SafeSlice: A Model Slicing and Design Safety Inspection Tool for SysML

Davide Falessi[1,2] Shiva Nejati[1] Mehrdad Sabetzadeh[1] Lionel Briand[1] Antonio Messina[2]

[1]Simula Research Laboratory
Oslo, Norway

[2] University of Rome (Tor Vergata)
Rome, Italy

{falessi,shiva,mehrdad,briand}@simula.no ant.messina@gmail.com

ABSTRACT

Software safety certification involves checking that the software design meets the (software) safety requirements. In practice, inspections are one of the primary vehicles for ensuring that safety requirements are satisfied by the design. Unless the safety-related aspects of the design are clearly delineated, the inspections conducted by safety assessors would have to consider the entire design, although only small fragments of the design may be related to safety. In a model-driven development context, this means that the assessors have to browse through large models, understand them, and identify the safety-related fragments. This is time-consuming and error-prone, specially noting that the assessors are often third-party regulatory bodies who were not involved in the design. To address this problem, we describe in this paper a prototype tool called, SafeSlice, that enables one to automatically extract the safety-related slices (fragments) of design models. The main enabler for our slicing technique is the traceability between the safety requirements and the design, established by following a structured design methodology that we propose. Our work is grounded on SysML, which is being increasingly used for expressing the design of safety-critical systems. We have validated our work through two case studies and a control experiment which we briefly outline in the paper.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications

General Terms

Design, Verification

Keywords

Safety Certification, SysML, Traceability, Model Slicing

1. INTRODUCTION

Models have long been used for capturing the design of safety-critical software in domains such as avionics, railways, maritime, and automotive. In addition to being one of the main drivers of the development process, models in these domains further serve as a key source of information for third-party assessment activities, particularly certification, where an external professional or regulatory body needs to verify that a system does not pose unacceptable risks to people or the environment. An important step in the certification process is for the assessing body to review (inspect) the design models and ensure that the design meets the safety requirements.

The tool we describe in this paper, SafeSlice, was motivated by the difficulties we observed during third-party inspections of the design of safety-critical software. The focus of our investigation was on the inspections performed during the certification of integrated control systems in the maritime and energy sector, but similar problems can be expected in other safety-critical domains, such as the automotive sector, where (software) safety certification is an emerging topic. In particular, we observed that the design models could be large and their safety-related aspect not clearly identifiable. Subsequently, checking compliance between the safety requirements and the design turned into a time-consuming endeavor, often forcing the inspector to browse through large models and manually analyzing a large numbers of dependencies before the safety-related design aspects could be determined.

To address this problem, we have developed a model slicing algorithm that enables inspectors to automatically extract model slices (fragments) related to safety requirements. This reduces inspection effort, and further makes it less likely that safety issues would be overlooked. Of course, this automation does not come for free and requires the development team to follow certain guidelines to link the safety requirements to the design. In [4], we elaborate a design methodology which ensures that the traceability links required for automated slicing are properly established. The methodology and the slicing algorithm in [4] are the basis for the SafeSlice tool that we describe in this paper.

Specifically, SafeSlice enables users to: 1) specify the traceability links envisaged by our design methodology, 2) check the consistency of the established links, and 3) automatically extract slices of design with respect to requirements. These slices can then be used for conducting inspections and ensuring that the design satisfies the safety requirements. In addition to implementing our proposed traceability mechanism and slicing algorithm, SafeSlice provides facilities for managing inspections and report generation.

We ground our work on the Systems Modeling Language (SysML) [7], as SysML is becoming the de-facto modeling standard for systems engineering [6]. Since safety is a system issue, and not just a software issue, this was a natural choice to ease the adoption of our approach in practice.

The rest of the paper describes the SafeSlice tool and summarizes the evaluation we have done to validate the approach behind it.

2. TOOL OVERVIEW

Figure 1 shows a high-level view of the process implemented by the SafeSlice tool. The process consists of three main steps: (1) Establishing traceability between safety requirements and SysML

ESEC/FSE'11, September 5–9, 2011, Szeged, Hungary.

Figure 1: Method of use for SafeSlice.

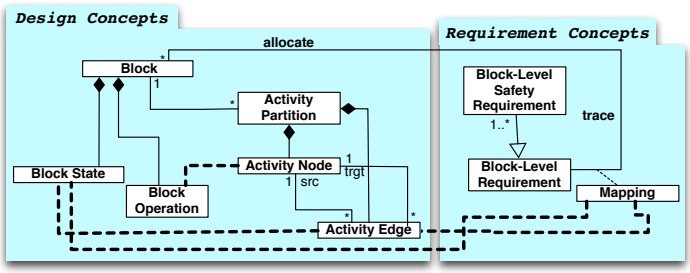

Figure 2: A Fragment of SafeSlice's traceability information model.

design diagrams, (2) Generating design slices based on the traceability links, and (3) Inspection of design slices by assessors. We require that the traceability links defined in the first step should conform to a traceability information model. This information model (discussed later in this section) provides a SysML adaptation of the traceability information requirements gleaned from observing actual certification meetings, consultation with certification experts, and reviewing major safety standards (most notably IEC 61508 [2]). The traceability links between safety requirements and the SysML design (defined in accordance with the information model) are used in the second step to automatically extract minimized slices for the safety requirements. These slices are then reviewed by the assessors to ensure that the safety requirements are met by the design. In the rest of this section, we elaborate the three steps of the process shown in Figure 1.

2.1 Establishing Traceability

SafeSlice is implemented as a plug-in for the Enterprise Architect tool (`http://www.sparxsystems.com/ea`) which, like other tools with support for SysML, has an environment for building SysML models and creating SysML's built-in traceability links. However, as we argue in our earlier work [4], existing SysML traceability mechanisms do not provide adequate support for addressing all the traceability information requirements mandated by safety standards and asked for by the assessors during certification. In [4], we provide methodological guidelines on how to extend and use SysML traceability links for the purpose of safety certification. These guidelines led to an information model, a small fragment of which is shown in Figure 2 – see [4] for the complete model. The fragment shown characterizes the links from requirements to SysML blocks and activity diagrams (the diagram types used later in the paper for illustration) along with the rationale for the links.

"*Rule Assistant*" is a feature of SafeSlice that enables guided and validated application of our proposed traceability information model. Our information model yields a set of 20 multiplicity and compatibility relational constraints (rules). We encoded these rules into a spreadsheet that can be easily edited by users to add new rules or to modify/delete the existing ones. Rule Assistant loads the entire set of rules in the spreadsheet at every invocation and executes the rules using the EA built-in rule engine. If a violation is detected, Rule Assistant provides diagnostic information about the violated rule, the model element(s) involved, and the action(s) to be taken to resolve the issue. As an example, in Figure 3, we show the diagnostic information generated by the Rule Assistant for the situation where a safety requirement has been defined but it is unspecified whether the requirement is at the system level or

```
ERROR - RULEID: 1
        -> DESCRIPTION: "The 'Safety Relevant' or 'Non-Safety Relevant' requirement
           should have 'Level' tagged value specified ('System level' or 'Block level')"
        ->  CATEGORY: Requirement tagged value unexpected/not specified
        -> HOW TO FIX: "Choose a value for selected Requirement's 'Level' tagged value"
        -> FIND THE ELEMENT (otElement): click here to select specified element
---- CHECK SUMMARY ---
Number of analyzed elements: 496
Number of checked rules: 177
Nymber of checked diagrams: 66
Elapsed scan time (sec): 22 (0 hours, 0 min, 22 sec).
----- Total Violations ----
Errors: 1
Warnings: 0
```

Figure 3: Example output from Rules Assistant.

at the block level. In this example, the user can read the "How to fix" field and then click on "Find the element" to quickly navigate to the element in question and apply the necessary fix.

Rule Assistant can check the information model rules in two modes: *investigator* and *listening*. In the investigator mode, the compliance of an entire project is checked. In this mode, the user waits until the compliance checking process is finished and then applies the suggested changes if violations are detected. In the listening mode, Rule Assistant is always active in the background and monitors every change made by the user and checks that the change is consistent with the information model. If a violation is detected, feedback is immediately shown in the diagnostic window. The time required by Rule Assistant to check all the rules is small. On a standard laptop, it took less than half a minute to check a SysML design with about a thousand elements (blocks, relations, activities, transitions, states, attributes, and operations) against all the 20 rules.

2.2 Slice Generator

"*Slice Generator*" is an implementation of the slicing algorithm proposed in [4]. A high-level overview of algorithm is shown in Figure 4. Briefly, a system safety requirement is refined into safety requirements at the level of blocks. If necessary, more detailed safety-relevant requirements could be defined in order to satisfy block-level safety requirements. In our tool, slices are constructed for atomic requirements, i.e., requirements that are directly related to design via traceability links. System-level requirements are never related to the design directly, because even in the simplest case where these requirements do not need to be decomposed, they still need to be allocated to some block and restated as a block-level requirement. If a block-level safety requirement is atomic, then our tool will generate a slice directly for the block-level requirement; otherwise, one slice will be generated for each of the atomic safety-relevant requirements contributing to the satisfaction of a block-level safety requirement. For example, Figure 5 shows a safety requirement, the traces, and the generated slices in terms of block and activity diagrams. The system underlying the example is a well-known benchmark embedded system, named the Production Cell System (PCS) [3], whose function is the transformation of metal blanks into forged plates by means of a press and their transportation from a feed belt into a container. We use PCS as a benchmark case study for our approach (see Section 3).

2.3 Inspection Assistant

The "*Inspection Assistant*" feature aims to help users in better managing the inspection process. In particular, Inspection Assistant can record the decisions made during (1) inspections of atomic requirements (conducted over design slices), and (2) inspections done to ensure that the atomic requirements together lead to the satisfaction of higher-level requirements and ultimately the system-level safety requirements.

Algorithm. *GenerateSlice*

Input: Requirement *R*, and a set of SysML design diagrams conforming to the input traceability information model
Output: Design slices related to *R*.

Step 1. Find design elements (indirectly) related to *R*.

Step 2. Extract block diagram slices:
Step 2.1 Remove operations, and attributes that are not related to *R*.
Step 2.2 Include relations between the blocks linked to *R*.

Step 3. Extract activity diagram slices:
Step 3.1 Identify activity partitions related to the blocks linked to *R*.
Step 3.2 Remove activity nodes and edges not related to *R* from these partitions.
Step 3.3 Add stuttering edges to maintain the connectivity between the diagram nodes.
Step 3.4 Identify new initial activity nodes.

Step 4. Extract state machine diagram slices:
Step 4.1 Identify state machines related to the blocks linked to *R*.
Step 4.2 Remove states and transitions not related to *R*.
Step 4.3 Add stuttering transitions to maintain the connectivity between the states.

Figure 4: Overview slicing algorithm in SafeSlice

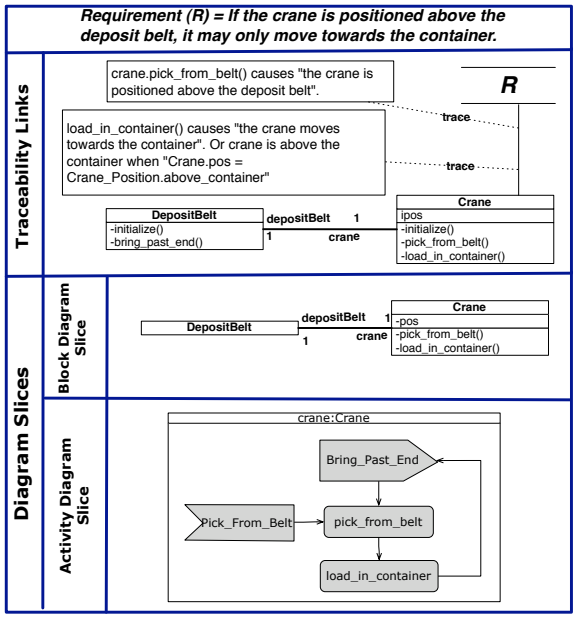

Figure 5: A safety requirement (R), the traces, and resulting slices.

Figure 6 shows the various states an atomic requirement can go through during the inspection process. The initial state is "Yet to approve". The user then reviews the design and marks the requirement as "Approved" or "Not approved", depending on whether she deems the design as satisfying the requirement or not. Because the design evolves over time, it is important to check the impact of design changes on safety requirements. Based on the traceability information, Inspection Assistant detects the requirements that need to be re-inspected due to changes. Specifically, if there is a change made to any of the block operations or attributes to which an atomic requirement is traced, then that requirement needs to be re-inspected. The status of an atomic requirement that needs to be re-inspected is set to "Out of date"; the status of the related higher-level requirements is set to "Out of date" as well. In addition, any change to the textual description of a requirement (at any level) would render the state of that requirement "Out of date" and the state will be propagated up to all the higher-level requirements.

To facilitate monitoring the progress of the inspection activities, Inspection Assistant can generate pie charts to visualize the relative proportion of requirements in different states. An example is shown in Figure 7. This pie chart depicts the status of the safety-relevant requirements that contribute to a selected system-level requirement (intermediate block-level requirements were filtered in this chart).

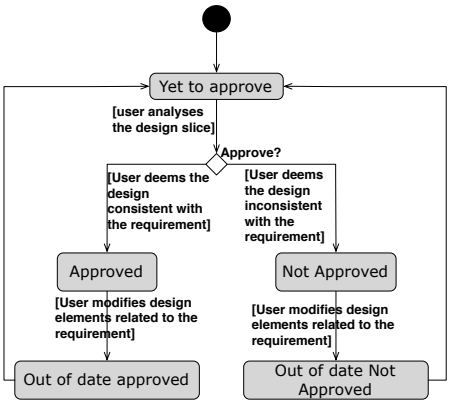

Figure 6: Inspection states for an atomic requirement.

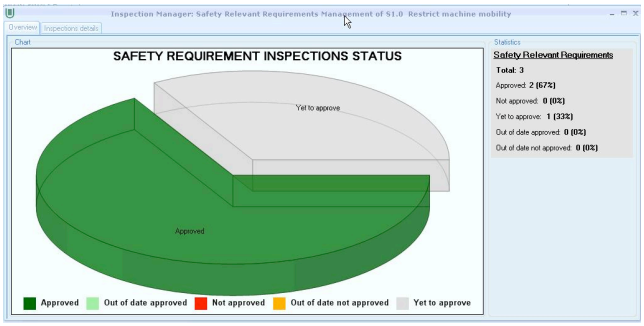

Figure 7: Monitoring the inspection progress using pie charts.

The chart indicates that there are three safety-relevant requirements for the given system-level requirement and of these, two have been already approved and the third is awaiting inspection. To make re-inspections more efficient, our tool keeps the previously-generated slices and allows the user to compare the new slices against the old ones. In this way, users can better investigate the differences.

2.3.1 *Report Generator*

Report generation is an important feature for supporting safety inspections. Reports are useful, for example, when a printed document needs to be signed off for legal obligations or simply for sequential reading of the inspection material. SafeSlice supports the automated generation of reports in PDF format. In particular, the tool allows the user to select the information to include in the report (e.g. design slice, whole design, statistics, pie charts). For instance, it is possible to generate a document reporting the inspection details of a given requirement or one that reports all the inspection details of all the requirements in a project.

2.3.2 *Advanced Navigator*

To make the inspection process more effective, SafeSlice supports advanced search and navigation of model elements. The Advanced Navigator feature allows the user to select an element x (e.g., a requirement, class, block, etc.) and retrieve the elements of given types (e.g., specific requirements, classes, blocks, etc.) that have a given type of relationship with x (e.g., satisfy, trace, derive, etc.). The user specifies the element/relation types to search for from a pre-defined list. This list can be modified by the user if needed, e.g., when a new SysML stereotype is defined for capturing new types of elements.

3. EVALUATION

We have evaluated the traceability and slicing techniques implemented in SafeSlice in several ways using a combination of formal methods [4], benchmark and industrial case studies [4], and a controlled experiment [1]. We outline our evaluation work below.

First, we provide a formal analysis of our slicing algorithm whereby we show that the generated design slices are sound for temporal safety properties [4]. That is, if a requirement holds over a design slice, it holds over the original (non-sliced) design as well. This ensures that the design slices will not be providing misleading information to the assessors.

Second, we have used SafeSlice in two case-studies [4]: the first case study is a benchmark case, the Production Cell System (PCS) [3], mentioned earlier in the paper (see Figure 5). The second case study is a real-world industrial system from the maritime and energy domain. The PCS design consisted of 58 diagrams with 479 elements, 419 relations, and 189 attributes. The industrial case study included 23 diagrams, with 194 elements, 186 relations, and 57 attributes. In both case studies, SafeSlice was used for creating traceability links, checking their compliance, and generating design slices. Our case studies indicated that: (1) The cost of applying our methodology, which mostly depends on the cost to establish traceability links, was reasonable. In particular, this was confirmed in the second case study by our industry partner. (2) Our slicing technique yields a substantial reduction in the amount of information that needs to be inspected to check that a given safety requirement is met by the design.

Lastly, we executed a controlled experiment to assess the impact of our traceability and slicing mechanism on inspectors' conformance decisions and effort [1]. Results clearly show benefits in terms of increasing the correctness of conformance decisions (from 50% to 63%), decreasing the proportion of uncertain decisions (-45%), and reducing the effort of inspections (-27%).

In summary, our evaluation suggests that our approach is effective for improving design safety inspections and thus of value in realistic settings for improving accuracy and reducing costs in software safety certification.

4. DESIGN AND IMPLEMENTATION

As stated earlier, SafeSlice is implemented as a plugin for Enterprise Architect (EA), which is a versatile and mature modeling environment. This makes it possible to seamlessly introduce SafeSlice into real development settings and further simplifies tool evolution and maintenance. Among the possible alternatives for modeling environments, EA was chosen due to its usability, wide industrial adoption (confirmed by our industrial partners), availability of detailed guidelines for plugin construction, and built-in support for storing and linking heterogeneous development artifacts (e.g., natural language requirements specifications, UML/SysML models, Word documents, source code).

Figure 8 shows the architecture of the SafeSlice tool. It communicates asynchronously with EA via events. All the information related to a development project is stored by EA in a database. The plugin can read from and write to this database via EA's API. In particular, the additional traceability information required in our methodology, previously-generated design slices and reports, and the decisions made by users during inspections are all stored and retrieved by the plugin via the API; this communication layer thus simplifies the implementation by hiding the underlying database technology.

SafeSlice builds on Microsoft ActiveX COM technology. We used Microsoft .NET Framework 2.0 and Visual Studio 2008 as the development platform. SafeSlice is written in Visual C# and is

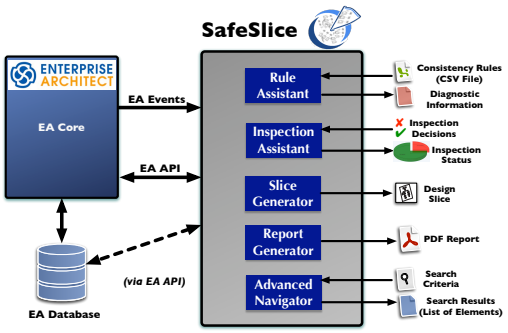

Figure 8: SafeSlice tool architecture

roughly 10,000 lines of code excluding comments and third-party libraries. SafeSlice was publicly released in May 2011. The tool is available at: http://sites.google.com/a/simula.no/safeslice/

5. CONCLUSION

We presented a tool, SafeSlice, for extraction and inspection of design slices (where the design is expressed in SysML) for use in certification of safety-critical software systems. The tool enables users to specify the traceability links envisaged by a given traceability information model, check the consistency of the established links, automatically extract slices of design with respect to safety requirements, and manage the design inspection process. Our evaluation indicates that the use of design slices substantially reduces the amount of information that needs to be inspected for ensuring that a given safety requirement is met by a SysML design model.

In the future, we plan to conduct studies to more conclusively quantify the gains in budget, time, and quality resulting from the use of design slices in the software safety certification process. Specifically, the quality of the slices generated by SafeSlice is related to the completeness and accuracy of the traceability information available. In the future, we plan to investigate whether the quality of the traceability information impacts the effectiveness of the inspection process.

SafeSlice is implemented as part of a larger research effort to build model-based support for software safety certification. A complementary part is a project aimed at developing conceptual models that precisely characterize the interpretation of dependability standards, such as IEC 61508, and recommended practices [5].

6. REFERENCES

[1] L. Briand, D. Falessi, S. Nejati, M. Sabetzadeh, and T. Yue. Traceability and SysML design slices to support safety inspections: A controlled experiment. Tech repo, Simula Research Lab, January 2011.

[2] IEC 61508: Functional Safety of Electrical / Electronic / Programmable Electronic Safety-related Systems, 2005. International Electrotechnical Commission: International Electrotechnical Commission.

[3] C. Lewerentz and T. Lindner, editors. *Formal Development of Reactive Systems - Case Study Production Cell*, volume 891 of *LNCS*. Springer, 1995.

[4] S. Nejati, M. Sabetzadeh, D. Falessi, L. Briand, and T. Coq. A SysML-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies. Tech repo, Simula Research Lab, May 2011.

[5] R. K. Panesar-Walawege, M. Sabetzadeh, L. Briand, and T. Coq. Characterizing the chain of evidence for software safety cases: A conceptual model based on the IEC 61508 standard. In *ICST*, 2010.

[6] W. Schafer and H. Wehrheim. The challenges of building advanced mechatronic systems. In *FOSE '07*, pages 72–84, 2007.

[7] OMG Systems Modeling Language (OMG SysML). 2008. Object Management Group (OMG), version 1.1.