A UML Profile for the Development of IEC 61508 Compliant Embedded Software

Dirk Kuschnerus†, Felix Bruns†, Attila Bilgic∗ and Thomas Musch†
†Ruhr-Universität Bochum
{Dirk.Kuschnerus, Thomas.Musch}@est.rub.de, Felix.Bruns@rub.de
∗KROHNE Messtechnik GmbH
A.Bilgic@Krohne.com

Abstract—In this paper we propose a UML profile that extends the Unified Modeling Language (UML) to support the development of safety-critical embedded software in accordance with the safety standard IEC 61508 [5]. Our profile enables software developers to precisely express certification-related information using the UML notation. This improved information density in software models can be exploited as foundation for activities in various software development phases, for example the reuse of certified software components or the deployment of safety-critical and non-critical software components to separated nodes.

Index Terms—Safety-critical, embedded, software development, UML, profile, IEC 61508

I. INTRODUCTION

Embedded software controls and supervises processes and applications in various domains, e.g. process industry, automotive and transportation. In many of the controlled processes the failure of the system under control or only parts of the system can lead to serious threats to both environment and humans. Consequently, these domains have a high demand for means that reduce the existing risks to a tolerable level. These safety functions [5] ensure the functional safety of the process and are often implemented by safety-critical embedded software.

To guide developers, certification authorities and customers in the field of functional safety, different general and domain specific standards exist. Our approach focuses on the standard IEC 61508 [5], which is a generic standard widely adopted in industrial domains [2]. IEC 61508 covers the entire software development process and recommends techniques that ensure software quality and functional safety.

The increased application of embedded software for safety-related tasks and the customers’ demand for a broader portfolio of measurement devices regarding both features and applications lead to a growth in software complexity. Additionally, the industry is facing the need to develop embedded software of high quality in a minimized amount of time. A natural solution to this situation is to enhance software processes towards more efficiency in software reuse. Popular approaches like component based development [8] and software product lines [4] do this by deriving a software platform that integrates exchangeable software modules.

The UML is widely accepted among industry and research as de-facto standard in modelling of object oriented software. It is also a key technology in the active research field of model driven development (MDD) [1],[18], that focuses on models as central elements in software development. It supports approaches like [4] and [8] by offering the possibility to model software components and their deployment on nodes and is mentioned as recommended semi-formal method for software architecture design and detailed design in IEC 61508.

However, to employ such processes in the domains of safety-critical and embedded systems, safety aspects have to be addressable in the software models. In this context the applicability of UML suffers from drawbacks like missing modelling support for certification information, safety requirements and safe communication concepts.

To overcome these drawbacks, we developed a domain specific modelling language (DSML), a concept closely related to MDD, that is capable of precisely specifying all aspects of a specific domain. A DSML enables the expression of domain concepts in combination with the benefits from modelling like formal specification and code generation. DSLMs can either be developed as stand-alone or on top of a generic language. In our approach, we utilize UML’s integrated profile mechanism that allows us to enrich the standard UML with the concepts of IEC 61508. This ensures that our approach will be easily adoptable to legacy tools and processes that already use UML.

Training overhead will be reduced for a big community of developers that are already familiar with UML due to it’s wide acceptance as modelling technology.

It is a commonly accepted practice that the process of developing a DSML includes the creation of a domain model that specifies the relevant entities and their relations from the domain that the DSML is targeted at. The advantage over a direct implementation of the DSML is that this domain model is less likely to include aspects that have their source not in the targeted domain but in the implementation mechanism of the DSML [6],[10],[19]. We adopted this practice by extracting the relevant concepts from IEC 61508 into a domain model and used this domain model as input for the creation of our UML profile.

The UML profile developed in this paper is designed to provide a foundation for further modelling techniques. It aims to offer significant improvements for the development of safety-critical embedded software over the use of plain UML and other approaches reviewed in Section II. The ability to integrate precisely modelled IEC 61508 concepts into software development models raises the information density of development models and lowers redundancies in separate
II. RELATED WORK

The model driven development of safety-critical software is an area of active research. Juerjens [9] introduces a UML profile that supports a collection of safety aspects that can be used for software safety analysis but does not target IEC 61508. The authors of [7] map safety patterns for IEC 61508 to UML models but do not apply UML profiles and focus on communication patterns. In [21] a UML profile targeted at the ‘Airworthiness’ standard RTCA DO-178B [17] is developed. The approach however is limited to UML class diagrams and does not introduce constraints to it’s models. Lu and Halang [11] introduce a UML profile that provides modelling for the concepts supported by the programming language ‘PEARL’, which is targeted at IEC 61508. [16] presents the design process used in the development of a safety-critical control software module in the railway domain. The process sets a high value on analysis of the model based software specification and uses UML stereotypes in this context. This includes modelling of faulty system states and error correcting transitions. [3] introduces a UML profile used as domain specific language for the railway domain. The profile is focused on the modelling of track networks and does not feature safety aspects in depth. The approaches in [9], [11], [16] and [21] do not support essential aspects like recommendations, certification data and modification of software from IEC 61508. Furthermore, the authors of [11] and [21] decided to directly build a profile in UML instead of following the widely recommended ([6],[10],[19]) path of defining an implementation independent domain model first.

Concluding our literature review, to the best of our knowledge there is no UML profile approach that comprehensively covers the safety concepts from IEC 61508.

III. DOMAIN MODEL

The foundation for our UML profile was to define a domain model as recommended e.g. in [6], [10] and [19]. A domain model contains core elements of the domain under investigation and the relationships among those elements. Additionally, constraints regarding relations and elements can be included into the model. The main benefit of this approach is that it separates the process of gathering information from a domain and the process of building a UML profile from this information. This leads to a higher quality of the profile because the original domain model is not biased with information specific to the implementation of the DSML [19].

We defined our domain model using the OMG Meta Object Facility (MOF) [12] and described the extracted domain concepts from IEC 61508 as classes and the relationships between those concepts as associations. Constraints between concepts are needed to circumvent ambiguities in the domain model [20] and were formally defined using the OMG Object Constraint Language (OCL) [14]. We divided the domain model into two parts taking into account the different kinds of information in IEC 61508. The first part of our model is closely related to the safety process defined in IEC 61508 and incorporates activities and recommended techniques. This includes e.g. monitoring concepts in the process phase of software architecture design. The second part covers the standard specific concepts such as definitions of safety terms and their relations. Examples for the second part are safety functions and the certification status of software modules. The iterative process of model specification is shown in Figure 1. The domain information from the standard was extracted and translated into model elements. The new model elements were then connected to existent elements by UML associations precisely defined by OCL constraints. Before the extraction of the following domain concept, the model consistency, e.g. duplicate elements or conflicting constraints was checked and recovered if necessary. The resulting models were evaluated with a domain expert and the required improvements were addressed in additional process iterations.

Figure 2 shows the domain model for communication as an example for standard specific concepts. In safety-critical systems, the two relevant kinds of communication are safe communication, where the transmission of data is secured by safety measures (e.g. a checksum) and untrusted communication where this is not the case. Communication is established between multi-layered communication stacks using a communication channel as medium and following a specified protocol. There are two concepts of communication channels described in IEC 61508. A white channel, which
Fig. 2. Domain Model for Communication

offers measures that ensure a safe communication, and a black channel, that does not provide such measures and is therefore to be treated as untrusted. In case of a black channel, the communication stacks must provide a safety layer that uses the black channel for transmission and protects the safety-critical data in a way that provides safe communication to the layers above. Provision and usage of communication stack functions is managed through an interface that ensures encapsulation and prevents applications from bypassing the safety measures provided by the communication stack. Figure 2 also shows the necessity to narrow the possible interpretations of the model with constraints. A safety layer must be applied when a black channel is used. This invariant is defined by the OCL constraint 'safetyLayerIfBlackChannel':

\[
\text{Context CommunicationChannel}
\text{inv : self.channelKind = blackChannel implies}
\text{self.communications →}
\text{forall(c : Communication|c.sourceStack.layers →}
\text{exists(l : CommunicationLayer|}
\text{l.isStereotyped("SafetyLayer"))})
\]

The overview of SIL activities and techniques in Figure 3 is the top level diagram for the second part of the domain model. It shows the relations between an Electrical/Electronic/Programmable Electronic (E/E/PE) safety-related system, its required safety integrity and the techniques that are used to realize the system. The SIL activity is a part of the safety process defined by IEC 61508. For each activity, the standard recommends one or several alternative techniques for each safety integrity level. Again, our domain model defines constraints to represent the guidelines of the standard. The constraints in the E/E/PE safety related system ensure that the developer has to justify deviations from the recommendations with rationales, e.g. when a highly recommended technique is not used in the development of the safety-related system.

IV. UML PROFILE

The domain model described in section III served as input for the derivation of our UML profile. The profile was created by mapping every top level element of the domain model to the basic UML construct that complied the most with the domain element [19]. The domain element was then derived from this UML metaclass as a stereotype. Where possible, lower level domain concepts were specialized from the previously derived stereotypes. We decided to reuse concepts from the standardized UML profiles MARTE [13] and SysML [15] whenever applicable. These profiles have the advantage that they are relatively well-known among software developers familiar with UML and are supported by a variety of modelling tools. For example, we used the SysML requirement stereotype as a baseline for modelling safety requirements.

To make our profile more versatile, we defined multiple metaclasses for stereotypes where modelling of a concept from different points of view was desirable. Figure 4 is an extract from the profile definition of the communication domain model described in Figure 2. It shows that the stereotype CommunicationChannel is derived from an Association as well as from a CommunicationPath. This enables the user of the profile to model a communication channel from a deployment (CommunicationPath) as well as from a component point of view (Association). The component point of view can be used to specify virtual communication channels used by software modules, e.g. shared memory, that would not appear on a deployment diagram.

IEC 61508 [5] requires that requirements concerning the safety of a device are defined prior to the development of hardware and software. UML’s use case concept is often used to specify requirements and their relations. However,
an explicit way to model software requirements does not exist by default. To address this issue and to bridge the gap between requirements specifications and software models, the UML profile SysML introduces a requirement diagram that is able to model requirements and their relations to other requirements and model elements. Our approach expands this ability of SysML by adding a new stereotype for safety requirements. This integration of requirements to software models is especially useful for maintaining requirements traceability, i.e. answering the questions by which software modules a requirement is realized and vice versa. It is thus possible to trace between software elements and safety requirements by using our profile.

The information weather and how a software part has already been certified is highly relevant during multiple software development phases. Models augmented by certification information can serve for example as foundation for deployment and recertification decisions. Figure 5 shows the realization of this concept in our UML profile. A certification is described by its date, the software version that has been certified, the approach that has been followed during to reach the certification and the targeted safety integrity level. A software part that has been certified is stereotyped as ≪TrustedSWModule≫. The relationship between ≪TrustedSWModule≫ and ≪Certification≫ is defined in the OCL Constraint 'TrustedSWModuleCertification' as follows:

```
Context Class
inv : self.isStereotyped("TrustedSWModule")
implies
exists(ce : Certification|ce.certifiedSW = self)
```

The certification is modelled as discrete model element and bound to the certified software part by an association. Our profile allows the developer to classify both components and classes as certified software parts, supporting both an abstract and a refined view on the software system.

V. EXAMPLE

This section gives an example for the employment of our profile in the development of a software component for an industrial measurement device. The developed component is located at the safety-critical output of the exemplary device and sets an analogue output current according to digital measurement values received through a device-internal bus system. A measurement device can have several different output options but in our example, the current output is the only one that sends values which are considered as reliable by the control room. This typical setup for industrial devices implies that the software involved in the measurement, processing and communication of data for the current output is considered safety-critical.

Our profile can be used as basic technique throughout several development phases of this safety-critical software. One of the first steps in most software development processes is the specification of requirements. We use traceability in combination with our ≪safety requirement≫ stereotype to track two basic safety requirements in our example. SR1 specifies that the timeliness of measurement values that the reliable output receives has to be ensured. The detailed description of SR1 considers to be fulfilled when a measurement value arrives at the current output every 50ms. SR2 states that the analogue signal from the current output has to be correct regarding the measurement values. Both requirements can be traced in various development phases. Figure 6 uses a high level of abstraction showing that both requirements are realized by the 'Safe Output Component'. This information is detailed in figure 7 where trace information allows the localization of the submodules where each requirement is realized.

After the specification of requirements, the system under construction is described on a coarse level of abstraction. In this phase, often referred to as architectural design, UML component models can be used to display a highly abstract view of the software parts and their interconnections. Figure 6 shows such a component based description of the measurement device introduced before. We can extract from the diagram that measurement values are sent from a 'Measurement' component to a component that performs calculations like filtering on the measurement data. The example device has two output options, one fieldbus and the reliable current output realized in the 'Safe Output Component'.

Using our profile, it is possible to distinguish between safety-critical and non-critical components and use the traceability
information to determine which component is involved in the fulfillment of specific safety requirements. Using this information, we see that there is a chain of safety-critical data and thus safety-critical software components from the measurement to the processing and the safe output component. From the expanded component diagram we also learn that the fieldbus component is not safety-critical, i.e. does not provide reliable output regarding functional safety. As mentioned above, the SysML requirements integrated into the component diagram allow us to express on a high level of abstraction that the safe output component realizes both safety requirements SR1 and SR2.

The software description, in our case the models specifying our device software, are continuously refined in the course of a software development process. After the definition of the software architecture in terms of connected components, our example process continues by detailing single components. Figure 7 shows an example for such a refined model which describes the safety-critical 'Safe Output Component' in more detail. From the model in figure 7 we can see that the output interface 'ISafeOutput' is delegated to a submodule that calculates the output current from the measurement values received over a communication stack. The model serves as a detailed description regarding communication architecture and certification for the safe output component. Concerning communication architecture, it states that submodules can access the communication stack for device internal communication via a safety-critical interface and an untrusted interface. The safety-critical interface 'ISafeCommunication', that is used by the submodule 'Output Calculation', delegates to a safety layer which is responsible for safe communication over the untrusted lower layers of the transmission system. The interface for non-trusted communication delegates directly to the lower layers and therefore does not offer the ability to communicate in a safe way.

Besides this communication aspects, the diagram in figure 7 shows how our profile enables the modelling and binding of certification information within software models. Following the links to the 'Certification' model elements, we can see that both the subcomponent that calculates the output current and the safety layer that enables safe communication have already been certified. As SysML requirements do for requirement specifications, the certification elements in our profile act as a bridging element to certification documents. The certification elements in the example model provide the information that the output calculation was certified in a proven-in-use certification reaching safety integrity level 2 while the safety layer was developed applying techniques recommended by IEC 61508 [5] for safety integrity level 3. This information is valuable for developers, in recertification questions and following development steps like impact analysis. As mentioned above, the requirements traceability has been detailed along with the model refinement in figure 7. The timeliness requirement SR1 for example was formerly traced to the 'Safe Output Component' as a whole. This trace link was actualized so that we can see that the safety requirement is realized in the safety layer of the communication stack.

The software components of a complex system can be developed individually by different groups of people. The process of integrating these components into a final software system includes the activity of deploying software components to hardware nodes. Figure 8 shows how our profile can assist the certification of embedded systems from the deployment point of view. The figure introduces the simplified setup of our exemplary measurement device with a sensor measuring physical values from a medium and a microcontroller receiving this values over a bus connection and performing the tasks of processing the received data and generating analogue output. The example assumes that the 'Measurement' component has been deployed to the sensor hardware and the safety-critical output module to the I/O microcontroller. It concentrates on the communication between the two modules deployed on different hardware nodes.

The communication is performed in a typical configuration for
VI. Conclusion and Future Work

A. Conclusion

In this paper, we give an overview of a UML profile for developing safety-critical embedded systems compliant to the standard IEC 61508. The profile aims to improve the productivity of model based software development regarding to the achievement of standard compliance, reuse of software artefacts and documentation of certification information. We believe that with our profile, the certification efforts especially for multiple certification processes can be significantly reduced.

B. Future work

The UML profile introduced in this paper can serve as a foundation for further research in the area of safety-critical embedded software. An important step is to implement tool support for our profile that aids developers in creating certified software. Tooling should gain awareness of IEC 61508 recommendations from the profile’s OCL constraints to actively aid developers in specifying software architectures and using techniques compliant to IEC 61508. Another important factor in embedded systems are constraints on available energy. An approach in this area will extend our profile to model energy constraints and to provide an algorithm that aids developers to construct a system optimized in terms of energy consumption.

REFERENCES


Fig. 8. Example: Deployment Model with Communication