An Integrated Safety Strategy to Model Driven Development with SysML

M.C. Hause, F. Thom

Artisan Software Tools Ltd. Eagle Tower Suite 701, Montpellier Drive, Cheltenham, Glos, UK GL50 1TA
FAX: +44 1242 229 301
Matthew.Hause@Artisansw.com, Fran.Thom@Artisansw.com

Keywords: SysML, Safety, Modelling, Profile, Aspects

Abstract

A building architect would not design a building without due consideration of existing safety standards related to the utilities (gas, electricity and water). These aspects of the building's design cut across the structural aspects of the building. The design of the building (commonly represented as a blueprint) contains aspects of structure, plumbing, wiring and implicit usage (layout of rooms, doors, stairs and windows etc.) within a single model using different notations to separate the different aspects. Similarly, no systems engineer would build a system and then add safety. Additionally, when using separation of concerns of decomposition to analyse the system, safety is not limited to a single or even many areas of the system. It needs to be inherent in all areas of the system. This paper will demonstrate how the UML and SysML can be used to construct a single coherent model of a system allowing the many different disciplines (e.g. safety engineers, systems engineers, hardware engineers, software engineers) to work in isolation whilst working together. 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. This paper will focus on the safety aspects of a SysML design and techniques for identifying risks, how these are managed and ultimately mitigated resulting in a safety case for the system under construction. It will include several illustrations of the techniques described above.

1 Introduction.

In order to evaluate a system, it must be broken up into its various concerns. This is the case regardless of whether one is performing requirements specification, analysis, design, or evaluating the entire system lifecycle. A concern is any focus of interest in a system. Separation of concerns (SoC) is the process of breaking a system into distinct features that overlap in their purpose as little as possible. The definition of concern used here is much broader than how it is used in traditional requirements management. Sommerville [16] views concerns as non-functional requirements or constraints. The term Separation of Concerns (SoC) was probably coined by Edsger W. Dijkstra in his paper "On the role of scientific thought" [3]. SoC was the basis behind modular system design, and helped to inform Object Oriented (OO) design and the UML. However, not all concerns can be addressed by modularization or OO. Ideally, most components in a system will perform a single, specific function. Additionally, they often share common, secondary requirements with other system elements. These secondary requirements are said to cross-cut into the primary requirements. These are known as cross-cutting concerns.

1.1 Cross Cutting Concerns

Examples of cross-cutting concerns include safety, traceability, timeliness, and risk. Some concerns may need to be addressed in all or disparate modules throughout the system. Modules created according to one implementation of SoC may also need to address additional concerns. In addition, the nature or details of these concerns may change during the evolution of the system. For example, security needs to be addressed for a communications system for the data, and communications links. It is necessary to assign a security level such as secret or top secret to the data as well as to the communications link. It is also necessary to ensure that any data exchange uses a transport link that supports a security level at least as high as the data to be exchanged. As the data exchange may in fact use several communications links to get from sender to receiver, this can become quite complex. Another example is where a part of 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 needs to be flexible, and needs to be identified early on in the requirements phase of the project to assess impact, and provide traceability.

1.2 UML and SysML.

The Unified Modelling Language (UML) was created to provide a means for Object-Oriented (OO) software engineers to model software systems. A glance at the introductory paragraph in version 1.1 of the specification clearly spells out its purpose. "The Unified Modelling Language (UML) is a general-purpose visual modelling language that is designed to specify, visualize, construct and document the artefacts of a software system. The UML is simple and powerful. The language is based on a small number of core concepts that most object-oriented developers can easily learn and apply. The core concepts can be combined and extended so that expert object modellers can define large and complex systems across a wide range of domains." [12]. This "software only"

emphasis was changed in version UML 2.0, where the scope of the language was widened to include systems in general, [13]. 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. Additionally, the use of SysML is not limited to OO methodologies as stated early on in the specification. "The language is intended to support multiple processes and methods such as structured, object-oriented, and others..." [14] Both UML and SysML are used extensively to provide Model Driven Development (MDD), where the model and not documents drive the development process.

1.3 Organising Principles.

Both UML and SysML employ a variety of organising principles. These include 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. [5] 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. Additionally, these need to be addressed early on in the project as they can have far reaching consequences for both product and process. Just as no architect would complete the design for a house, and only then address plumbing, gas, and electricity, no system designer would finish their design without having addressed safety, traceability and risk. Microsoft Windows is a notable example where interoperability and security were added on after the initial releases, and the industry is only too painfully aware how long that has taken to be resolved, assuming that in fact it has been. Consequently, it is essential that mechanisms exist to allow system modellers to document these concerns and how they will be addressed / resolved.

1.4 Early Aspects.

In computer science, an aspect is a part of a program that cross-cuts its core concerns, therefore violating its separation of concerns. [18] "Early aspects are defined as cross-cutting concerns in the early life cycle phases including the requirements analysis, domain analysis and architecture design phases" [18]. By their nature, early aspects cannot be localized and tend to be scattered over multiple early life cycle modules. This reduces the modularity of the artefacts in the early life cycle which might consequently lead to serious maintenance problems. Of great management concern is the fact that the early system development phases actually set the early design decisions and have a large impact on the whole system. Stoewer [17] reported that 85% of the total system development costs are determined during the first 15% of the system lifecycle. Bennatan [1] found that up to 40% of the project time can be spent on Requirements and planning. Coping with aspects at the early life cycle phases as such is essential. Consequently, it is essential that cross-cutting concerns are addressed in modelling environments.

Figure 1 shows an example of how cross-cutting concerns relate to the model. The model is made up of Requirements, Application and Hardware hierarchies. These are shown as the light blue boxes, red boxes, and dark blue circles respectively. The hierarchies are made up of components and elements related to each other via association and ownership. Each of these hierarchies has relationships with the other hierarchies such as satisfy, trace, deploy, uses and constrains shown by the dashed arrows. 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. The example shown is a simplified version of a model to illustrate the concepts in this paper.

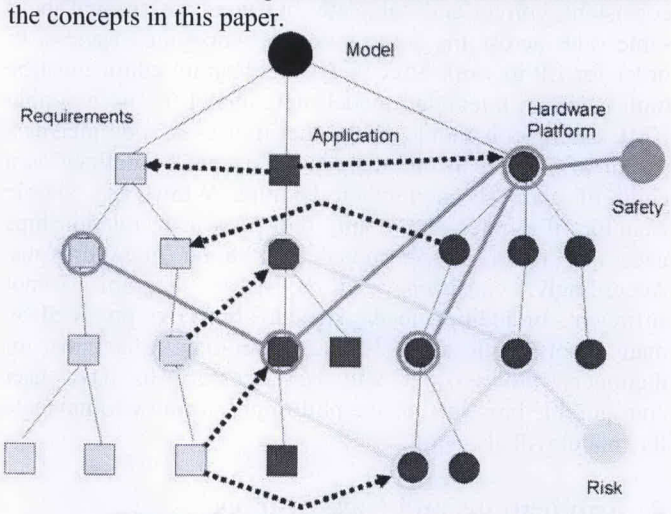

Figure 1 Conceptual System Architecture

2 Multiple Views and Viewpoints

Traditionally, engineers have addressed separation of concerns by creating different views and viewpoints of a system. A View is a "representation of a whole system from the perspective of a related set of concerns" [10]. Views are not necessarily orthogonal, but each view generally contains specific information. 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 by establishing the purposes and audience for a view and the techniques for its creation and analysis" [10] As regards a SysML model, a viewpoint then could be considered 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.

3 Ergonomic Profiling in UML

A UML profile constitutes a coherent set of stereotypes, constraints, tag definitions, and tagged values, defined for specific purposes. [13] If we group common requirements stereotypes together into a collection, we can create a

requirements profile, to support SysML for example. In addition, the appearance of the diagrammatic elements on the screen must be in a format familiar to modellers. 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 consequence of deleting elements that have relationships to others. This is called Ergonomic Profiling (EP). Additional aspects of Ergonomic Profiling are described in the following sections.

3.1 Integrated Models

A model editor that ensures that a single diagram is consistent, correct and complete, but does not enforce these same rules across the entire model is worse than useless. In order for EP to work effectively, the diagram editor must be built using an integrated model meta-model. Using a simple UML example, it should ensure that 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. In addition to the standard browsers provided by many tools such as a Package hierarchy, diagrams, or dictionary browsers, it will be necessary to have user configurable browsers, as the philosophy for how to navigate the model will also vary.

4 Applications and Case Studies

The following examples illustrate the points discussed above. They look at different concerns, and demonstrate how Ergonomic Profiling is used to provide a visual representation of the concern, and enforce relationships and rules. Finally, they allow report generation to demonstrate that the viewpoint is consistent, coherent, and complete. Most of these profiles are being used by customers on active projects. Those examples have been deliberately made generic to ensure anonymity, and protect customer Intellectual Property.

4.1 Robustness

In order to illustrate the previous points, we will start with a simple example. The robustness diagram is an example of a diagram created to address specialized concerns in a UML model. Rosenberg & Scott [15] describes a technique called robustness analysis. 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. The 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. In other words they can be used to ensure that your use cases are sufficiently robust to represent the usage requirements for the system under construction. Another use is to identify potential objects or object responsibilities to support the logic of the use case and to ensure that the object relationships are consistent within their defined subtypes. 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.

A robustness diagram is basically a simplified UML communication/collaboration diagram. Its chief benefits are that it uses graphical symbols to show its different subtypes and can enforce rules governing the relationships between these subtypes. It makes use of well-known object oriented subtypes such as entity, control, and boundary, as well as actors, defined below.

Actors. Any external object that interacts with the system, but is not part of the system – the Stick Man.

Boundary objects. Objects that enable actors to interact with the system. These are often called interface objects – The green circle with the vertical bar to the left.

Control objects. Their job is to manage the various elements and their interactions. These are also known as process objects or simply as controllers – the yellow circle with the arrow on top.

Entity Objects. These are entity types representing persistent data and message information – the cyan circle with the horizontal bar.

An example Robustness diagram depicting a trade use case is depicted in Figure 2.

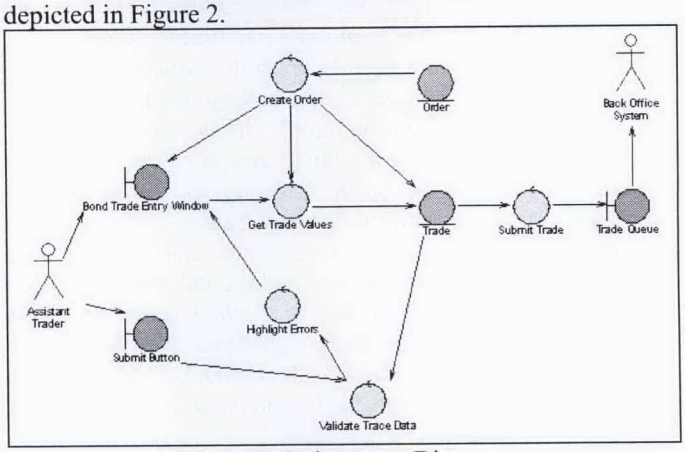

Figure 2: Robustness Diagram

It is also important that correct object type relationships are enforced. For example, Control Objects can communicate with both entity and Boundary objects, however Boundary objects cannot communicate with Entity objects. Also, actors may only communicate via Boundary Objects. The objects having been analyzed, classes can then 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. Additionally, a set of scripts was created to enforce the rules described above. These are all shown in Figure 3. In addition, the objects themselves are represented in the browser using the corresponding icons.

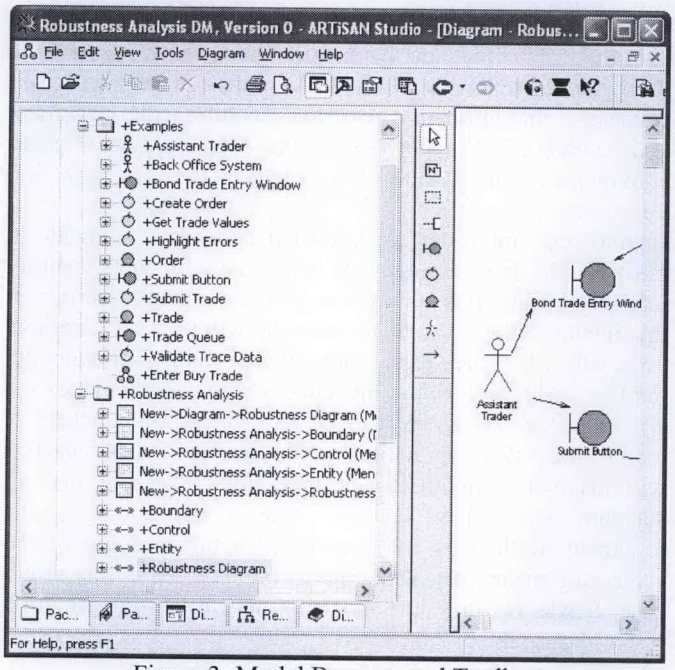

Figure 3: Model Browser and Toolbar

4.2 Requirements

System Engineers have to deal with many different stakeholders, as well as many different categories of requirements. Hooks [8] identifies a checklist of requirement drivers as "Functional, performance, interface, environment, facility, transportation, deployment, training, personnel, reliability, maintainability, operability, safety, regulatory, security, privacy, and design constraints". Such a wide range of requirements will manifest themselves in different formats in the requirements specification, system model and throughout the development lifecycle. Integrating them into the model will enable greater visibility, and provide a direct means of indicating traceability and compliance. This is demonstrated in the following examples, but most notably the requirements and safety case examples.

4.2.1 SysML Requirements Diagram.

One of the goals of SysML is the ability to integrate requirements into a UML model [14]. The Requirements Diagram was created to support the underlying Requirements Model. "The requirements model describes the SysML support for describing textual requirements and relating them to the specification, analysis models, design models, etc. A requirement represents the behaviour, structure, and/or properties that a system, component, or other model element must satisfy". [14] Requirements specifications can be modelled containing sets of requirements. Each requirement will have the text "shall" statement, as well as priority, safety critical level, etc. The requirements model includes relationships among requirements and between requirements and other model elements. The containment relationship shown in Figure 4, (circle with a "+" sign) is used to decompose a requirement into its constituent requirements. A derived requirement can be related to one or more source requirements using the derive relationship. A requirement can be linked to a model element that is intended to realize or satisfy the requirement. The trace and satisfy dependencies can be grouped to define the rationale for a set of traces or a set of links, for example. This enables the modeller to identify supporting information that provides the basis for a derived requirement, a design rationale, or a reference to a test case that verifies a particular relationship.

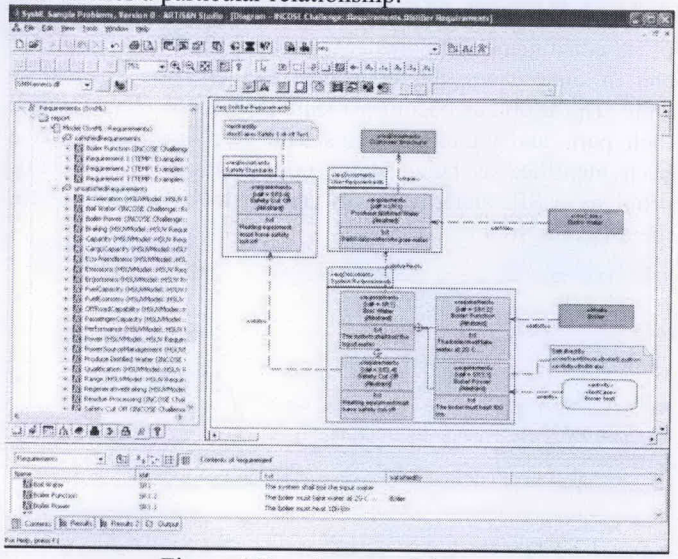

Figure 4 Requirements Diagram.

More specialized requirements can be specified using stereotypes. For example, safety requirements can be specifically identified. The requirements model can be shown in graphical, tree structure, or tabular format [14]. The graphical format using nodes and paths is shown in Figure 4. The different types of model elements are differentiated by the stereotype labels, shapes and colours. For example, requirements are green, blocks are pink, and test cases are white rounded rectangles. Reports can also be created for the model as shown in the browser pane on the left, where satisfied and non-satisfied requirements are listed. A summary of the requirements is shown at the bottom with the name, id, text, and satisfied by elements listed. Finally, to create the diagram, toolbar buttons are provided for requirements, and satisfy, verify, etc relationships.

In this diagram, the Safety Standards, User Requirements and System Requirements area 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. Hause and Thom [6] describe how UML/SysML supports the modelling of these different types of requirements in more detail.

4.3 Safety Integrity Levels

The Safety Integrity Level (SIL) concept was created in order to quantify the safety requirements of a specific components or set of components in a safety related system [9]. Specifically, a SIL is the likelihood of meeting required safety features. One of the problems in dealing with SIL is traceability to the component and connecting parts. In other words, if component A is required to comply with SIL 2, all of its constituent components must also comply with SIL 2, and in some cases, interconnected elements are affected as well. The problem becomes identifying these elements and their parts and interconnections, and verifying that all have been identified as being at the correct level. To solve this problem, a SIL stereotype was created with Tag Values for the different SILs.

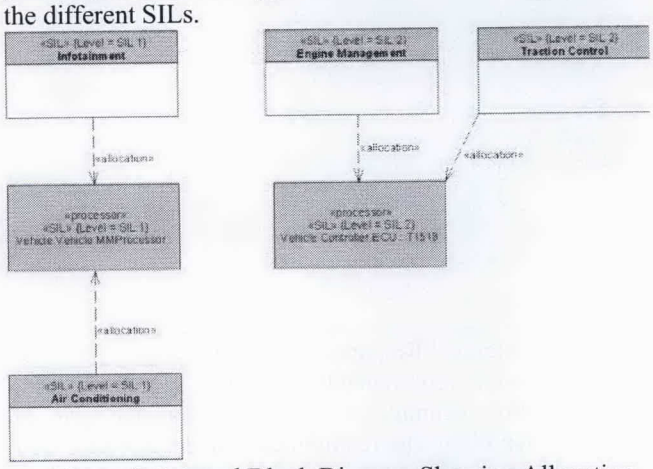

Figure 5. Internal Block Diagram Showing Allocation

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 5. 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.

4.5 Safety Cases

It is beyond the scope of this paper to offer a complete definition of safety cases. However, a brief overview will be included to illustrate certain points. For further information, the reader is directed to [11]. Their paper contains a detailed description of safety cases and their components. These definitions are reproduced below. "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.

Engineering and verbal skills do not necessarily coincide in most people. Indeed, engineers are famous for their inability to communicate. It is common to find that textual descriptions that include complex logical reasoning will be hard to follow, especially when they are written by engineers. Consequently, the text and therefore the meaning and considerations of the written safety argument, can be ambiguous and unclear. If many different interpretations of the text can be drawn, this will make 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. It was for this purpose that Goal Structuring Notation (GSN) was created.

4.5 1 Goal Structuring Notation.

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.

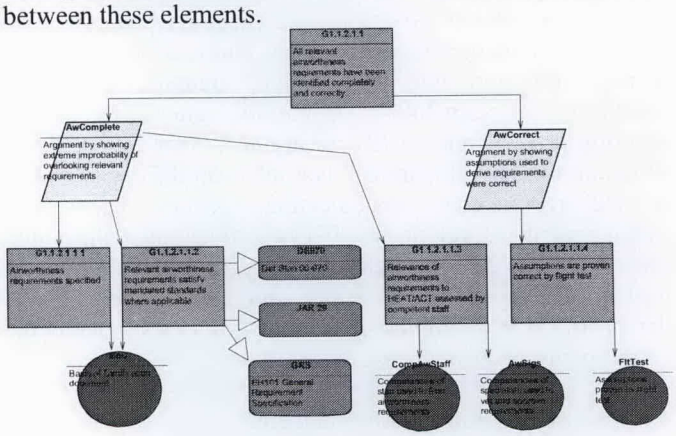

Figure 6. GSN Reproduced from Chinneck, et al [2]

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 as shown in Figure 6. The diagram is a modified class diagram with stereotyped classes. Goals are shown as green boxes, context as blue rounded rectangles, solutions as red circles, strategies as yellow parallelograms, and undeveloped goals as light green boxes with a diamond underneath (not shown). These can be linked together via the connectors, with checks done to ensure notational compliance. Reports and traceability consistency checks can be created to ensure a cohesive and coherent model. Chinneck [2] found that "reading and review of other safety cases were significant in building confidence in our own ability to recognise – and subsequently produce – well-structured and robust

arguments." Also, the author received a "number of complements relaying how simple the document was to read and understand. These comments came from both technical and non-technical staff, proving the benefit of GSN in representing the argument in a clear and unambiguous fashion."

4.5.2 An Integrated Safety Case.

Because many of the elements of the safety case and safety argument are already contained within the model it is useful to integrate GSN and the 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. Hause & Thom [4] describes how use cases in general can be used to support a safety case.

5 Conclusion

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 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. Ergonomic profiling provides a means of examining the model from the different viewpoints, as well as in-built consistency checks and reports. The methods outlined in this paper provide these capabilities and do so in an unobtrusive, flexible, and UML compliant manner, making use of the extension mechanisms available in UML. They have been used successfully on projects and have helped ensure their successful completion.

References

[1] Bennatan, E.M., On Time Within Budget, Third Edition, Wiley Canada, 2000.

[2] Chinneck, P., Pumfrey, D, McDermid, J., The HEAT/ACT Preliminary Safety Case: A case study in the use of Goal Structuring Notation, accessed online July 2007, Available from . http://www-users.cs.york.ac.uk/~djp/publications/Chinneck-Pumfrey-reviewed.pdf

[3] Dijkstra, Edsger, 1974, On the Role Of Scientific Thought, accessed online November, 2005 from http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html

[4] Hause, M., Thom, F., Using Use Cases and Scenarios as Part of a Safety Case, May 2002, INCOSE UK Chapter – Spring Symposium 2002 Proceedings.

[5] Hause, M., Thom, F., Creating Flexible Architectures for Systems Engineering, July 2004, INCOSE International Symposium 2004 Proceedings.

[6] Hause, M., Thom, F., Modelling High Level Requirements in UML/SysML, July 2005, INCOSE International Symposium, Rochester 2005 Proceedings.

[7] Hause, M., The Systems Modelling Language - SysML, Sept 2006, EuSEC 2006 Proceedings.

[8] A. Hooks, I., "Writing Good Requirements." Proceedings of the Third International Symposium of NCOSE. Vol 2, 1993.

[9] IEC, Functional safety and IEC 61508 A basic guide, November 2002, The International Electrotechnical Commission (IEC) available online from http://www.iee.org/oncomms/pn/functionalsafety/HLD.pdf

[10] IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description of Software Intensive Systems [online] Available from http://standards.ieee.org [Accessed November 2005].

[11] Kelly, T. and Weaver, R. The Goal Structuring Notation – A Safety Argument Notation, DSN 2004 Workshop on Assurance Cases: Best Practices, Possible Obstacles, and Future Available online from http://2004.dsn.org/

[12] Object Management Group (OMG), 1997, UML Semantics, version 1.1. OMG Document: ad/97-08-04, Available from www.OMG.org [Accessed September 2005]

[13] Object Management Group (OMG), 2004. OMG Unified Modelling Language Specification. [online] Available from: http://www.omg.org [Accessed September 2004].

[14] Object Management Group (OMG), 2006, Systems Modelling Language version 1.0, Available from www.omgsysml.org. [Accessed July 2006].

[15] Rosenberg, Doug and Scott, Kendall, Use Case Driven Object Modelling With UML (1999)

[16] Sommerville, I. And Sawyer, P., Requirements Engineering, A good practice guide, John Wiley and Sons, June 2000..

[17] Stoewer, H., 2003, Competitive Systems Engineering, Keynote address to the INCOSE UK Spring Symposium, Maldon Essex, 2003. Available online.

[18] Van Den Berg, et al, AOSD Ontology 1.0 - Public Ontology of Aspect-Orientation, May 2005, Available online.