

COVER STORY:

## Integrating safety in model-driven development with SysML

SPECIAL FEATURES:

- Safety & Security
- Software Development
- MCUs & DSPs

# Integrating safety in model-driven development with SysML

By Matthew Hause and Francis Thom, Artisan Software Tools

*Systems modelling language, SysML, enables the construction of a system with techniques for identifying, managing and ultimately mitigating risks, resulting in a safety case that the system is acceptably safe to operate in a particular context.*



■ Where systems are required to meet specified industry safety standards, safety concerns need to be intrinsic to the system architecture, so that safety is inherent in all areas. Systems modelling language, SysML, can be used to construct a single coherent model of a system allowing the various different disciplines (e.g. safety, systems, and hardware and software engineers) to work both in isolation and as a team. Using an integrated database and ergonomic profiling to support all the disciplines, it is possible to create bespoke views of the model based on whatever notation is preferred by a single discipline, and to enforce the rules that cut across to other disciplines.

In order to evaluate a system, it must be broken up into its various concerns, where a concern is any focus of interest in a system. The process is known as the separation of concerns. Ideally, most components in a system will perform a single, specific function. However, they will often share common, secondary requirements with other system elements that cross-cut into the primary requirements. Cross-cutting concerns usually include traceability, timeliness, risk and safety. For example a system may change from safety integrity level (SIL) 2 to SIL 3, due to configuration changes in other areas. Consequently, the means of addressing these concerns not only needs to be flexible but must be identified early on in the requirements

phase of the project, both to assess impact and provide traceability. The unified modelling language (UML) was created to provide a means for object-oriented (OO) software engineers to model software systems. The “software only” emphasis of UML was changed in UML 2.0, where the scope of the language was widened to include systems in general. The systems modelling language (SysML) further expanded on this by defining a set of extensions necessary for systems engineers to evaluate systems. These extensions include requirements modelling, system structure, parametric modelling, allocation, extensions to activity modelling to include probability and continuous systems.

Both UML and SysML employ a variety of organising principles - hierarchies such as inheritance and aggregation, networks such as the associations and connections between system elements, and dependency relationships that can exist between and within system elements and packages. UML also provides viewpoints for expressing these paradigms such as browsers and static structure diagrams. However, cross-cutting concerns are difficult to express using traditional paradigms, and also need to be addressed early on in the project as they can have far reaching consequences for both product and process. Consequently, it is essential that modelling environments allow system modellers to

document these concerns and how they will be addressed and resolved. Figure 2 shows how cross-cutting concerns relate to the model. The model is made up of requirements, application and hardware hierarchies which are shown as the light blue boxes, red boxes, and dark blue circles respectively. In addition, there are concerns that cut across the three hierarchies such as safety and risk. These are shown as the green and yellow circles.

Traditionally, engineers have addressed separation of concerns by creating different views and viewpoints of a system. For UML, a view is a collection of models or model information that represents one aspect of an entire system. A view applies to only one system, not to generalizations across many systems. A “viewpoint” is a specification of the conventions for constructing and using a view - a pattern or template from which to develop individual views. For a SysML model, a viewpoint could be considered as a set of diagrams or model projections that allow the developer to visualize, create reports, verify, and create consistency checks on the system relating to the concern.

However, modelling systems is more than just drawing pictures. It is also necessary to provide additional rules governing the creation of relationships between elements, interactions, multiplicity, cross diagram relationships, and the

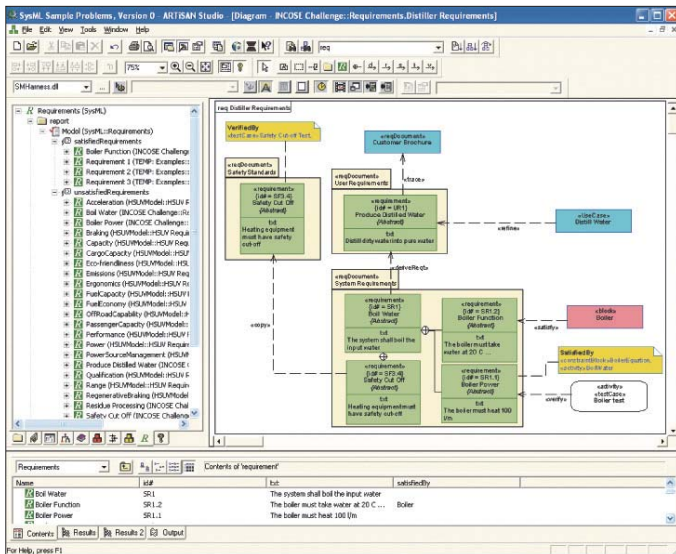


Figure 1: Example of a safety requirements diagram

consequence of deleting elements that have relationships to others. This process is called ergonomic profiling (EP). In order for EP to work effectively, the diagram editor must be built using an integrated meta-model. Simply put - if class A is defined as a parent of class B in one diagram, it cannot be defined as a child of class B on another diagram. Whilst this simple example illustrates the point, the rules and relationships necessary for a correct model can be far more onerous. Accordingly, consistency on a single diagram is not sufficient. To illustrate different

concerns and demonstrate how EP can be used to provide a visual representation of the concern, and enforce relationships and rules, we have developed a series of example case studies.

Robustness of a system usually refers to its tolerance of invalid inputs. However, in this case it is taken to evaluate the consistency of the object usage and whether or not the individual objects, and the system as a whole, are fit for purpose. A robustness diagram provides a method for analyzing the steps of a use case to validate the business logic within it, and to ensure that the terminology is consistent with other use cases previously analyzed. The diagram allows the modeller to focus on the relationships and responsibilities of the objects under consideration, and to determine the efficacy of the cross-cutting robustness concern (figure 3).

It is also important that correct object type relationships are enforced. Once the objects have been analyzed, classes can be created for those objects, reflecting their primary concerns as well as the robustness concern. Additionally, rules can be set up to ensure that class associations do not violate rules defined for their object instances. To implement this view, a robustness analysis profile was created containing the boundary, control, entity and robustness diagram stereotypes, with corresponding toolbar buttons.

One of the goals of SysML is the ability to integrate requirements into a UML model. A requirements diagram can be created to support the underlying requirements model. A requirement represents the behaviour, structure, and/or properties that a system, component, or other model element must satisfy. Requirements specifications can be modelled

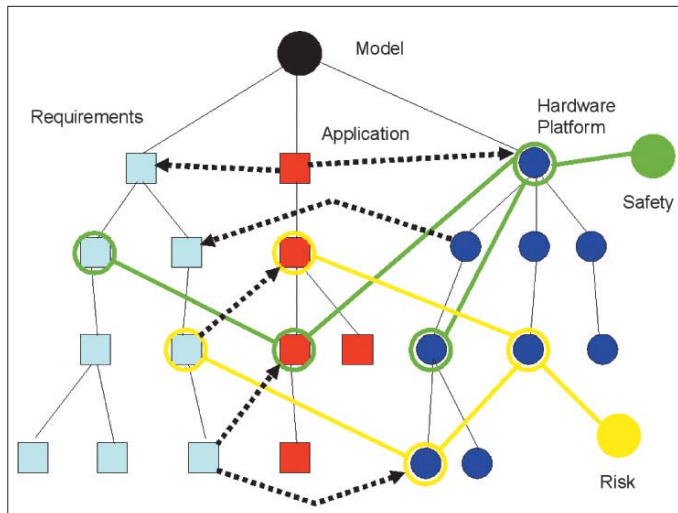


Figure 2: Conceptual system architecture

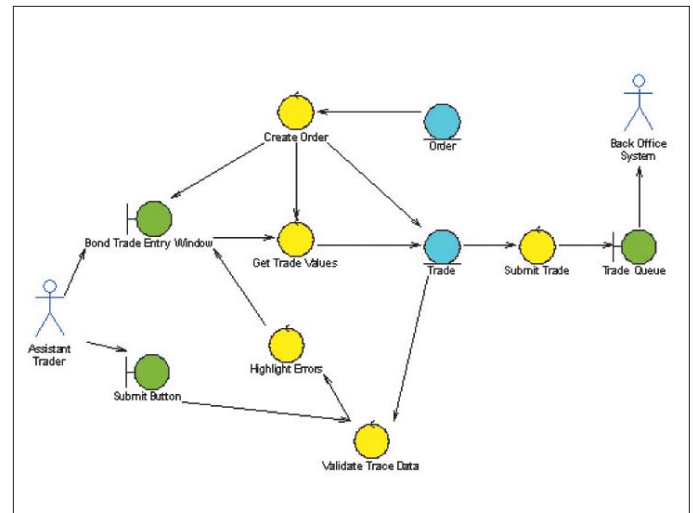


Figure 3: Robustness diagram

containing sets of requirements. The requirements model includes relationships among requirements and between requirements and other model elements.

Specialized requirements, for example safety requirements, can be specifically identified and specified using stereotypes. In this example (figure 1) the safety standards, user requirements and system requirements are packages with the reqDocument stereotype grouping sets of requirements. The requirement attributes are also modelled. The user requirements trace to the customer brochure, the distil water use case refines the produce distilled water requirement, and the boiler satisfies the boiler function requirement. The safety cut-off requirement in the system requirements package is a copy of the requirements in the safety standards package. This requirement is verified by a safety cut-off test. This simple example illustrates how traceability to a safety requirement can be shown. Problems can also be identified and flagged. This integration of requirements into the model provides a means by which requirements can cross cut any of the model hierarchies.

The safety integrity level (SIL) concept was created in order to quantify the safety requirements of specific components in a safety related system. SIL is the likelihood of meeting required safety features. One of the problems with SIL is traceability to the component and connecting parts. In other words, if component A is required to comply with SIL 2, all its constituent components must also comply with SIL 2, and in some cases, interconnected elements are affected as well. In this example, to solve the problem of identifying these elements, a SIL stereotype was created with tag values for the different SILs. The stereotypes were then applied to the different components and their contained parts. The hardware architecture

was modelled using the SysML assembly diagram, and additional stereotypes applied to differentiate between boards, busses, disks, etc. The software classes modelled for the hardware also had these stereotypes applied. The SysML notation for deployment of the software system to the hardware is shown in figure 4.

In this example safety-related items such as engine management and traction control are deployed on a different processor from the infotainment and air conditioning. To verify compliance, a consistency checker was created to traverse the different hardware and software relationships ensuring that all contained and deployed elements were of the component SIL or higher. Elements that were not correct were flagged in the model and the appropriate action could then be taken.

A safety case should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context. An argument is used to demonstrate how someone can reasonably conclude that a system is acceptably safe from the evidence available within this context. The safety case consists of three principal elements: requirements, argument and evidence.

The purpose of the safety argument is to communicate the relationship between the evidence and objectives. Where engineers are concerned, it is common to find that textual descriptions that include complex logical reasoning will be hard to follow. Consequently, the text and, therefore, the meaning and considerations of the written safety argument, can be ambiguous and unclear, making it difficult, if not impossible, to ensure that all stakeholders involved share the same understanding of the argument. If there is no agreement on the meaning of the safety arguments, it will be impossible to agree on the safety case.

For this reason the goal structuring notation (GSN) was created. The GSN is a graphical argumentation notation that represents the elements of the safety argument (goals, strategies, solutions, context and the relationships that exist between these elements. For example, they can explain how an individual requirement represented by a goal is supported by strategies, how strategies are supported by solutions, and the context that is defined for the argument. When these elements are linked together they create a network called a goal structure.

Because many of the elements of the safety case and safety argument are already contained within the model, it is useful to integrate the GSN and safety argument into a UML/SysML model. For example, previously it was demonstrated how SysML provides the ability to model system requirements, and how these requirements could be expressed as use cases, classes, and relationships. In the same way, requirements, particularly safety requirements, and other system elements represent goals in the safety case. Derived requirements can map to derived goals, and system elements, system tests (from the SysML requirements model) as solutions. There is further benefit as impact analysis and traceability from requirements to implementation will also be able to take the safety case into consideration.

Not all concerns in a system can be easily modularized, nor is it often desirable to do so. It is therefore necessary to be able to identify those system elements affected by, or responsible for, implementing a concern in such a way that it does not prevent the satisfaction of more primary concerns. Additionally, it must also be possible to easily identify those elements affected by the concern to determine impact analysis, traceability, and other reports. By integrating these concerns into the model, it is

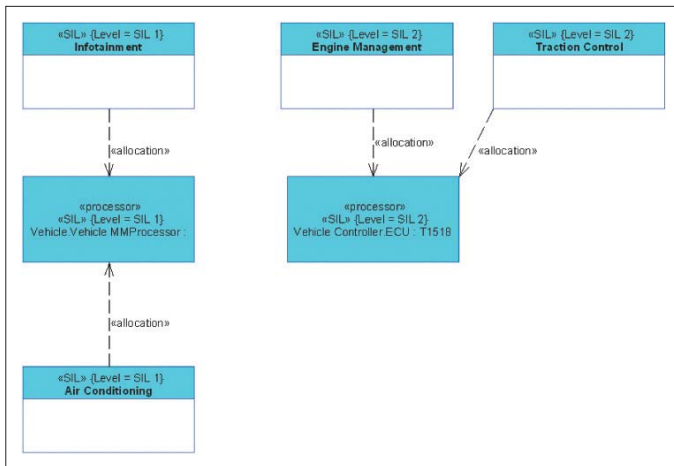


Figure 4: Internal block diagram showing allocation

possible to make use of existing system elements to support other concerns, as in the safety case example where requirements and test cases defined in the requirements model were used to support the safety argument. This increases traceability and provides a means to obtain a more complete picture of the system in terms of impact analysis. The normal procedures are to create separate model elements and often models, and then attempt to integrate them afterwards. Using the MDD philosophy, a single model is created with cross-cutting concerns thereby eliminating duplication and corresponding errors. EP provides a means of examining the model from the different viewpoints, as well as in-built consistency checks and reports. ■