SOPHIA: a Modeling Language for Model-Based Safety Engineering

Daniela Cancila[1], Francois Terrier[1], Fabien Belmonte[2], Hubert Dubois[1], Huascar Espinoza[1], Sébastien Gérard[1], and Arnaud Cuccuru[1]

CEA LIST*, ALSTOM**

Abstract. Development of increasingly more sophisticated safety-critical embedded systems requires new paradigms, since manual approaches are reaching their limits. Experiences have shown that model-driven engineering is an approach that can overcome many of these limitations. Using model-based approaches however lead to new challenges regarding the cohesive integration of both safety engineering and system design along the system development process. In this paper, we present SOPHIA, a modelling language that formalizes safety-related concepts and their relations with system modelling constructs. We particularly focus on accident models and on how to achieve confidence that the frequency of possible accidents will be tolerable. In addition, we explore some strategies to implement SOPHIA as a complementary modelling language to SysML and reuse some useful constructs form the UML MARTE profile.

1 Introduction

In order to cope with the increasing design complexity of safety-critical systems, safety assurance should be considered as early as possible in the design process. Among other goals, safety assurance allows achieving confidence that the frequency of accidents will be acceptable. For this purpose, safety engineers need to specify all possible safety parameters that directly impact the software architecture design, and then to determine the probability rates of the deviation from fulfilling the system functions. The *Safety Integrity Level* (SIL) attribute is an example of such a parameter. The design of a given system and its subsystems changes according to the value of the SIL associated with each functionality of the system. Possible values range between "0" (less critical) and "4" (most critical) [19]. Thus, a system architecture including SIL4-functionalities must guarantee the maximum level of safety integrity, which would for example imply to add redundant hardware nodes.

* CEA LIST, Laboratoire d'Ingénierie dirigée par les modèles pour les Systèmes Embarqués Point Courrier 94, Gif-sur-Yvette, F-91191 France {daniela.cancila, francois.terrier, hubert.dubois, huascar.espinoza, sebastien.gerard,arnaud.cuccuru}@cea.fr

** Alstom Transport Information Solution 48 rue Albert Dhalenne, 93482 Saint-Ouen Cedex fabien.belmonte@transport.alstom.com

Recently, the railway safety community has proposed a new methodological guidance to enhance safety evaluation. In this proposal, SIL-to-function allocations exploit a new attribute named *Tolerable Accident Rate* (TAR). The TAR is defined as the "threshold between what is tolerable and what is undesirable with respect to the consequence of an accident" [8, 3]. In current industrial practice, the TAR is manually calculated, typically by using pre-defined tables. As the value of TAR influences the value of SIL, it may impact the architecture of a system. More precisely, the value of the TAR for a given accident (e.g. the head-on collision between trains) is used to calculate the value of the *tolerable hazard* for the same accident. Hence, we have a 1:1 correspondence between tolerable hazard and SIL. (We refer the reader interested in the technical details to [8, 3].)

Most of current practices on system safety assurance rely mainly on manual processes. They are therefore very dependent of the skill and experience of the engineers. This problem is exacerbated by the fact that safety engineering and software design domains have developed their own techniques and methodologies. Let us consider the example of the railway application domain. On the one hand, safety actors adopt standards that provide recommendations for safety assessment. Illustrative examples are fault tree analysis [22] and formal verification techniques, such as the B method [2]. On the other hand, actor from the software design and development community follow component-based techniques, such as [33, 9, 14]. In this context, defining the "right mapping" between safety models and models for software design/development is an essential challenge.

In order to avoid error-prone processes and to integrate both safety engineering and system design, we adopt a *model-based safety engineering* process. Model-Driven Engineering (MDE) [30, 31] is being successfully adopted in several industrial research projects [21, 18, 32, 29]. Two kinds of approaches are actually put into practice. In the first case, safety engineers and system designers share the same model of the system while using different views of it. In the second case, they use different models with clearly and formally defined relationships (using for example model transformation descriptors). In both cases, the direct benefits of MDE concern the possibility of automating part of the process of safety assurance, e.g. by automatically calculating certain information such as TAR parameters from the input safety parameters. This capability does not only simplify the process. It also enables to save time and, more importantly, it makes safety assurance as explicit part of an iterative design process. Indeed, new results can be more easily generated once the model has been changed. Moreover, the fact that models are more formally defined reduces the probability of introducing errors or omitting important details, since the analysis information is linked to the architectural model of the system.

As a main contribution within this paper, we introduce for the first time SOPHIA, a modelling language for safety concerns. SOPHIA provides an answer to safety industrial concerns by allowing designers to specify the safety attributes in a software design model. Our paper focuses on the infrastructure of SOPHIA, which is similar to that of MARTE [25]: It is based on a metamodeling, a profiling and a modeling space. As a result, SOPHIA has an independent language specification that can complement more general-purpose languages such

as UML or SysML [24]. At the profile level, we propose to use some suitable concepts from MARTE, the OMG's UML profile for real-time embedded systems [25], in particular the *Value Specification Language* (MARTE::VSL).

In Section 2, we discuss some related works and we identify fundamental criteria and principles for model-based safety engineering. In Section 3, we explain our industrial motivations. We also provide a rationale for SOPHIA as well as a description of its Fundamental Concepts. Moreover, we investigate the strategies regarding the integration of safety and software design. Section 4 is the central part of the paper. For the first time, we discuss the whole structure of SOPHIA: from its Fundamental Concepts to the implementation. In Section 5, we compare our approach with those given in Section 2. Finally, conclusions and on-going works are presented.

2 Related Work

Integrating safety concerns in general-purpose modelling process is a big challenge that has been explored in many directions. In this section, we focus on a few works which are receiving specific attention in the MDE community.

In order to study dependability in AADL (Architecture Analysis & Design Language) [1], P. Feiler and al. introduce a framework to model the error state propagations in a hierarchical architecture [17]. Error propagation can occur at component level (by composition of the components), at the hardware level (by interconnecting processors) and between the hardware and the components ("due to their binding to the execution platform" [17]). In order to limit, or even avoid, the error propagation, the authors provide suitable filters (guards), for example between the interconnection of components. In [17], P. Feiler and al. addresses error modelling as a complementary view to system architecture, which is an important topic related to safety. However, it does not cope with the problem of accident case modelling and the specification of safety attributes such as the SIL.

In order to complement AUTOSAR (the European industrial standard to specify component-based software infrastructures in automotive applications [6]), some European industries and academics have defined an architecture description language, so called EAST-ADL [5]. This includes requirements modelling, feature content at the level of a vehicle description, architecture variability, functional structure of applications, middleware, plant (environment), abstract hardware architecture, and preliminary functional allocation. In addition, EAST-ADL enables the modelling of system failure behaviour and allows analysis of that behaviour using safety analysis tools. In particular, EAST-ADL aimed at using a safety design flow compatible with that defined by the upcoming ISO 26262 standard, including support for concepts such as hazards, safety goals and requirements, and the representation of ASILs (Automotive SILs). Many of these concepts were represented in the first version of EAST-ADL, but there were many others not considered, e.g. accident and its consequences, or ASIL decomposition.

FTA is one of the main safety analysis tools. In [10], Douglass introduces a UML safety profile defining notions such as fault, hazard, and traceability of requirements. Such notions allow us to create fault tree analysis (FTA) diagrams and, hence, to study how "conditions and faults combine to create hazard". One of the main contributions of this approach is to adopt UML and its profiling mechanism to provide a common specification language to integrate safety and design activities. This facilitates the collaboration and common understanding between safety engineering teams and

system design teams. The underlying approach is the following. First, designers create a model with safety attributes, from which FTA is automatically generated. Engineers then study FTA and then they may manually change the model architecture. In other words, this approach does not deal with "safety reverse engineering". As a result, safety analysis is made *a posteriori*. When we deal with real industrial cases, a model quickly increases in complexity and in number of components. Consequently, it also occurs in the related FTA. The study of FTA is then a very complex work. In order to reduce such a complexity, one possible way is to *iteractivly* integrate safety engineering into model-based engineering of an architectural system. The underlying process is to have an automatic propagation of safety attributes in the architectural model such that it is correct with respect to "given safety requirements".

In [15], de Miguel and al. propose an approach similar to work [10]. Therefore, it has similar advantages and drawbacks. Finally, in [26], the authors introduces the UML profile for quality of service and fault tolerance analysis, called QoS&FT profile. In this profile, some aspects of safety analysis are covered (such that fault, errors, fault, non desirable events, etc). Notions, such as accidents and SIL are however not here considered.

3 Safety Engineering

This section provides some background information that have been taken into account for the definition of SOPHIA. Before describing the safety fundamental concepts, we want to discuss the high-level requirements for safety modelling from an industrial perspective.

Standards EN 50126 [11], EN 50128 [12] and EN 50129 [13] define a safety process plan for programmable electronic signalling devices including risk evaluation, SIL to function mapping and the life cycle recommendations by SIL. In particular, these standards recommend applying fully formal specification to ensure SIL 4. It means that engineers must provide mathematical proven demonstration for the safety properties of a given component.

Typically, in industry there is a gap between formal methods and textual system specifications, as well as between subsystem specifications. The main reason for this gap is that there is no standard and common language can be used to capture the different aspects. Semi-formal modelling approaches can bring a common basis to interconnect these different specification aspects. This is the main motivation for formalizing safety attributes into system models, from the early phases of the development process.

Therefore, SOPHIA has the following objectives:

1. enabling the specification of safety attributes in the architectural model of as sytem;
2. automating the calculation of some safety parameters in order to afford model-based engineering of safety;
3. providing an environment for system development in which coherence (compatibility of all requirements at the same level of abstraction, i.e., horizontal development) and correction ("good" decomposition of parent requirements into children requirements abstraction, i.e., vertical development) properties can be guaranteed by construction and/or verified a posteriori.

Provided these general needs, we present in the next section an excerpt of some fundamental concepts of SOPHIA related to accident case concerns

3.1 SOPHIA Fundamental Concepts

The concepts of SOPHIA (and their relationships) are based on Alstom Ontology [7] and on Alstom works such as [8, 3], which model their safety domain knowledge. In this section, we will use metamodels to describe the Fundamental Concepts of SOPHIA. They are organized into a set of packages and libraries. We use packages to introduce notions and their relationships, and we use libraries to specify data types. The package SOPHIA Fundamental Concepts contains two main subpackages, so called respectively SystemDesing and SafetyConcepts. Package SystemDesing specifies the relationships between safety concepts and model elements of a system. Package SafetyConcepts contains the following packages:

- **package ACCIDENTS,** which describes notions and relationships that are involved in an *accident*.
- **package MITIGATIONS,** which describes notions and relationships about mechanisms (*barriers*) to mitigate an *accident*;
- **package FaultContainmentRegion,** which describes notions and relationships that are involved in *error propagations*.

In this paper, we focus on package ACCIDENTS. Our intent is to show the details of the whole language design chain, from the formalization of the TAR attribute in the SOPHIA Fundamental Concepts, to the language implementation details. The result of our work is a first, but firm step towards model-based safety engineering.

Figure 1 shows some notions of package ACCIDENTS. Among them, we depict the TAR attribute. The notions are represented as metaclasses.

Hazard is "an event observable at the system boundary, which has potential either directly or in combination with other factors (external to the system), for giving rise to an accident at railway system level" [3].

AccidentCase is an unintended event with undesirable outcomes. AccidentCase leads to AccidentConsequenques. An AccidentCase is identified by the following properties: a unique ID; an AccidentType chosen from a statically pre-defined list;
AutomaticTolerableAccidentRate and TolerableAccidentRate.

AutomaticTolerableAccidentRate is the maximum rate of occurrence that is tolerable for a likely Accident [3]. It is specified by a number of events per hour (real number) and it is derived from the frequency and the severity of an accident.

TolerableAccidentRate and AutomaticTolerableAccidentRate are similar properties. The only difference is that TolerableAccidentRate is manually set by safety engineers when they have to deal with exceptional cases (i.e., for which a pre-defined table is not available). In Figure 5, Table "a:" identifies the *Risk Tolerability* of an accident. It is described with combinations of the following properties: severity of the consequences and frequency of the accident. TolerableAccidentRate is undefined by default. However, if TolerableAccidentRate is set to a different value than undefined, then it has a higher priority with respect to AutomaticTolerableAccidentRate. The importance of having both properties (i.e., one automatically specified and the other one manually set), is that: 1.) the modelling process can be automated in a *correct* way that respects table *Risk Tolerability*, 2.) we have traced to the computation, which is automatically derived from table *Risk Tolerability*. Furthermore, we can identify the divergence points specified by the exceptional cases. In case of divergence, designers must motivate their choices with respect to the value automatically calculated from table *Risk Tolerability*. From an implementation standpoint, designers could motivate their decisions in a suitable dialog box. The implementation of this latter is part of our ongoing work.

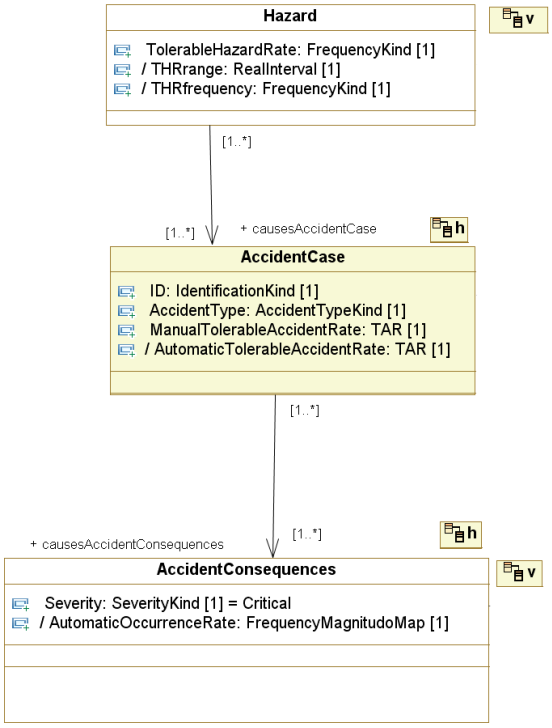

Fig. 1. SOPHIA : AccidentCase

AccidentConsequences is the result of a given AccidentCase. It is defined by the severity of the consequences with respect to the given AccidentCase. Severity may take only one of the following four predefined values: Catastrophic, Critical, Marginal, or Insignificant. These values of severity are captured by an enumeration which is part of our SOPHIA Fundamental Model library.

Next, we discuss the strategies to integrate the safety conceptual concepts defined above with a given general-purpose modelling language, in this case SysML.

3.2 Integration Strategies for SOPHIA and SysML

SysML was chosen by Alstom since it is an OMG standard specification for modelling of complex systems. Although SysML provides a formalism to manage requirements and system design together, SysML is lacking of concepts for dealing with specific concerns of safety. We have three possible strategies to integrate SOPHIA and SysML.

Strategy a defines SOPHIA as an extension to SysML. It has the advantage to be optimally tailored to the aimed integration with SysML. One of the main drawbacks of this strategy is that safety concepts will strongly depend on SysML. Then, any modification of SysML might lead to a modification in the SOPHIA extensions. In addition, safety concepts and SysML are conceptually disjoint, although complementary, and directly extending SysML does not make sense in our context.

Strategy b defines SOPHIA from scratch, i.e. as a pure domain-specific modeling specification language (DSML) (i.e. independently of UML) and then combining this metamodel with SysML. Consequently, *Strategy b* surmounts the drawbacks of *Strategy a*: safety concepts are independent not only of SysML but also of UML. It provides a framework that is fully dedicated to safety concepts and it is independent from other formalisms. As discussed in work [16], *Strategy b* has the following drawback: having safety models defined using two independent formalisms leads to strong difficulties for interfacing both types of models of the same system. This is particularly problematic for tracing safety information with the system architecture models. This problem is mainly reflected at tool level, since traceability always imply an important endeavour.

Strategy c proposes to firstly introduce SOPHIA as a package of Fundamental Concepts via a metamodel, in a way that is independent of the UML formalism. In a second stage, this metamodel (also called *domain model*) is implemented as a UML profile. In this way, we overcome the drawbacks of *Stategies a* and *b*, because the concepts are defined independently of UML, and, thereby, gain the benefits of a domain-specific approach. Moreover, this approach improves tool interoperability and facilitates the interface and traceability between different modelling aspects of the same system. SOPHIA and SysML languages (which are both designed as UML profiles) may indeed be used jointly in the same UML tool. A successful example of this approach is MARTE [25].

	Language Engeneering Domain		End User Domain	Tool environment
	Metamodel	Profile	One single Model	One single Tool
strategy a	NO	YES	YES	YES
strategy b	YES	NO	NO	NO
strategy c	YES	YES	YES	YES

Fig. 2. Strategies to integrate safety modeling language in the system architecture

4 From SOPHIA Safety Concepts to Implementation

4.1 SOPHIA Architecture

We adopt *Strategy c* and we develop it further. First af all, we strategically use the definition of profile, firstly introduced by S. Cook, and successfully adopted by other researchers: a profile is a family of related languages. It suggests the idea of exploiting the composition of pre-existing profiles. Indeed, our intent is not to define completely "new" metamodel and profile, covering all concepts from safety to architectural design. Our intend is indeed to reuse the existing work as much as possible, so that we can take advantage of pre-existing works and related tools.

In spite of SysML role for requirements and system's architecture (requirement and block diagrams), SysML lacks in the specification of temporal attributes [4]. Several European research projects are therefore willing to define a combined usage of both SysML and MARTE.

In the context of SOPHIA, we were particularly interested in MARTE to define non-functional properties (`MARTE::NFP`) and `MARTE::VSL` to valuate these properties. The `MARTE::NFP` package allows designers to annotate a UML model with non-functional properties. `VSL` stands for *Value Specification Language* and allows designers to specify "parameters/variables, constants, and expressions in textual form" [25]. Moreover, `VSL` supports arithmetic and logical expressions. Beyond the benefits of the SOPHIA alignment with a recognized international standard, the main advantage of this integration is the ability to support a well-formed syntax and semantics for safety parameters and to consequently enable automated derivation of dependent safety variables (See Section 4.2).

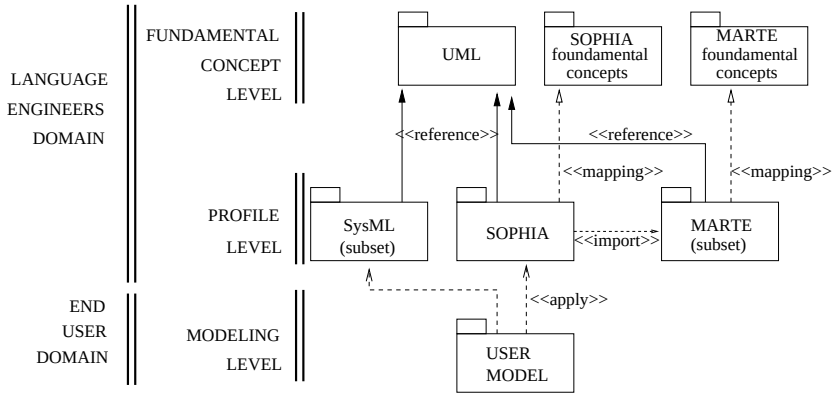

Fig. 3. Overall Structure

Figure 3 shows the overall structure. We have two main domains: end user domain and language engineering domain, which is in turn subdivided in two levels, Profile and Fundamental Concept. End user domain corresponds to M1 level in the OMG four-level hierarchy [23]. Designers only work in the modeling level. The language engineering domain corresponds to M2 level in the OMG four-level hierarchy. In the Profile level, we specify namesake profiles. In the Fundamental Concept level, we have UML metamodel and data (Fundamental Concepts for safety and MARTE in the figure). In the following, we discuss the overall structure, as illustrated in Figure 3.

At Modelling level, designers specify the model of a system. In order to specify the architecture of a system and associated requirements, designers need to apply SysML to their UML model. Next, designers annotate the model with safety attributes by applying the SOPHIA profile. In order to specify temporal attributes, designers exploit the MARTE stereotypes that are already imported by SOPHIA.

At Profile level, we have suitable languages of (at least) three families (profiles): SysML, SOPHIA and MARTE. One of our on-going works is to identify the minimum subset of SysML and MARTE to specify the requirements given by Alstom.

SysML is a UML profile. In Figure 3, UML stereotype "reference" shows such a relationship [27]. Note that SysML is only introduced as a UML extension and, then, SysML intentionally has not a fundamental concepts level.

SOPHIA is a UML profile for safety modelling modelling whose definition is based on SOPHIA Fundamental Concepts. In UML, there is not a specific symbol between a profile and its fundamental concepts. To explicitly capture this relationship, we use a dashed arrow annotated with the word "mapping", following the OMG notation introduced in work [28].

Like SOPHIA, MARTE extends UML and it is based on MARTE Fundamental Concepts, so-called MARTE Domain Model.

At fundamental concepts level, we have UML metamodels respectively denoting SOPHIA Fundamental Concepts and MARTE Fundamental Concepts.

4.2 SOPHIA, a UML profile for safety

In this section, we describe the UML profile for SOPHIA. It consists of a set of UML extensions and libraries concretized through stereotypes and data types. They map to the SOPHIA Fundamental Concepts (see Figure 3 for a big picture). Similarly to the SOPHIA Fundamental Concepts packages, the corresponding UML profile the profile is designed following a modular approach by grouping language constructs into individual packages, with the ability to select only those packages that are of direct interest in a given model. Due to space limitations, it is not possible to provide details covering the all profile. Therefore, we will focus on the SOPHIA package ACCIDENTS described in Section 3.1.

In the package ACCIDENTS every fundamental concept will directly result in a UML stereotype with its corresponding properties. In this case, there is a 1:1 mapping between the Fundamental Concepts and the profile element. The bottom package in Figure 4 defines how the metaclasses of the UML metamodel are extended with SOPHIA concepts, while the top-hand package shows a subset of the SOPHIA library with some enumeration types of interest.

Before describing the details of how SOPHIA exploits `MARTE::VSL`, let us introduce a real industrial railway example of risk assessment.

Example Figure 5 shows a typical example of risk assessment tables used in the railway domain. Such tables are used to identify the tolerable accident rate (TAR) of a given accident case (stereotype AccidentCase in Figure 4) and the occurrence rates of the different consequences of an accident case (stereotype AccidentConsequences in Figure 4). Typically, these tables are standardized by the territory authorities. For the sake of simplicity, we focus on the calculation of the TAR and the consequence occurrence rate parameters for a given country.

Starting from "Table a:" of Figure 5, safety engineers define a severity level for every accident case. This information allows for identifying the threshold of the accident case risk. (annotated with "T" in the figure.) The threshold risk identifies the upper limit of a tolerable risk. For instance, let us consider a Critical severity level. The corresponding threshold risk can be identified in the fourth row of the Critical column. This corresponds to the Undesirable risk level. The obtained threshold risk level can then be used to identify a corresponding threshold frequency of the accident case. In our example, such frequency level is Remote. This yields an input value for "Table b:".

"Table b:" describes a mapping of frequencies of accident cases and numerical information about the magnitude order of such frequencies. This magnitude order is specified as an interval of real numbers. The lower bound value of this interval, corresponding to the threshold frequency level of a given accident case, represents the

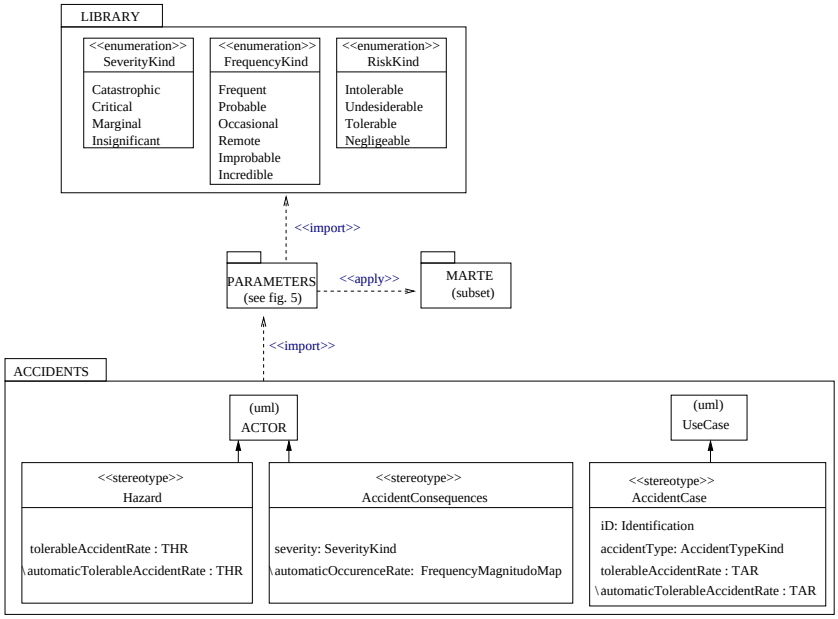

Fig. 4. SOPHIA: focus on TAR

TAR value for this accident case. In Figure 5, the TAR value the studied accident case is 1x10-8.

Figure 6 shows the model representation of "Table a:". In a:RiskTolerabilityAccident, each line of attribute RiskMapping represents a line of "Table a:". Consider "Table a:". We can read it as: we taken two values, one for colomn and one for row, then, we uniquely identify one cell, which contains the value of the risk. For example, for the colomn we select severity = $critical$ and, for the row, frequency = $Incredible$ Hence, we achieve a unique cell, which contains risk = $Negligeable$. The first line in Figure 6 represents the list of these attributes and their values. In instance a:RiskTolerabilityAccident, each line is given by the above procedure. In order to model the threshold between what is tolerable and what is undesiderable (noted by "T" in Figure 5), we introduce the Boolean attribute isThreshold in Figure 6. Therefore, if severity = $critical$, then risk = $Undesiderable$, because isThreshold = $True$.

a:RiskTolerabilityAccident is an instance of class RiskTolerabilityAccident, which contains one attribute riskMapping of type RiskMappingType. We stereotype RiskMappingType with VSL::TupleType. As might be expected, the VSL package (which contains a set of stereotypes extending the data type of UML) is applied to some of the SOPHIA data types. By definition, a TupleType is a data type that combines different types into a single aggregated type [25]. This allows instances of these tuple types to be annotated as composite values following the textual syntax defined for VSL tuple specifications.

Similarly to Table "a:", we represents "Table b:" by Figure 7, where some predefined MARTE data types are imported and reused in SOPHIA constructs. For instance, RealInterval (a MARTE's data type stereotyped VSL::IntervalType) is typing the magnitudoOrder property of FrequencyMapType. This allows instances of IntervalType to

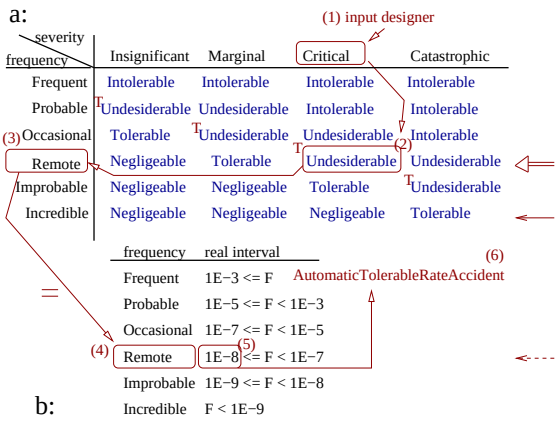

Fig. 5. Table Risk Tolerability and Table Frequency

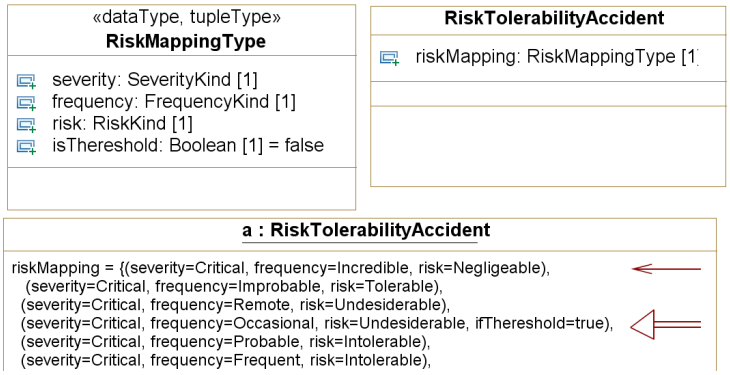

Fig. 6. package PARAMETERS: SOPHIA and MARTE for "Table a:"

be specified with the VSL syntax for interval values, as depicted in b:FrequencyMagnitudoMap.

Algorithm to calculate the TAR parameter: In the sequel, we describes the algorithm used to derivate the TAR parameter.

GLOBAL VAR RiskTolerableAccident : **ARRAY**[SEVERITY_KIND][FREQUENCY_KIND]: [risk:RISK_KIND,IsThereshold:BOOLEAN];
GLOBAL VAR FrequencyMagnitudeMap: ARRAY[FREQUENCY_KIND]: REAL_INTERVAL;
AutomaticTARCalculate (UserSeverityValue:SEVERITY_KIND): TAR;
VAR MyFrequency: FREQUENCY_KIND;
VAR MyInterval: REAL_INTERVAL;
VAR j: INTEGER;

j := 0;

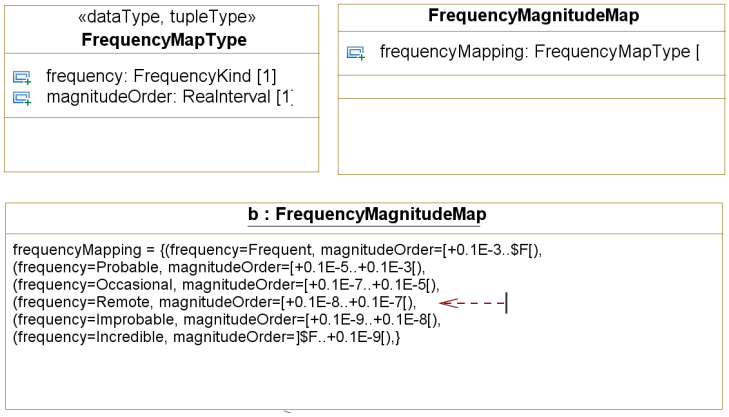

Fig. 7. package PARAMETERS: SOPHIA and MARTE for "Table b:"

```
WHILE (RiskTolerableAccident [UserSeverityValue][j].IsThereshold ⟨⟩ TRUE)
        DO j := j+1;
MyFrequency := FREQUENCY_KIND[j];
MyInterval := FrequencyMagnitudeMap[MyFrequency];

RETURN AutomaticTARCalculate := MyInterval.LowerBound;
```

Although it is written in pseudo-code, it can be implemented in Java code and easly introduced in a static profile implementation of SOPHIA.

5 Discussion

In Section 2, we have discussed some works on safety that have a great impact in the MDE community. Most of them provide both a way to specify the safety attributes in the design model, and tool support for safety analysis. Often, we have faced on two different models: one for safety and another one for design modelling. The focus is then on the "right mapping" and in the a-posteriori verification of the safety attributes.

SOPHIA is based on four capital pillars:

- SOPHIA Fundamental Concepts;
- reuse of pre-existing profiles (and then their tool support);
- automation on the propagation of the safety attributes in the design model;
- a-priori verification of the safety attributes in a correct way regarding to pre-defined risk tables.

In the sequel, we discuss each SOPHIA pillar with respect to some of the works presented in Section 2.

Altough SOPHIA is a UML profile, SOPHIA Fundamental Concepts have been created as free as possible from considerations related to specific solution technologies so as to not embody any premature decisions that may hamper later language use. This means that the fundamental concepts model can be concretized not only as a

UML profile, but also as an independent modelling language, possibly implemented as an Ecore metamodel or an XML schema, as well. Note that, although the SOPHIA Fundamental Concepts are specified in the form of a metamodel with a textual semantic description (like in MARTE), it represents only conceptualization entities synthesizing the "universe of discourse". This pillar is similar to that presented in work [15] in which the authors first define a safety conceptual model of safety-aware component-based architectures and just then define a safety UML profile.

The second pillar introduces SOPHIA as a UML profile, by adopting the definition of profile firstly given by S. Cook. As a result, SOPHIA profile strategically reuses some packages of MARTE and can be easily integrated in a SysML system architecture.

The third pillar put the strength in improving the automation of the modelling process. In particular, SOPHIA provides a framework to automatically generate the value of TAR and the frequency of an accident, from the specification of only one attribute by users, which is the severity attribute of a consequence. This attribute is given by engineers by choosing one of four possible values.

Finally, the fourth pillar's objective is to enable safety ensurance calculation along the development process, in a way that is correct with respect to pre-defined tables.

6 Conclusions

In this paper, we present for the fist time SOPHIA, a *model-based safety engineering* approach. SOPHIA responds to industrial needs regarding the integration of safety engineering and system design. SOPHIA provides a metamodeling and profiling infrastructure to specify and propagate the safety information on design models. We have particularly focused on the TAR calculation, which is the first step of the risk evaluation of an accident. The result of some safety attributes, such as TAR, influences the SIL and, hence, changes the model architecture. Such safety information is *a-priori correct* regarding to pre-defined risk tables. Currently we are performing tests on industrial real cases. We are applying the same process (as discussed for TAR) to other safety attributes. We also intend to mathematically formalize correctness of the automatic propagation of the safety attributes in the design model.

Acknowledgment

This work has been performed in the context of the IMOFIS project of the System@tic Paris Région Cluster. It is sponsored by the "Safe, reliable and adapted transportation" program (PREDIT) of the "Agence Nationale pour la Recherche". The authors would like to thank the all member of the IMOFIS project [20] and the reviewers of ACES^{MB} Workshop for their valuable suggestions.

References

1. AADL. Architecture Analysis & Design Language. `www.aadl.info/aadl/currentsite/index.html`.
2. J. R. Abrial. *The B-book: assigning programs to meanings.* Cambridge University Press, 1996.

3. Alstom. Guidance for safety analysis. MODTRAIN, MODCONTROL Sub-Project, 2008.
4. C. André. Time Modeling in MARTE. In *FDL'07 Forum on specification and Design Languages*, Barcelona, Spain, 2007.
5. ATESST Project. Advancing Traffic Efficiency and Safety through Software Technology. ATESST STREP - FP6 project. `http://www.atesst.org`.
6. AUT@SAR. Automotive Open System Architecture. `www.autosar.org`.
7. F. Belmonte. T1.1 guide de modélisation. Projet IMOFIS, Alstom Transport, System@tic, 2009.
8. A. Blas and J. L. Boulanger. Comment améliorer les méthodes d'analyse de risques et l'allocation des THR, SIL et autres objectifs de sécurité. In *Lambda-Mu, 16e Congrès de Maîtrise des Risques et de Sûreté de Fonctionnement*, Avignon, France, 2008.
9. S. Bliudze and J. Sifakis. The Algebra of Connectors - Structuring Interaction in BIP. In *Int. Conf. EMSOFT*, pages 11–20, 2007.
10. B.P. Douglass. Build Safety-Critical Designs with UML-based Fault Tree Analysis-Defining a Profile. `www.embedded.com/design/opensource/217200312?pgno=1`.
11. CENELEC. EN-50126: Application ferroviaires -Spécification et démonstration de Fiabilité, Disponibilité, Maintenabilité et Sécurité (FMDS). Norme, CENELEC, 1999.
12. CENELEC. EN-50128: Applications ferroviaires - Système de signalisation, de télécommunication et de traitement - Logiciels pour systèmes de commande et de protection ferroviaire. Norme, CENELEC, 2001.
13. CENELEC. EN-50129: Application ferroviaires - système de signalisation, de télécommunication et de traitement - systèmes électroniques relatifs à la sécurité pour la signalisation. Norme, CENELEC, 2001.
14. L. de Alfaro and T. A. Henzinger. Interface automata. In *Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering*, pages 109–120. ACM Press, 2001.
15. M. de Miguel, J. Briones, J. Silva, and A. Alonso. Integration of satety analysis in model-driven software development. 2008.
16. H. Espinoza, B. Selic, D. Cancila, and S. Gérard. Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems. In *In Proc. of Int. Conf. on Model Driven-Architecture Foundations and Applications (ECMDA 09)*, volume 5562. LNCS, 2009.
17. P. Feiler and A. Rugina. Dependability Modeling with the Architecture Analysis & Design Language (AADL). Technical report, Software Engineering Institute, Carnegie Mellon, 2007.
18. B. Hamid, A. Radermacher, A. Lanusse, C. Jouvray, S. Gerard, and F. Terrier. Designing fault-tolerant component based applications with a model driven approach. In *IFIP Workshop on Software Technologies for Future Embedded and Ubiquitos Systems (SEUS 2008)*, Springer LNCS.
19. IEC. *61508:1998 and 2000, part 1 to 7. Functional Safety of Electrical, Electronic and Programmable Electronic Systems.*, 2000.
20. IMOFIS Project. Ingénierie des MOdèle de FonctIons Sécuritaires. `www.imofis.org/`.
21. E. Jouenne and V. Normand. Tailoring IEEE 1471 for MDE Support. In *UML Modeling Languages and Applications*, LNCS, Springer, 2005.
22. N. Limnios. *Fault trees.* ISTE, 2007.
23. OMG. http://www.omg.org/.

24. OMG. Systems Modeling Language SysML. www.sysml.org.
25. OMG. UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 3. www.omgmarte.org.
26. OMG. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics & Mechanisms (QoS & FT profile). www.omg.org.
27. OMG. Unified Modeling Language (UML) Specification: Infrastructure. Version 2.0. www.uml.org, 2004.
28. OMG. UML Profile for Schedulability, Performance, and Time Specification. www.uml.org, 2005.
29. F. Ougier and F. Terrier. ADONA: an open Integration Platform for Automative Systems Development Tools. In *Euopean Congress Embedded real Time Software (ERTS)*, 2008.
30. D. Schmidt. Model-driven engineering. *IEEE Computer*, pages 25–31, February 2006.
31. B. Selic. From Model-Driven Development to Model-Driven Engineering. Keynote talk at ECRTS'07. http://feanor.sssup.it/ecrts07/keynotes/k1-selic.pdf.
32. F. Terrier and S. Gerard. MDE benefits for distributed, real-time and embedded systems. In *From Model-Driven Design to Resource Management for Distributed Embedded Systems, IFIP TC 10 Working Conference on Distributed and Parallel Embedded Systems (DIPES 2006)*, 2006.
33. Veryard Projects. Component-based Development FAQ. http://www.users.globalnet.co.uk/~rxv/CBDmain/cbdfaq.htm, February 2008.