), but not for safety. Another aspect of future work is the integration with security and reliability patterns.

In analogy with Misuse patterns (Fernandez 2013) and Failure patterns we intend to write Mishap patterns that describe how hazards result in mishaps, showing how the units of the system participate and how the mishap could be avoided.

## ACKNOWLEDGEMENTS

We thank our shepherd Nazila Golmohammadi for her excellent suggestions that significantly improved this paper. The participants in the Writers' Workshop at the conference also provided very valuable ideas.

## REFERENCES

S. Bernardi, J. Merseguer, D.C.Petriu, "A dependability profile within MARTE", Soft. Syst. Model., 2011, 313-336. DOI 10.1007/s10270-009-0128-1

Ingrid Buckley, Eduardo Fernandez, Marco Anisetti, Claudio A. Ardagna, Masoud Sadjadi, and Ernesto Damiani, "Towards Pattern-based Reliability Certification of Services, Procs. 1st International Symposium on Secure Virtual Infrastructures (DOA-SVI'11), 17-19 Oct. 2011, Crete, Greece, Vol. 7045 Springer Lecture Notes in Computer Science.

Ingrid Buckley and E.B.Fernandez, "Failure patterns: A new way to analyze failures",

First International Symposium on Software Architecture and Patterns in conjunction with the 10th Latin American and Caribbean Conference for Engineering and Technology, July 23-27, 2012, Panama City, Panama.

Y. Choi: "Early Safety Analysis: from Use Cases to - Component-based Software Development", in Journal of Object Technology, vol. 6, no. 8, September - October 2007, 185-203 http://www.jot.fm/issues/issue_2007_09/article4.

E.B.Fernandez, "Security patterns in practice: Building secure architectures using software patterns", Wiley Series on Software Design Patterns, 2013.

E. B. Fernandez and S. Mujica, "Two patterns for HIPAA regulations", Procs. of AsianPLoP (Pattern Languages of Programs) 2014, Tokyo, Japan, March 2014.

E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Boston, Mass., 1994.

A. A.Hauge and K. Stølen,"SACS: a pattern language for safe adaptive control software", Procs. of the 18th Conference on Pattern Languages of Programs (PLoP '11).

A. A. Hauge and K. Stølen,"A pattern-based method for safe control systems exemplified within nuclear power production", SAFECOMP, LNCS 7612, Springer 2012, 13-24.

A. A. Hauge and K. Stølen, " Developing safe control systems using patterns for assurance", Third Int. Conf. on Performance, Safety and Robustness in Complex Systems and Applications (PESARO 2013).

M. Heimdahl, "Safety and software intensive systems: Challenges old and new", FoSE 2007.

Yuling Huang, "Safety-Oriented Software Architecture Design Approach", International Conference on Information Science and Computer Applications (ISCA 2013)

M. Krotofil, D. Gollmann, "Industrial control systems security: What is happening?" Procs. IEEE INDIN 2013, 664-669.

Edward A. Lee, "Cyber Physical Systems: Design Challenges", Procs. of ISORC, Orlando, FL May, 2008

N.G. Leveson, M.P.E.Heimdahl, H. Hildreth, and J.D.Reese, "Requirements specification for process-control systems", IEEE Trans. on Soft. Eng., vol 20, No 9, September 1994, 684-707.

N. G. Leveson, Safeware: System Safety and Computers, Addison-Wesley, 1995

J. Nicolas, J. Lasheras, A. Toval, F.J.Ortiz, B.Alvarez, "An integrated domain analysis approach for teleoperated systems", Requirements Eng., vol. 14, 2006, 27-46.

R.K.Panesar-Walawege, M. Sabetzadeh, L. Briand, "Using model-driven engineering for managing safety evidence: Challenges, vision, and experience", IEEE First Intl. Workshop on Software Certification, 2011, 7-12.

K Thramboulidis, S Scholz, "Integrating the 3+1 SysML view model with safety engineering", IEEE Conf. on Emerging Technologies and Factory Automation (ETFA), 2010, 1-8.

J. Warmer and A. Kleppe, The Object Constraint Language (2nd Ed.), Addison-Wesley, 2003.

W. Wu and T. Kelly, "Safety tactics for software architecture design", Proc. 28th Ann. Int. Conf. of Computer Software and Applications, 2004, 368-375.