Towards an Approach to Represent Safety Patterns

Pablo Oliveira Antonino, Thorsten Keuler
Embedded Systems Development Department
Fraunhofer IESE
Kaiserslautern, Germany
(Pablo.Antonino, Thorsten.Keuler)@iese.fraunhofer.de

Elisa Yumi Nakagawa
Department of Computer Systems
USP - University of São Paulo
São Carlos, Brazil
elisa@icmc.usp.br

***Abstract*—Safety-critical systems are complex entities, which, due to severe regulations, demand continuous development of approaches for supporting their construction. To keep safety-critical systems free of failures, it is fundamental to identify potential failure modes and their causes, and to eliminate them. One major approach to solving failure modes is the application of safety patterns at the architectural level of such systems. However, this is not trivial, since safety patterns have not been represented in a widely accepted way that would facilitate their understanding and use. In order to contribute to filling this gap, we present in this paper an approach for representing safety patterns in a way that allows them to be properly modeled and also offers means to support their application in architectural models. To this end, we propose the joint use of a UML profile and rules that are descriptive structures stating safety patterns application constraints. We have observed that our approach makes the safety patterns easy to represent and apply, thus contributing to the development of safety-critical systems.**

Keywords-Safety Pattern; UML Profile; Pattern Descriptive Rule; Architectural Model.

I. INTRODUCTION

Domains such as automotive and avionics demand high integrity levels between hardware and software to ensure proper execution of their systems, which, in turn, are constantly becoming larger and more complex [1]. A commercial airplane, for instance, contains systems that control ground proximity, navigation, and engine commands, amongst others. Almost all of these systems are safety-critical; i.e., a failure would lead to a catastrophic situation, endangering human lives and/or the environment. To minimize the probability of failure in safety-critical systems, it is necessary to integrate tactics based on well-known fault-tolerant methods to deal with failure avoidance, detection, and containment, for example, monitoring and redundancy [2]. Most of these tactics are concretized by means of specific design patterns, best known as Safety Patterns. Examples of safety patterns are Watchdog, Homogeneous Redundancy, and Sanity Check [3].

The growing popularity of model-driven approaches, such as Model Driven Architecture (MDA) and Model Transformation, has triggered the use of such tactics in the construction of safety-critical systems [1]. In this context, UML (Unified Modeling Language) [4] has been used by several MDA-based approaches for representing the required models. The reason is the diversity of elements and diagrams offered by UML, which provides means for expressing systems from diverse perspectives [5], and the existence of extension mechanisms like UML Profiles, which allows modeling particularities of domains by means of customizations of UML's syntax and semantics [6].

The UML's official specification for representing design patterns consists of using Parameterized Collaborations [7]. However, due to singularities that are inherent to safety-critical systems and safety patterns, UML and the others initiatives mentioned in the literature [8][7][9][10][11] are not appropriate for representing safety patterns in a standardized way that jointly facilitates their understanding and supports the automatized application of patterns in architectural models. Actually, the existent approaches belong to one of two extremes: (1) too complex and far from intuitive, requiring deep knowledge about formal specification techniques for representing patterns, or (2) providing only subjective information about the pattern that is useful only for reasoning on high-level concerns, which is important, but not enough to support the application of safety patterns in architectural models.

To fill this gap, we propose an approach for representing safety patterns that offers means for graphically expressing the structure and purpose of a safety pattern, and also provides information to facilitate its automatized application in architectural models by means of Model Transformations. In a nutshell, this corresponds to the joint use of UML Profile and descriptive rules stating safety pattern constraints that are worth being considered for their application in architectural models. After representing a set of safety patterns relevant in the domain of safety-critical systems described in [3] with our approach, we observed that the safety patterns became easier to represent and reuse, thus indicating our contribution to the construction of safety-critical systems.

The remainder of this paper is structured as follows: In Section II, the overall context is presented; in Section III, the related works are described; in Section IV, we present our approach; in Section V, we show the complete representation of a safety pattern using our approach; and in Section VI we conclude and present perspectives for future work.

 ISBN: 978-1-61208-230-1

II. CONTEXT

A. Safety-Critical Systems

Safety-critical systems are those that, in case of failure, will cause unacceptable drastic consequences to human beings and/or to the environment [12].

There are four concepts that are intrinsically related to safety-critical systems [12]: (i) *Mistake:* Cause of a fault happening during development; (ii) *Fault:* The adjudged or hypothesized cause of an error; (iii) *Error:* If the system is running, an error is an erroneous state that could lead to a failure; and (iv) Failure: An event that occurs when the system terminates its ability to provide the correct service. As perceived, a Failure is caused by an Error, which is caused by a Fault, which is a result of a Mistake. According to Avižienis et al. [12], all faults that might affect a system during its existence are part of specific Fault Classes, which, in turn, are grouped into specific Fault viewpoints. With respect to failures, Avižienis et al. [12] discuss *service failure modes*. A *service failure* happens when a service delivered by a system deviates from its correctness, and *service failure modes* are the different ways in which the deviations are perceived.

B. Safety Tactics and Safety Patterns

Safety tactics are architectural design decisions made to avoid or handle failures that safety-critical systems are subject to [2]. They are based on well-known fault-tolerant design methods, and were inspired by the notion of architectural tactics proposed by the Software Engineering Institute (SEI). Architectural tactics are "means of satisfying a quality-attribute-response measure by manipulating some aspect of a quality attribute model through architectural design decisions." [13]. Following the same principle, Wu and Kelly developed an analytic safety model focusing on the relationship between safety attributes and architectures with respect to failures. Based on this analytic safety model, they organized safety tactics into three categories: (i) tactics for failure avoidance, (ii) tactics for failure detection, and (iii) tactics for failure containment. To be compliant with the SEI's tactics approach, they proposed a hierarchical organization for such tactics, as illustrated in Figure 1.

Due to the nature of safety-critical systems, the decision about whether the tactics should be addressed at the software or at the hardware level are mainly driven by regulations, which state exactly where and how a tactic must be applied [2].

Safety tactics should be seen as the highest abstraction level of a safety pattern. A safety tactic will be addressed in a safety-critical system when any pattern (or combination of patterns) that implements the tactic is considered in the system architecture.

Safety patterns have been considered for years mainly in the Electrical Engineering field, due to the fact that safety-critical systems, in most cases, are the result of a very tight synergy between the hardware and the software embedded in electronic devices - so-called Embedded Systems [14]. However, methods originating from the Computer Science field have been widely considered in the development of such systems, mainly because the software portion of embedded systems is continuously acquiring more responsibility. In this regard, safety patterns have been considered a topic of interest for software engineers.

Some aspects of safety patterns that make them special and demand specific mechanisms for dealing with them are [3]: (i) they can be essentially formed by roles that represent software or hardware entities, or by a combination of both, demanding specific ways to show how these entities are related; (ii) a great number of roles are common to many safety patterns, differentiating basically in how they are connected and distributed along execution channels. Execution channels are pipes comprised of roles that sequentially transform input data into output data [3]; (iii) each safety pattern is meant to avoid, detect, or do the containment of a failure and faults [12]. Therefore, it is necessary means to deal with the reuse of roles while modeling safety patterns, which will ensure that the roles are properly connected and are associated with the proper execution channel. Moreover, there must exist means to indicate the fault class and service failure mode that the safety pattern is supposed to handle, as well as the safety tactic that the safety pattern implements.

With respect to the description and representation of safety patterns, we considered the general pattern description principle proposed by Alexander [15] that, when reasoned in our context, states that a pattern description also provides means with which can be observed how the resulting system architecture will look like after the application of a pattern. This generative property enforces that a pattern description should not only show the characteristics of a pattern, but coach how to apply it [16]. In this regard, we understand that a safety pattern constitutes a set of entities related to each other by specific rules, which, by definition, represent knowledge that states actions to be followed for the achievement of a purpose [17].

C. UML Profiles

UML Profiles are mechanisms for specifying rules to be used in parts of the model of a system where specific constraints are required [7]. Profiles are based on stereotypes and tags, which are the concrete entities that must be applied to UML elements such as classes, components, and connectors, with the aim of ruling parts of a system with respect to constraints of the domain or of the modeling process.

III. RELATED WORK

In the computer science field, software pattern specification and representation have been widely discussed, mainly after the work of GoF (Gang of Four) [18]. The

 ISBN: 978-1-61208-230-1

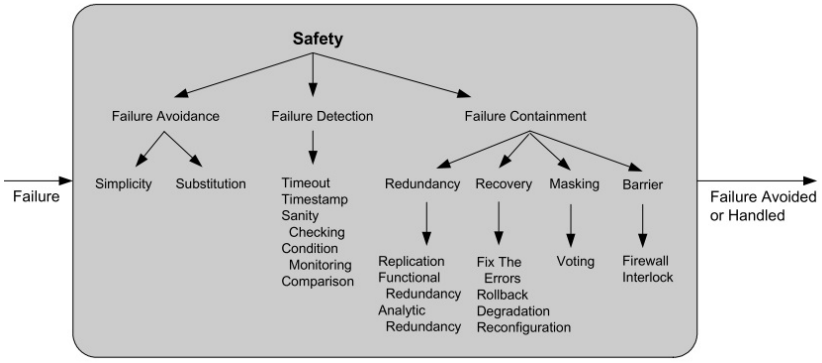

Figure 1. Hierarchical organization of safety tactics (extracted from [2]).

UML's official pattern representation approach is based on a parameterized collaborations model, rendered in a way similar to UML template classes [7]. Rosengard and Ursu [19] proposed an ontological representation for patterns. Mak et al. [20] proposed an extension to UML 1.5, using meta-modeling techniques and collaboration diagrams to specify the collaboration among the elements of the model. Guennec et al. [21] proposed the use of UML collaboration models combined with the Object Constraint Language (OCL) for representing patterns. Eden et al. [10] proposed a declarative and higher-order language, called LePUS, to represent generic solutions indicated in the patterns. Kim [9] proposed a language called Role-Based Meta modeling Language (RBML), which comprises abstract syntax, meta model level constraints, and constraint templates. Selonen et al. [11] established a language for defining profiles hierarchy, which is derived to support patterns representation using only UML Profiles.

With respect to safety patterns, Douglass [3] [22], Pullum [23], Koren and Krishna [24], and Hanmer [25] have documented a vast list of safety patterns. However, their presentation of these patterns focuses mainly on general information related to the general structure, the problem addressed, the context of use, and the consequences.

Regarding the representation of safety patterns, Armoush et al. [8] proposed an approach for representing safety patterns that consists of a traditional table template for pattern documentation, with a series of fields that address subjective information like the safety pattern's name, problems that it solves, and consequences of use. Such a canonical structural form of representing a pattern is basically useful for understanding the nature and purpose of the patterns, but does not offer any information to support the proper pattern application in architectural models.

An approach that is closer to ours is the one proposed by Tichy and Giese [26], which uses degradation rules to describe the structure and deployment restrictions of safety patterns and specify the behavior that is executed while degrading the systems functional or non-functional properties. Actually, such rules are used only to complement the structural and deployment documentation of safety patterns. On the other hand, our rules were built foreseeing automatized processes of safety patterns application in architectural models by means of model transformation mechanisms.

IV. Our Approach

Our approach for representing safety patterns aims at facilitating the modeling and application of safety patterns in the architectures of safety-critical systems. For this, it jointly uses:

1) *Graphic models of the safety pattern* that show (i) the structure of the safety pattern in terms of the roles that compose the pattern and how they are connected, modeled with elements defined in the *Safety UML Profile*; (ii) the safety tactic that the pattern is related to; (iii) the fault class, and (iv) the service failure mode for which the pattern is appropriate.
2) *Pattern Descriptive Rules*, which express detailed design constraints of the safety pattern, providing information that is useful to support the construction of statements used by mechanisms that perform automatic application of safety patterns modeled with the *Safety UML Profile*, in architectural models.

The remainder of this section covers: (i) the *Safety UML Profile*; (ii) how to model safety patterns using elements defined in the *Safety UML Profile*; and (iii) the Pattern Descriptive Rules.

A. Establishing the Safety UML Profile

We have defined an UML profile, the so-called *Safety UML Profile*, which aggregates *stereotype* elements representing the roles of each safety pattern. Actually, each role of a safety pattern becomes a stereotype of the *Safety UML Profile*. For instance, Figure 2 shows the roles of the Protected Single Channel Pattern (PSC). This pattern is composed of six roles: Input Sensor, Input Processing,

Data Transformation, Data Validation, Output Processing, and Actuator. Each role corresponds to a stereotype in the *Safety UML Profile*. The role Input Processing, for example, becomes the stereotype ≪Input Processing≫.

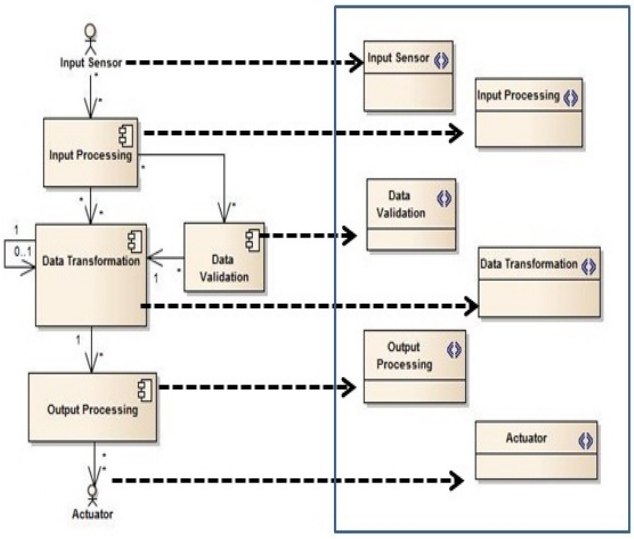

Figure 2. Roles of the Protected Single Channel pattern mapped to stereotypes of the Safety UML Profile.

We understand that a unique UML profile is enough to comprise the entities required for a concise representation of a safety pattern, due to the fact that the roles of the safety pattern are very often repeated in most patterns, differing basically in how these roles are linked, the execution channel that each role is part of, and in the quantity of elements that are present in a specific pattern. For instance, consider the PSC (shown in Figure 2) and the Triple Modular Redundancy pattern (TMR) [3] (shown in Figure 3). It can be seen that the TMR contains roles that are also present in the PSC, but that are replicated along three execution channels. TMR contains an additional role called *Voter*, while PSC has another one called *Data Validation*. To address the roles of the TMR, the Safety UML Profile shown in Figure 2 (which already contains the roles of the PSC) is modified by adding only a new stereotype that represents the role *Voter*.

Each stereotype representing a role of a safety pattern has associated with it a textual description of the role. Such information is useful for engineers to better understand the purpose of each role. There is no standardized way for the description. However, it must be detailed enough to ensure that the purpose of the role is clearly understandable.

Beyond the roles that compose the safety patterns, the *Safety UML Profile* also defines three other stereotypes: ≪Safety Tactic≫, ≪Fault Class≫, and ≪Service Failure Mode≫. These stereotypes are respectively used to indicate the safety tactic (cf. Figure 1) that the safety pattern is associated with, and the *fault classes* [12] and *service failure modes* [12] that a safety pattern solves when applied in

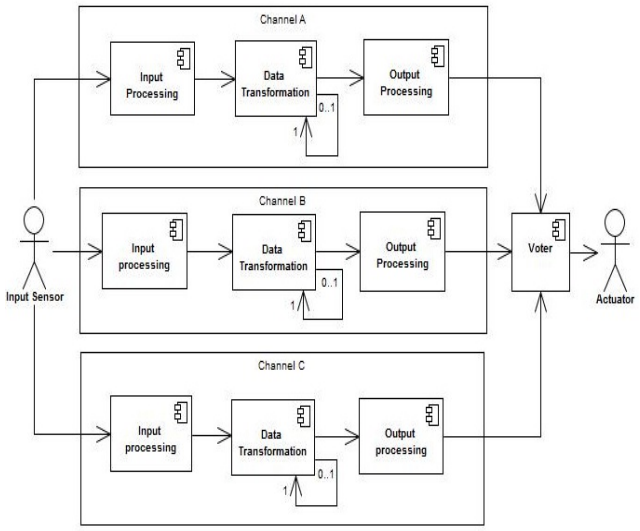

Figure 3. Triple Modular Redundancy pattern and its multiple execution channels (adapted from[3]).

the architecture of a safety-critical system. This is done by adding of three UML classes in the same diagram where the structure of a safety pattern is represented with instances of the *stereotypes* available in the Safety UML Profile. The class associated with the *fault classes* is stereotyped with the stereotype ≪Fault Class≫, and named according to the fault class that the safety pattern solves. The class associated with the *service failure modes* is stereotyped with the stereotype ≪Service Failure Mode≫, and named with the Service Failure Mode that the safety pattern solves. The class associated with the *safety tactic* is stereotyped with the stereotype ≪Safety Tactic≫, and is named according to the safety tactic that the safety pattern is associated with.

B. Representing Safety Patterns with Safety UML Profile

According to our approach, the structures of safety patterns are designed using instances of the *stereotypes* available in the Safety UML Profile. The reason for using instances is that we understand that the *stereotypes* defined in the profile are reusable elements, which are present in multiple patterns, since, as already mentioned, the same role is present in various patterns. For example, consider that the architect wants to model the PSC pattern (cf. Figure 2). In a separate diagram, he/she creates instances of the stereotypes available in the *Safety UML Profile* that comprise the PSC pattern, and connects these stereotype instances with directed links indicating the direction of the information flow, as shown in Figure 4.

Due to the fact that safety patterns can be composed of hardware, software, or both entities at the modeling level, it is important to reason on them in terms of functional entities, regardless of their roles as hardware or software. For instance, the *Data Transformation* role of the PSC pattern

 ISBN: 978-1-61208-230-1

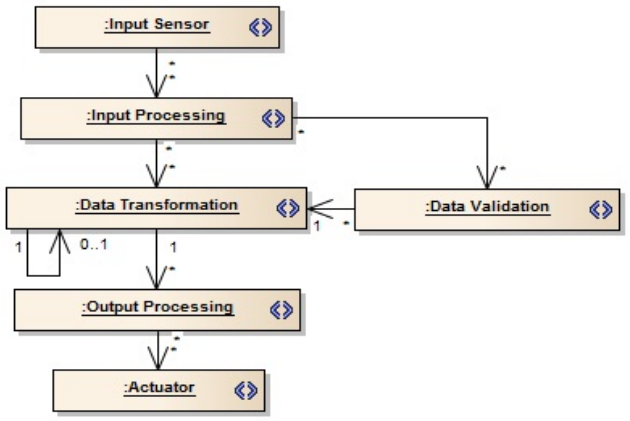

Figure 4. Protected Single Channel pattern specified with instances of stereotypes defined in the *Safety UML Profile*.

can be a hardware or a software entity. Representing the roles with stereotypes offers the flexibility of mapping the *Data Transformation* of the PSC pattern, for instance, to software components, deployment units, or any other UML element of the architectural model, once they are conceptual entities.

If we take a closer look at Figure 2, we observe that the roles of the PSC pattern, except for *Input Sensor* and *Actuator*, are surrounded by a boundary called *Channel*. A particular characteristic of safety patterns is that the roles that compose them, with the exception of Sensors and Actuators, run under execution channels [3]. Our way of dealing with this characteristic is to use Tagged Values [7]. Each stereotype instance used for designing a specific safety pattern is tagged with *Name* ≡ ***ExecutionChannel*** and *Value* ≡ ***Name of the Channel***. When the roles are under a unique execution channel, as in the PSC pattern, each stereotype instance is tagged with a tag that has *Tag Name* ≡ ***ExecutionChannel***, and *Tag Value* ≡ ***Channel A***. In other patterns, such as the TMR pattern, the roles are under multiple execution channels (cf. Figure 3). In this case, the stereotype instances of this pattern (cf. Figure 5) are tagged in accordance with the execution channel they are part of. This means that the three instances of the stereotype that represents the role *Input Processing*, for instance, are tagged differently, due to the fact that they run under different execution channels. Their tags will have *Tag Name* ≡ ***ExecutionChannel***, and *Tag Value* ≡ ***Channel A***, ***Channel B***, and ***Channel C***, respectively. On the other hand, the roles *Input Sensor*, *Voter*, and *Actuator* will have *Tag Name* ≡ ***ExecutionChannel*** and *Tag Value* ≡ ***null*** because they are not associated with any execution channel.

With respect to the Fault Class, Service Failure Mode, and Safety Tactic of the TMR pattern, consider Figure 5, which shows the TMR pattern specified with stereotypes instances. On the bottom left there are (i) a class called *Random Fault*, stereotyped with ≪Fault Class≫, (ii) a class called *Content Failure*, stereotyped with ≪Service Failure Mode≫, and (iii) a third class called *Failure Containment*, stereotyped with ≪Safety Tactic≫. This indicates that the TMR pattern is a concretization of a failure containment tactic, and is appropriate for dealing with random faults and content failures.

C. Pattern Descriptive Rules for representing Design Constraints of Safety Patterns

We understand that a safety pattern constitutes a set of entities related to each other by a specific rule. Therefore, we propose a set of rules, which we call Pattern Descriptive Rules, which were designed to support the future construction of statements used by mechanisms that perform automatic application of safety patterns in architectural models. An example of such a mechanism are Model Transformation rules, which provide indications on how the structure of a model is orchestrated in the model transformation processes, in terms of which elements should be created, updated, or deleted [27].

Our Pattern Descriptive Rules are structured to describe each specific role that participates in a safety pattern, which was previously represented with instances of stereotypes defined in the *Safety UML Profile*. As already mentioned, we argue that the participation of a role in a safety pattern composition is determined by (i) the information flow among the roles in each pattern; (ii) the quantity of instances of the same role in each safety pattern; and (iii) how the instances of the same role are distributed along execution channels. For example, in the PSC Pattern (cf. Figure 4), there is only one instance of the *Output Processing* role, which receives input from the *Data Processing* and provides output to the *Actuator*. On the other hand, in the TMR pattern (cf. Figure 5), there are three instances of the *Output Processing* role, each one in a different channel, and all of them providing output to the *Voter* role.

In this regards, our approach states that for each role that participates in a safety pattern:

1) There is one Pattern Descriptive Rule representing the participation of the role (this includes information about the execution channel where the role is located).
2) There is one Pattern Descriptive Rule describing to which other roles the output flow is directed to.

For instance, for the *Input Processing* role in the PSC pattern (cf. Figure 4), there is one Pattern Descriptive Rule related to the role itself and its execution channel, and one Pattern Descriptive Rule related to the connections between the *Input Processing* and the *Data Transformation*, and another one between the *Input Processing* and the *Data Validation*.

The Pattern Descriptive Rule that we propose for representing *a role and its execution channel* is:

 ISBN: 978-1-61208-230-1

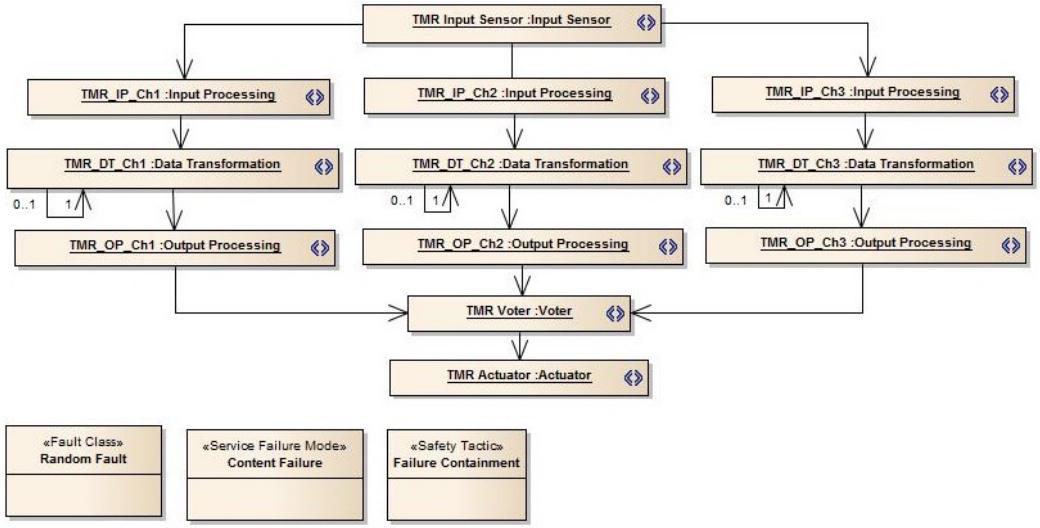

Figure 5. Triple Modular Redundancy pattern specified with elements of the *Safety UML Profile*.

rule *Rule Name:*
σ*:UML* $\cup$ *((*$\epsilon \in \kappa$*)* $\in \psi$*: Safety UML Profile)*
where,

$\sigma \equiv$ *Architectural model, modeled according to the standard UML meta model, where the safety pattern will be applied.*
$\epsilon \equiv$ *UML element representing the role instance.*
κ *Execution channel where the role instance is located.*
$\psi \equiv$ *Safety pattern modeled with instance of stereotype defined in the Safety UML Profile.*

This Pattern Descriptive Rule is read as: *The architectural model* σ *is modified by the insertion of a UML element that represents the role instance* ϵ*, which is tagged with information related to the execution channel* κ *and composes the safety pattern* ψ.

As already mentioned, if a role instance is not part of an execution channel, like the Input Sensor and Actuator in the PSC, and Input Sensor, Actuator, and Voter in TMR, the tags will have ***Tag Name*** $\equiv$ *ExecutionChannel*, and ***Tag Value*** $\equiv$ *null*.

The Pattern Descriptive Rule that we propose for representing *connections between roles in a safety pattern* is:

rule *Rule Name:*
$\tau \cup (\sum \lambda \in (\epsilon \in \kappa))$
where,

$\tau \equiv \sigma$*:UML* $\cup$ *(*$\sum \epsilon \in \psi$*: Safety UML Profile), i.e., Architectural model* σ *containing all the roles* ϵ *that compose the safety pattern* ψ *being applied, but without presenting connections among the role instances.*

$\lambda \equiv$ *one connection originated in the element representing a specific role* ϵ *that is part of an execution channel* κ.

This Pattern Descriptive Rule is read as: *The architectural model with all the roles* ϵ *that compose the safety pattern* ψ *being applied is modified by the insertion of all the connections* λ *originated in the element that represents a specific role* ϵ*, which, in turn, is part of an execution channel* κ.

For example, consider the PSC pattern (cf. Figure 4). The rule that represents the *Input Processing* role of this pattern and its execution channel is:

rule *Input Processing of the PSC pattern Rule:*
M1:UML $\cup$ *((Input Processing* $\in$ *Channel A)* $\in$ *PSC pattern: Safety UML Profile)*

This rule states that a UML element, stereotyped with *Input Processing* and tagged with *Tag Name* = ***ExecutionChannel***, and ***Tag Value*** = ***Channel A***, must be introduced in the UML model M1 in the application of the PSC pattern. It is worth emphasizing that every role that composes the PSC pattern has a rule like this one. Consider now the existence of a model M2, containing six UML elements, one for each role that composes the PSC pattern. The Pattern Descriptive Rule that represents connections originated in the *Input Processing* role is:

rule *Connections of the Input Processing of PSC pattern:*
M2 $\cup$ *(Connections Input Processing(Channel A): Data Transformation(Channel A), Data Validation(Channel A)), where:*

M2 = (M1 $\cup$ *(*$\sum$*Roles* $\in$ *PSC pattern: Safety UML Profile))*

 ISBN: 978-1-61208-230-1

This Pattern Descriptive Rule states that the model M2 is modified by adding the connections between the elements representing the *Input Processing* role and the elements that receive its output: Connection 1 = Input Processing and Data Transformation; Connection 2 = Input Processing and Data Validation. In this case, all the role are under the same execution channel. However, it is important to have such indication for the case of association of roles in different execution channels.

The graphical representation of safety patterns with stereotypes instances is an appropriate front-end that allows engineers to reason on the safety pattern structure (roles and connections) in terms of abstract functional entities. Moreover, it provides means to state safety specificities (fault classes, services failure modes, and safety tactics) that are singular to each safety pattern. Our Pattern Descriptive Rules state actions that foresee the automatized application of safety patterns in architectural models, providing fundamental highlights on how the artifacts necessary to perform the complete pattern application using Model Transformation mechanisms should look like. When combining the graphical representation with the Pattern Descriptive Rules, engineers have means to represent the pattern, taking in consideration not only the pattern design, but also explicitly indicating application constraints.

V. Representing the Homogeneous Redundancy Pattern with our Approach

We have represented with our approach the safety patterns proposed by Douglass [3] and observed that they become easier to represent and reuse. Due to space constraints, however, for this section, we selected the Homogeneous Redundancy pattern to be represented with our approach. This pattern is composed of seven roles: *Input Sensor, Input Processing, Data Transformation, Data Validation, Output Processing, Actuation Validation,* and *Actuator*. Consider that initially, the *Safety UML Profile* contains only three stereotypes: ≪Fault Class≫, ≪Service Failure Mode≫, and ≪Safety Tactic≫. The first step is to create stereotypes on it that represent the roles of the Homogeneous Redundancy pattern in the *Safety UML Profile*, as shown in Figure 6. At this point, each role has an associated documentation providing general information of it, as can be seen in Table I.

After having the roles available in the Safety UML Profile, the pattern structure is created in a separate diagram using instances of these stereotypes (cf. Figure 7). To clearly differentiate between the execution channels that the roles are part of, the roles in light gray are part of the *Primary Actuation Channel*, the ones in dark gray are part of the *Secondary Actuation Channel*, and the white elements are sensors and actuators that are not part of any channel. The roles that are part of the *Primary Actuation Channel* are

Role Name	Role Description
Input Sensor	This is the source of the information used to control the actuator.
Input Processing	Acquires and performs the first processing on the data sent by the Primary Input Sensor.
Data Transformation	Performs a single transformation step on the input data.
Data Validation	Validates if the data is correct or reasonable, and also stops the processing on the current channel and begins it on the second channel when a fault is detected.
Actuation Validation	Compares the output to the commanded output, and determines when some application specific fault occurs.
Output Processing	Performs the last stage of the data transformation, and controls the Actuator.
Actuator	This is the device performing the actuation. This is the actuator used by default.

Table I
DESCRIPTION OF ROLES THAT COMPOSE THE HOMOGENEOUS REDUNDANCY PATTERN.

tagged with *Tag Name* ≡ ***ExecutionChannel***, and *Tag Value* ≡ ***Primary Actuation Channel***. The roles that are part of the *Secondary Actuation Channel* are tagged with *Tag Name* ≡ ***ExecutionChannel***, and *Tag Value* ≡ ***Secondary Actuation Channel***. The sensors and actuators are tagged with *Tag Name* ≡ ***ExecutionChannel*** and *Tag Value* ≡ ***null***. The three classes on the bottom left side of Figure 7 represent the Fault Class, Service Failure Mode, and Safety tactic related to the Homogeneous Redundancy pattern, which are Random Fault, Content and Timing, and Functional Redundancy, respectively.

For each role that composes the Homogeneous Redundancy pattern, there is one rule representing the role and its execution channel, and one rule describing to which other roles its output flows are directed. For this example, consider the UML model called *SourceModel* as the original model where the Homogeneous Redundancy pattern is to be applied, and another UML model called *TargetModel* as the model that already contains elements representing every role of the Homogeneous Redundancy pattern. The rule representing the role *Input Sensor* (so called Primary Input Sensor) associated with the Primary Actuation Channel is:

rule *Primary Input Sensor of Homogeneous Redundancy:*
SourceModel:UML ∪ *((Primary Input Sensor* ∈ *null)* ∈ *Homogeneous Redundancy pattern: Safety UML Profile)*

This Pattern Descriptive Rule is read as: *The architectural UML model SourceModel is modified by the insertion of an UML element that represents the role Primary Input Sensor, which is tagged with Tag Name* = ***ExecutionChannel****, and* ***Tag Value*** = ***null****, and composes the safety pattern Homogeneous Redundancy.* It is worth highlighting that the

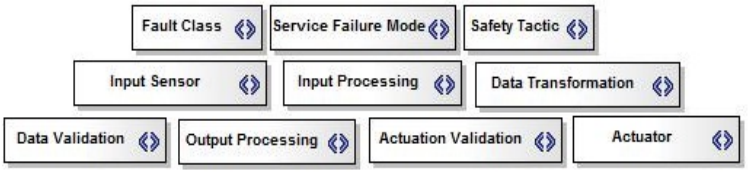

Figure 6. Safety UML Profile with the Roles of the Homogeneous Redundancy pattern.

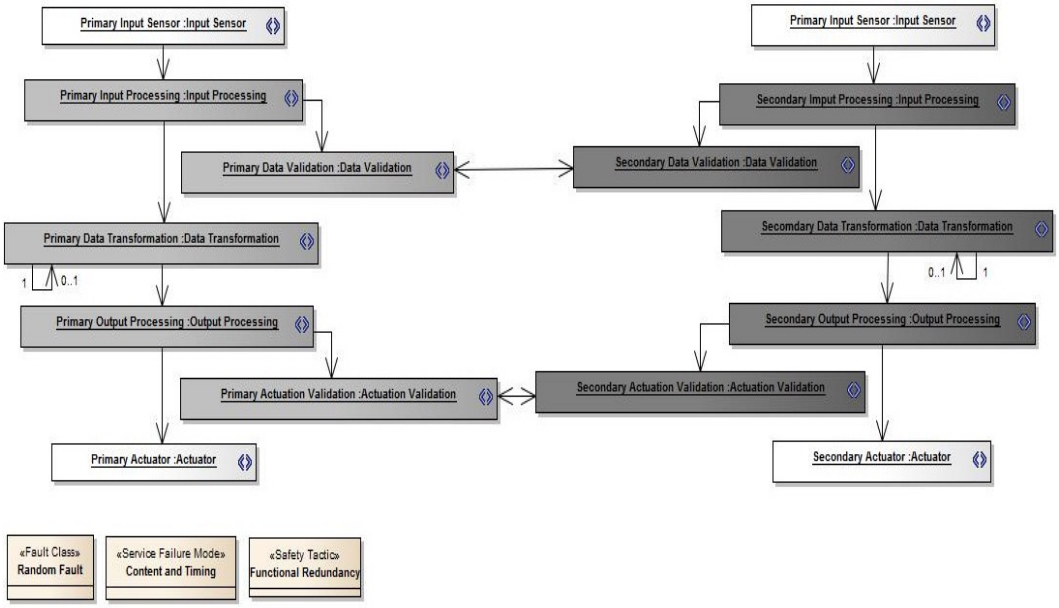

Figure 7. Homogeneous Redundancy pattern modeled with instances of stereotypes defined in the *Safety UML Profile*.

tag ***ExecutionChannel*** has a value ***null*** because it is not part of any Execution channel.

The following Pattern Descriptive Rule is related to the Primary Input Processing, and is read as: *The architectural UML model SourceModel is modified by the insertion of an UML element that represents the role Primary Input Processing, which is tagged with Tag Name = **ExecutionChannel**, and **Tag Value = Primary Actuation Channel**, and composes the safety pattern Homogeneous Redundancy.*

rule *Primary Input Processing of Homogeneous Redundancy:*
SourceModel:UML ∪ ((Primary Input processing ∈ Primary Actuation Channel) ∈ Homogeneous Redundancy pattern: Safety UML Profile)

The rules for the others roles that are part of the Primary and Secondary Actuation channel, as well as the Secondary Input Sensor and both actuators, follow the same construction principle. For instance, the rules representing the Secondary Data Validation role and the Secondary Actuator are as follows:

rule *Secondary Data Validation of Homogeneous Redundancy:*
SourceModel:UML ∪ ((Secondary Data Validation ∈ Secondary Actuation Channel) ∈ Homogeneous Redundancy pattern: Safety UML Profile)

This Pattern Descriptive Rule is read as: *The architectural UML model SourceModel is modified by the insertion of an UML element that represents the role Secondary Data Validation, which is tagged with Tag Name = **ExecutionChannel**, and **Tag Value = Secondary Actuation Channel**, and composes the safety pattern Homogeneous Redundancy.*

rule *Secondary Actuator of Homogeneous Redundancy:*
SourceModel:UML ∪ ((Secondary Actuator ∈ null) ∈ Homogeneous Redundancy pattern: Safety UML Profile)

The Pattern Descriptive Rule above, which describe the Secondary Actuator of Homogeneous Redundancy pattern, is read as: *The architectural UML model SourceModel is modified by the insertion of an UML element that represents the role Primary Actuator, which is tagged with Tag Name = **ExecutionChannel**, and **Tag Value = null**, and*

composes the safety pattern Homogeneous Redundancy. As with the Primary Input Sensor peviously mentioned, the tag ***ExecutionChannel*** has the value ***null*** because it is not part of any Execution channel.

For describing the rules that represent the connections of the roles of the Primary and Secondary Actuation Channel, as well as the Input Sensors and Actuators, consider the *SourceModelWithRoles* as the original source model *(SourceModel)* modified by the addition of representatives for every role of the Homogeneous Redundancy pattern (or, using our notation, SourceModelWithRoles = (SourceModel $\cup$ ($\sum$Roles $\in$ Homogeneous pattern: *Safety UML Profile*)).

The following Pattern Descriptive Rule represents the connections originated in the Primary Input Sensor that connects it with the Primary Input Processing, and is read as: *The architectural model containing all the roles that compose the safety pattern Homogeneous Redundancy is modified by the insertion of the connections that have origin in the element that represents the role Primary Input Sensor, and that link it with the element that represents the role Primary Input Processing, which, in turn, is part of the Primary Execution Channel.*

rule *Connections of the Primary Input Sensor of Homog. Redundancy Pattern:*
SourceModelWithRoles $\cup$ (Connections Primary Input Sensor: Primary Input Processing (Primary Actuation Channel)

The rule representing the connections originated in the Primary Input Processing is:

rule *Connections of the Primary Input Processing of Homog. Redundancy Pattern:*
SourceModelWithRoles $\cup$ (Connections Primary Input Processing: Primary Data Transformation (Primary Actuation Channel); Primary Data Validation (Primary Actuation Channel))

This rule is read as: *The architectural model containing all the roles that compose the safety pattern Homogeneous Redundancy is modified by the insertion of the connections that have their origin in the element that represents the role Primary Input Processing (part of the primary execution channel), and that link it with the elements that represent the role Primary Data Transformation and the role Primary Data Validation, which are both part of the Primary Execution Channel.*

The rule that represents the connections originated in the Primary Data Validation is:

rule *Connections of the Primary Data Validation of Homog. Redundancy Pattern:*
SourceModelWithRoles $\cup$ (Connections Primary Data Validation: Secondary Data Validation (Secondary Actuation Channel))

This rule should be read as: *The architectural model containing all the roles that compose the safety pattern Homogeneous Redundancy is modified by the insertion of the connections that have their origin in the element that represents the role Primary Data Validation (part of the primary execution channel), and that link it with the element that represents the role Secondary Data Validation, which, in turn, is part of the Secondary Execution Channel.*

As our approach requires two rules per role that compose a safety pattern (one rule describing the role itself and its execution channel, and another one describing the connections originated in the role), the complete representation of the Homogeneous Redundancy pattern consists of the model shown in Figure 7, and additional twenty eight Pattern Descriptive Rules. Therefore, due to space limitation it is not possible to show all the rules. However, as already mentioned, they look similar to the ones previously shown, with the appropriate changes of the connected roles instances and the actuation channel they are part of.

It is important to emphasize that the rules described in this work are not enough to ensure the application of a safety pattern in an architectural model. Actually, our Pattern Descriptive rules are to Model Transformation rules as algorithms are to Imperative languages. It means that they are language independent and, when considered together with the information available in the graphical model (for instance Figure 7), offer the basis for constructing model transformation rules in languages like TRL or ATL [27], which will ensure the automatic application of safety patterns in architectural models of safety critical systems, by means of model transformation mechanisms.

VI. Conclusion and Future Works

This work was motivated by the lack of ways to represent safety patterns in a way that engineers can reason on the safety patterns composition in terms of the functional entities that comprise them, and on the safety patterns constraints that should be considered when applying them in architectural models. To fill this gap, we have proposed a safety pattern representation approach that consists of the joint use of: (i) a graphical representation of safety pattern using elements defined in a UML profile called *Safety UML Profile*; and (ii) a set of pattern descriptive rules that describe each role participation in the safety patterns, its execution channel, and the connections of each role.

Being able to express safety patterns with elements of a UML profile and pattern descriptive rules in a unified fashion is the first step in our attempt to perform safety pattern applications in architectural models of safety-critical systems by means of Model Transformation mechanisms.

Acknowledgements: This work is supported by the Fraunhofer-Innovation Cluster Digitale Nutzfahrzeugtechnologie (Digital Engineering for Commercial Vehicles).

 ISBN: 978-1-61208-230-1

REFERENCES

[1] P. Liggesmeyer and M. Trapp, "Trends in embedded software engineering," *IEEE Software*, vol. 26, no. 3, pp. 19–25, May 2009.

[2] W. Wu and T. Kelly, "Safety tactics for software architecture design," in *in Proceedings of the 28th Annual International Computer Software and Applications Conference, (Hong Kong, 2004)*. IEEE Computer Society, 2004, pp. 368–375.

[3] B. P. Douglass, *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[4] O. M. G. Group. UML specification, version 2.0.

[5] N. Rozanski and E. Woods, *Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley Professional, 2005.

[6] L. Fuentes-Fernández and A. Vallecillo-Moreno, "An introduction to uml profiles," *Journal of UML and Model Engineering*, vol. 2, 2004.

[7] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

[8] A. Armoush, "Design patterns for safety-critical embedded systems." Ph.D. dissertation, RWTH Aachen University, 2010, http://d-nb.info/1007034963.

[9] D.-K. Kim, R. France, S. Ghosh, and E. Song, "A role-based metamodeling approach to specifying design patterns," *Computer Software and Applications Conference, Annual International*, p. 452, 2003.

[10] A. H. Eden, Y. Hirshfeld, and A. Yehudai, "Lepus - a declarative pattern specification language," Department of Computer Science, Tel Aviv University, Tech. Rep., 1998.

[11] P. Selonen, M. Siikarla, K. Koskimies, and T. Mikkonen, "Towards the unification of patterns and profiles in uml," *Nordic J. of Computing*, vol. 11, no. 3, pp. 235–253, Sep. 2004.

[12] A. Avižienis, J. . Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004.

[13] F. Bachmann, L. Bass, and M. Klein, "Deriving architectural tactics: A step toward methodical architectural design," *Software Engineering Institute (SEI), Technical Report, No. CMU/SEI-2003-TR-004*, 2003.

[14] J. C. Knight, "Safety critical systems: challenges and directions," in *Proceedings of the 24th International Conference on Software Engineering*, ser. ICSE '02. New York, NY, USA: ACM, 2002, pp. 547–550.

[15] C. Alexander, *The Timeless Way of Building*. Oxford University Press, 1979.

[16] B. Appleton, "Patterns and software: essential concepts and terminology," *Object Magazine Online*, vol. 3, 1997.

[17] P. Avgeriou and U. Zdun, "Architectural patterns revisited - a pattern language," in *Proceedings of the 10th European Conference on Pattern Languages of Programs (EuroPLoP 2005)*, Irsee, Germany, Jul. 2005.

[18] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, 1994.

[19] J.-M. Rosengard and M. Ursu, "Ontological representations of software patterns," *Proceedings of KES'04*, 2004.

[20] J. K.-H. Mak, C. S.-T. Choy, and D. P.-K. Lun, "Precise modeling of design patterns in uml." in *ICSE'04*. Scotland: IEEE Computer Society, 2004, pp. 252–261.

[21] A. L. Guennec, G. Suny, and J.-M. Jzquel, "Precise modeling of design patterns." in *UML*, ser. Lecture Notes in Computer Science, A. Evans, S. Kent, and B. Selic, Eds., vol. 1939. Springer, 2000, pp. 482–496.

[22] B. P. Douglass, *Doing hard time: developing real-time systems with UML, objects, frameworks, and patterns*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[23] L. L. Pullum, *Software fault tolerance techniques and implementation*. Norwood, MA, USA: Artech House, Inc., 2001.

[24] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[25] R. Hanmer, *Patterns for Fault Tolerant Software*. Wiley Publishing, 2007.

[26] M. Tichy and H. Giese, "Extending fault tolerance patterns by visual degradation rules," in *Proc. of the Workshop on Visual Modeling for Software Intensive Systems (VMSIS) at the the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05), Dallas, Texas, USA*, 2005.

[27] K. Czarnecki and S. Helsen, "Classification of model transformation approaches," in *OOPSLA03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.

 ISBN: 978-1-61208-230-1