2014 IEEE 15th International Symposium on High-Assurance Systems Engineering

Assuring Safety for Component Based Software Engineering

Philippa Conmy[1] and Iain Bate[1,2]
[1]Department of Computer Science, University of York, York, UK
[2]Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
email: {philippa.conmy, iain.bate}@york.ac.uk

***Abstract*—Developing Safety-Critical Systems (SCS) is an expensive activity largely due to the cost of testing both components and the systems produced by integrating them. In more mainstream system design, Model-Based Development (MBD) and Component-Based Software Engineering (CBSE) are seen as complementary activities that can reduce these costs, however their use is not yet well supported in the safety critical domain, as safety is an emergent property. The contributions of this paper are to describe some of the challenges of using these approaches in SCS, and then argue how through appropriate safety argument patterns the challenges can be addressed.**

I. Introduction

Software Intensive Safety Critical Systems is the term used for any software based system whose malfunction could cause harm to life, or the environment. Traditionally each system has been certified independently, via a collection of largely monolithic system analyses (described as "certification data" in this paper) which are used to demonstrate the system is adequately safe [1], [2]. Mainstream software engineering now successfully uses component based design using models. Such approaches are being adopted with some limited success in the safety critical domain, however many critical obstacles need to be overcome, including how to present a comprehensive safety argument or justification, backed up with component level certification data generated outside the current project (either re-used or generated independently).

One strategy for reusing verification evidence would be to determine which evidence is still valid and which needs verification repeating. However, the purpose of the majority of verification activities is not merely a "tick-box" exercise, but vital to ensure safe operation of components in a specific system context. Ariane 501 shows that context is the most difficult part of safely reusing components [3]. In the case of Ariane 501, sub-systems were reused unchanged and therefore those sub-systems had little extra testing performed. The designers had overlooked important context related to the change of processing platform and the interaction of the sub-system with the rest of the system. Therefore, a technique is needed to maximise the re-use of appropriate types of verification data, whilst maintaining the understanding of the safety of the overall system. In order to do this we need to understand the original context in which the data was assessed, our confidence in it's veracity and applicability, as well as summarising it's content. This paper addresses these issues.

This paper is laid out as follows. First we describe related work in this area, and background context. Then we show the overall approach and method to our work. Next we describe a number of the component argument templates, describing how they would be instantiated for different types of certification data. Then we describe how these may be used in an overall argument. Finally, we present conclusions and further work.

II. Related Work

Background research into this area has been grouped into three areas: Official industrial standards and guidance (as all safety critical systems developers will need to consider these), contracts to capture dependencies and interfaces on either component function or data, more general modelling techniques to support the process.

Very limited numbers of standards actively consider compositional certification, but those that do provide some basic requirements for the research presented in this paper. The aviation guidance document "Integrated Modular Avionics (IMA) Development Guidance and Considerations ED-124/DO-297" [4] looks at certification of a particular computer design paradigm for avionics systems. The standard recognises certifications are only given for aircraft or engines, however, certification data can be "approved" by an authority, making it easier to re-use. When anticipating the re-use of a component, the guidance requires that during initial development the designers must describe in depth the limitations and assumptions being made about the components' use in this system and others, i.e. the context.

The standard "Road Vehicles - Functional Safety ISO 26262", part 10 [5], contains guidance on development of Safety Elements out of Context (SEooC). This has a different initial stance, and allows generic components to be developed independently, and hence the evidence produced out of context. Components are expected to be reasonably generic, or conform to the standard AUTOSAR design paradigm [6]. Hence, it is assumed that they should be reasonably re-usable as long as, again, assumptions about how they will be used and the environment they

978-1-4799-3466-9/14 $31.00 © 2014 IEEE
DOI 10.1109/HASE.2014.25

will work in is well defined. The standard notes the need to manage mismatched assumptions, and how whether these impact on the overall safety of the vehicle, however doesn't give guidance on how to do this.

"Informal" contract based approaches of various kinds can be found in [2], [7], [1], [8]. These all consider how to capture various non-functional dependencies between safety related components, using natural language to express dependencies and guarantees. It should be noted that there may be a very complex chain of safety dependencies within a SCS, as shown in Figure 1 - a failure in one component, may be caused by another, mitigated by a third, and so on. A modular safety argument for CBSE will need to consider this, and how to reconcile and argue for the adequate safety of this chain of events, particularly where there may not be a complete match between functionality needed by one component and that offered by another.

A more formal approach to contract expression (specific to high assurance) is found in [9], where the authors present a restricted language for describing rely/guarantee conditions between software applications and computing hardware. This has the advantage that it can be used to generate automatically a limited set of arguments about the composed behaviour of the software, including for failure behaviour. However, this doesn't allow us to capture information such as qualitative properties of the certification data, or assumptions about (for example) environmental deployment. Other means outside of the safety domain, e.g. the Object Constraint Language [10], allow expression of some functional properties.

The approach within this paper is neutral to the actual means used to describe dependencies, we have assumed natural language to be used as part of our arguments claims, but more formal means are used for parts- making the type of matching process described in IV-A simpler in places.

In terms of modelling approaches, in [11] Habli describes methods for certification of product-lines, particularly looking at safety argumentation. A product-line is a set of similar products, which have common components (known as assets) which can be re-used within the set (possibly configured differently in each product). However this research is based around a specialised design methodology, and in a constrained domain. This paper provides a more general approach.

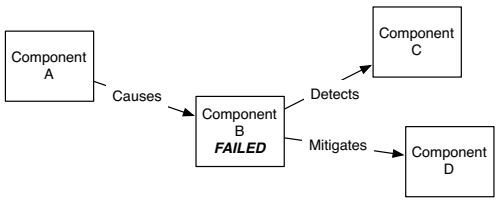

Fig. 1. **Failure Propagation and Mitigation**

III. Envisaged Process

This section presents an overview of the proposed approach to safety arguing, based around the a component model approach. We are assuming that a software component has been already designed, and implemented, and a component model is created for after or during the fact. During the design process a certain amount of what this paper calls "certification data" (this is anything used as evidence about a component such as analysis results, design documents, staff competencies etc.) will be produced for it. Our aim is to facilitate, and maximise, the re-use of this data using modular safety arguments. We note that some analyses may feed into other analyses, for example design reviews that lead to design alterations or requirements specifications that must be met in a software implementation.

The proposed approach to safety arguing is in two parts, the initial preparation for individual components, followed by a system argument phase during an actual system design (Figure 2). It is summarised as follows:

Preparation phase: Produce argument fragments for components and link to component model. These argument fragments are based on a catalogue of patterns, as presented in this paper.

System argument phase:

1) Produce top down system safety analyses, down to specific requirements. Express these in argument form, using the principles/patterns presented here.
2) Resolve and match argument fragments to the specific requirements - joining the top down and bottom up arguments. Safety arguitationment contracts will be used for this purpose.
3) Iterate and update design and arguments as necessary, e.g. where resolutions are not possible.

The preparatory phase process is summarised quite simply as follows. Ready prepared component models and certification data are combined with pre-defined safety argument patterns (and other information as appropriate), in order to produce safety argument fragments for a given component. The details of what should be included in component argument fragments is described in section IV-A of this document.

The system argument phase (Figure 2) is more complex, and has six main process parts (P1-6), and a number of decision points (D1-3). Various types of data that are produced and/or used throughout the process are detailed. The process must start with system safety analyses (P1), for example high-level failure modes and effects analyses, hazard identification, functional failure analyses. These techniques are well established and, hence, we do not describe them here (see [12] for a comprehensive overview). These create system specific hazard and risk data, and some system safety requirements which can be allocated as necessary to different components for fulfilment, and can shape the design. For example, duplicates might be used to fulfil reliability requirements, and several different

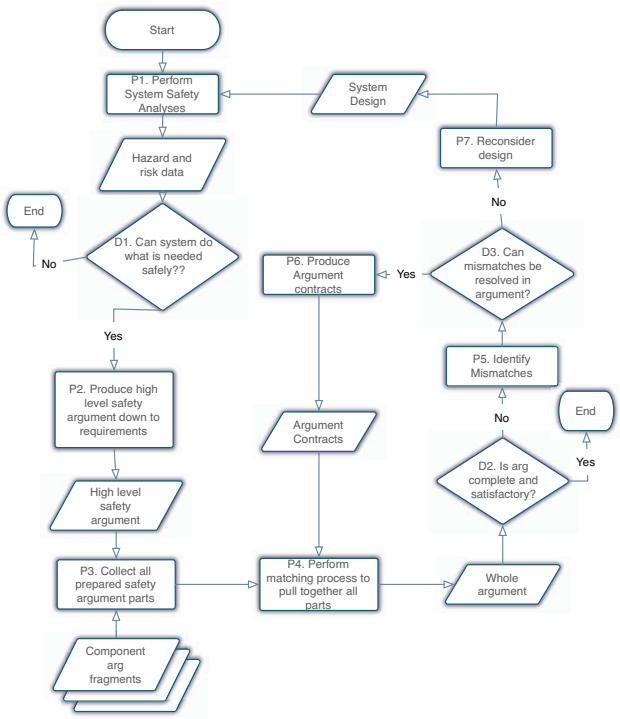

Fig. 2. **Process for producing system level safety arguments including argument fragments**

components might be used in concert (or independently) in order to meet a system level functional requirement. Note that it is necessary at this point to consider whether the system is going to be acceptably safe AND fulfil the functional needs. If it isn't then the current design should be abandoned.

A system level safety argument needs to be created or modified (P2), which describes how and why these requirements have been allocated. This argument is then collected with the component argument fragments for all components (P3). Once all the parts have been brought together, careful consideration is needed as to how to join the different arguments together (P4). In some cases it may be very easy to work out how to link goals together, in others it may be much more complex. However, the aim of this part of the process is to put together a complete argument to be assessed. After this there is a decision point: D2. If the argument is considered satisfactory then there is no need to continue. If it isn't then further analysis is needed to identify difficulties and mismatches (P5). Some of these mismatches may be resolved via argument, and some via re-examining the design. Any design mismatches must be addressed first, as any resolutions that mean a change to the design (P7) are likely to mean a change to the high level analyses in P1 (note the iterative loop). There is little point in addressing argument conflicts if the high level argument has to change again.

Assuming there are no design mismatches, and justifications can be made for the argument conflicts then a series of one or more safety argument contracts must be developed to explain how they are resolved. Once this has been addressed, the complete argument is once again pulled together (P4) and the argument must be reassessed. Once it is satisfactory then we can finish the process.

We note that this process diagram glosses over some aspects of a typical design process, such as smaller iterations and changes for reasons other than safety needs. However, we assume that the developer will take pragmatic steps to minimise the need to repeat system safety analyses over and over again, during the design iterations.

IV. Arguing The Safety of Components

A. Introduction to Goal Structuring Notation

The purpose of the safety arguing is twofold, first we wish to better support the re-use of safety related certification data about a component, second there is a genuine need to provide a justification that a system developed from multiple, independently developed, components is acceptably safe in context.

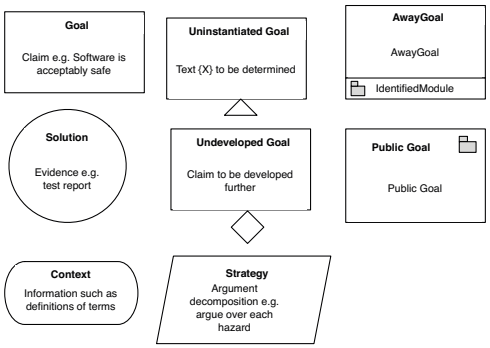

Fig. 3. **Goal Structuring Notation**

The arguments in this paper are expressed using the Goal Structuring Notation (GSN) [13] as shown in figure 3. Modular extensions to the GSN allow for composition of arguments from sets of fragments (or modules). When using modular GSN, a public goal (a guaranteed or offered claim) is matched to an away goal (a claim which cannot be supported in the current argument structure). As goals are typically expressed using natural language these may not be readily directly linked (due to slight differences in terms, levels of abstraction being described, and so on). For example, an away claim might be that a piece of software needs a guaranteed secure communication in order to function correctly. A public goal for different supporting software (in a separate argument module) may provide evidence that one particular secure method is provided. Some thought is needed by the developers as to whether the offered method is that required, and gives enough security. Where it is difficult to directly match claims we use a Safety Argument Contract (SAC). Note

that this contract is the resolution between two (or more) slightly mismatched claims, and is different to a design contract. The latter describes an interface.

B. Relationship between Components and Safety Arguments

In an attempt to make the claims matching process simpler, and provide a more structured methodology, we present a series of argument fragments patterns to guide the production of safety arguments that capture context, justifications and essential characteristics of the evidence. The fragments can be re-used within any system safety argument. The system safety argument will contain system specific hazard and safety requirements. The component arguments will be based on assumptions about these safety requirements (SRs). SACs will be used to join and resolve the assumptions with the actual requirements. This is summarised in figure 4.

The proposed method to ease matching of component argument fragments to a system level argument is to use the same high-level argument strategies for all sets of arguments. This is not intended to limit the content of the argument, but does constrain the broad organisation of the parts. For example, if components are used in similar systems in the same domain there may be high level functions common across all of them. E.g. Automotive systems will contain braking, steering, engine management functions, all met by a number of different components. For all components developed for these similar systems, it is envisaged there will be some idea of how they are to be re-used whilst they are being developed. Structuring the argument fragments in terms of how they might contribute to the high level functions would help immensely in merging the component fragments to a system level argument which is, again, structured around these functions.

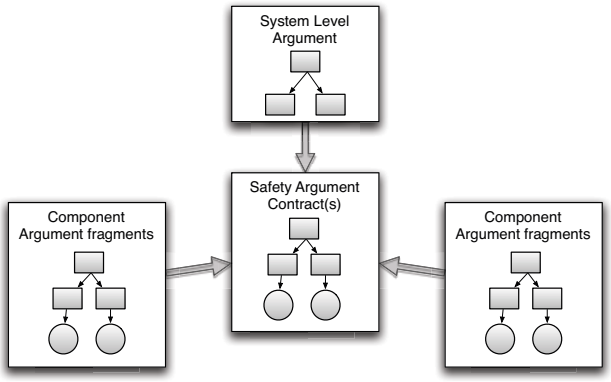

Fig. 4. **Summarising the relationships between instantiated argument fragments, system argument, and SACs**

However, this does become more difficult when considering components that may be used across multiple domains. Functions may be superficially similar (e.g. braking and steering in trains, planes and automobiles) but hazardous situations, environments, and hence tolerances etc. are so different that it is difficult, and potentially dangerous, to compare these directly. The SAC will be used to try and resolve differences, however, this paper concentrates only on similar systems. Support for cross-domain arguments will be examined in the future. The safety arguing approach presented in here is able to highlight commonalities as well as variabilities, by making explicit contextual information and assumptions about how a component is used.

In practical terms, this broad level strategy approach means there is a need to make certain goals in the component argument fragments worded in a particular way, or prominently justified, with all contexts explicit. This should make the away and public goal matching process, and resolution via SACs, simpler.

As part of a CBSE process, a set of high-level requirements for a system will be produced. This will include hazard identification. Then a set of components which can be used to fulfil these requirements will be found. The model-based approach allows us to quickly assess potential components, and model their interactions, without the expense of fully building the system. In addition, it allows us to re-use components by capturing their essential properties in a model. Similarly, argument fragments (with appropriate content) about the component properties can be re-used to assess early on whether component certification data is compatible, and whether components will safely interact with one another.

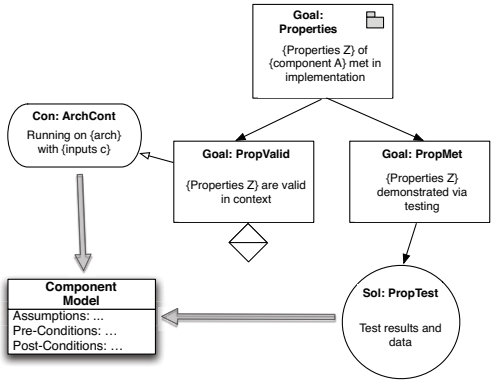

Fig. 5. **Links between safety arguments and a component specification model**

The relationship between a component model, and the type of information presented in the safety argument, is elaborated in Figure 5. Each component has a set of one or more design contracts associated with it (pre-and post conditions), and evidence supporting the contracts is linked to via the "solutions" at the bottom of an argumentation fragment. However, the argument is only valid within a set of defined, and assumed, contextual information (Assumptions). It should be noted that there might be more contextual information in the argument than that contained in the component model. For example, context such as which standards are being used for assessing and

developing the component. These relationships will be further understood as the component model is developed.

C. Component Model Argument Fragments

As described earlier, each component model will have one or more safety argument fragments attached to it. These argument fragments should concern properties of the actual component which are constrained and do not change fundamentally when a component is used. In other words, although some properties may be configurable for a component type, the type of configuration possible doesn't change. If it does then the argument may not be re-usable (as the configuration is part of the context within which the argument is constructed).

We noted that the argument fragments must contain appropriate content to allow the assessment of components suitability for a new safety system. Extensive discussions with industrial partners within the funding project consortia have identified that the safety arguments fragments for the component model need to be able to at least address the following things:

1) *Confidence in the evidence:* there are many aspects or characteristics of the evidence that may be subjectively assessed, via an argument. These include:
   a) Qualitative assessment of how well it has been applied, including: Completeness, Depth, Probability and Coverage
   b) Competency of personnel
   c) Risk associated with the age of the evidence or which version/configuration of a component it was applied to. In other words, is it directly relevant to our particular system.
   d) Appropriateness of a technique for showing a particular property
2) *Compliance to standards:* this type of safety argument would be used for a component developed using a particular standard as a reference point. For example, a component developed to ASIL B in ISO 26262 [5] would have a particular design flow, with analysis techniques at given points (some of which may or may not be applied). The argument would capture the depth of compliance. *for reasons of space, this argument is omitted from this paper, however the reader is directed to similar work on these specific types of argument in [14]*
3) *Traceability and specification:* an argument should be traceable to fundamental properties of the component that we wish to assert, and we should be able to argue about the qualities of the specification (see section IV-E).
4) *Proven in use:* it may be possible to gather evidence about a components operation, when it has been used in a system. For example, we may have compelling evidence about probabilities of hardware failure, or show that a component has tolerance in difficult operating environments. Capturing this in a safety argument (along with any caveats or limitations on the data) would be very valuable from a re-use perspective.

Argument patterns for each of 1, 3 and 4 are now presented, using the GSN pattern extensions [13]. These patterns contain high-level strategies for argument structure, with information to be instantiated contained within braces.

D. Confidence in Evidence Argument Fragment Pattern

We anticipate that the most frequently used argument fragment will be related to describing evidence, and its provenance and quality. A pattern for this is shown in Figure 6. The main purpose of this pattern is to capture qualitative information about the confidence we have in a piece of evidence. This type of argument may be re-used for all kinds of different pieces of evidence, and may be linked to some of the other argumentation fragments (as will be discussed).

The top level goal **AATop** has three pieces of information to be instantiated, the analysis technique type, what we describe as "confidence" and also which specified properties (or contracts as shown in the component meta-model) are being demonstrated. Earlier, we noted that the component argument modules would be most likely to contain very generic information about how a specification was met, rather than whether that specification was correct (something that can only be assessed in a system context). This pattern supports that viewpoint.

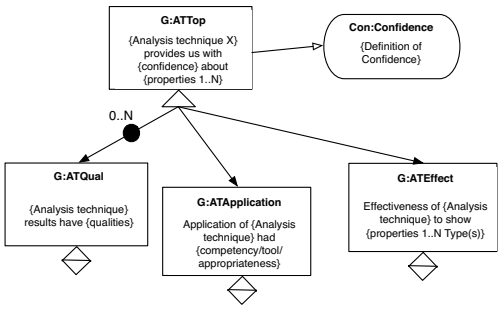

Fig. 6. **Confidence Argument Pattern**

A description of "confidence" is harder to define. It may be very difficult to capture an implicit understanding of how confident we are in the results, as it cannot be quantified, and will depend on the weight of the argument below. Nevertheless, it is useful to try and capture this understanding in a concise manner if we can. Note that we may also wish to state the limitations of the evidence here too.

The goal **ATQual** should be instantiated 0 to N times for the many different qualities of the evidence (indicated by the arrow with the filled in circle). For example, we may have some assurance of the results of a technique, such as testing, due to code coverage. We may have assurance of hardware tolerances due to stress testing in an appropriate environment. We anticipate that more specific (but still

re-usable) versions of this argument pattern could be produced for frequently used analysis technique types.

The goal **ATApplication** should describe how the technique was performed. For example, were the personnel suitably qualified or experienced in its application? Were any tools used (e.g. fault tree management tools or static code analysis tools) appropriate and correctly applied? This will allow us to further assess our confidence in the results.

Finally, the goal **ATEffect** refers to the general confidence we have in the analysis technique and its appropriateness to show the properties we desire. For example, some formal proof techniques are very useful to show functional properties but not much use for timing assessments. This information is not related to this specific component development, but is much more general (and again re-usable).

As an illustrative example, we consider a Worst Case Execution Time (WCET) report, within which timing analysis is presented for a component. The actual timings are presented as properties within a component model. Instantiated argument fragments might describe this data as follows. **AATop** becomes "Statistical testing provides us with 95% confidence of WCET 3ms". Confidence description would include in how many tests we have done. **ATQual** would be about the specific results, and show that the longest path had been exercised and why 3ms was the expected answer (for example). **ATApplication** would be expanded to describe the tools used, the accuracy of measurements etc.. Finally, **ATEffect** would describe how effective statistical testing was for showing WCET, as opposed to other methods. Note that the latter two branches of the argument may also be re-usable, across many instantiations of WCET arguments.

E. Specification Argument Fragment Pattern

This argument pattern relates to the quality of the component model and its specification. Typically there are three quality attributes related to a specification: completeness, consistency and correctness.

1) Completeness refers to whether the specification covers all needed properties.
2) Consistency refers to whether the specification is internally consistent, i.e. free of contradictions.
3) Correctness refers to whether the specification actually describes what the component does.

It was noted earlier, that modular arguments related to components would typically deal with very generic assertions about whether a component fulfils a specification. We are assuming the use of component models, that describe a number of different aspects relating to a component, e.g. ports, events. It also has Contracts. These contracts describe the component properties as a series of assume/guarantee relations. A component model, conforming to the component meta-model, should specify most aspects of the component, but may not contain all relevant properties (if they are not needed for the CBSE process). Therefore, this pattern refers to the component specification.

The pattern is shown in Figure 7 and has been designed to be simple and flexible. The top-level goal, **PropTop**, has three pieces of information to be instantiated: the specification, the component, and a definition of sufficient. The latter refers to the quality of the specification and may be difficult to define. If we are developing a component to conform with a particular standard there may be definitions of sufficiency relating to the type of specification (e.g. do we need to include all functions, all failure response for out of range data, all timings) etc.. It is up to the developer of a component to decide what is most appropriate here.

The remainder of the pattern is based on very generic repetitions of different reviews of the specification (instantiated for each review type, as indicated by the arrow). The review may be manual, use a static analysis tool or involve a formal proof. In each case, different sets of properties may be being reviewed, depending on their type (e.g. function or non-functional). Note that a further argument fragment about the quality of review, similar to the confidence argument), could also be provided.

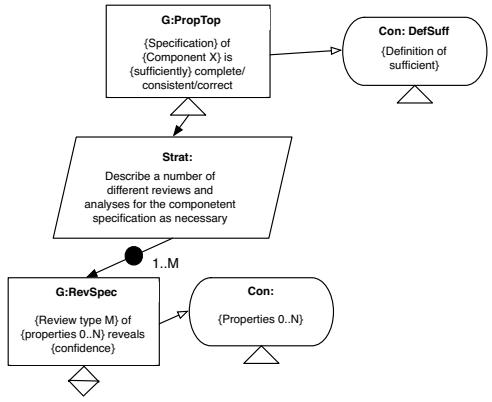

Fig. 7. **Specification Argument Pattern**

F. Proven in Use Argument Fragment Pattern

Another aspect to component argumentation is to present "proven in use" evidence. When a component has been used in other systems before, evidence about its operation may have been gathered, strengthening the initial argument/evidence about the component. The difficulty with this type of argument is that it is difficult to make it outside of a specific system or context. For example, if we have evidence of a hardware components' likelihood of physical failure from operational data, we need to demonstrate that the physical environment was sufficiently similar to the current environment for the evidence to be comparable and relevant. This means that it may not be possible to re-use the arguments or the evidence. However, there is still a common form for this type of argument which we have captured in the pattern in Figure 8.

The goal **PIUTop** has three pieces of information to be instantiated: the component being referred to, the properties about the component which are examined, and the systems which it has been used in. Descriptions of these systems should be presented as contexts in the instantiated argument. The main thrust of the argument is two pronged: on one hand each property (or sets of properties) is examined, alongside their evidence. We can reuse the analysis technique pattern from section 1) for this. The arguments would be repeated for multiple properties, or sets of properties. This division of properties is at the discretion of the developer.

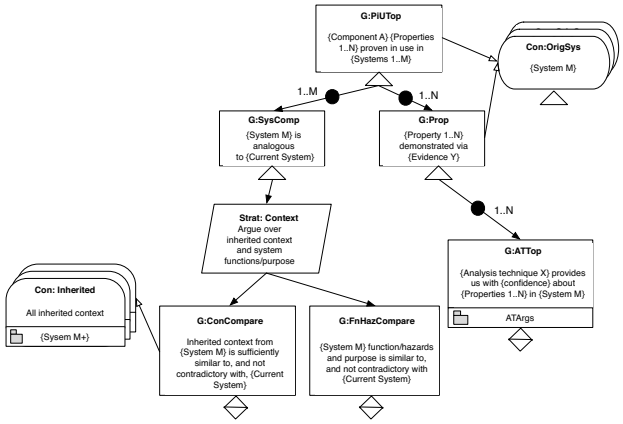

Fig. 8. **Proven In Use Argument Fragment**

The other main strand to the argument is to examine and compare the contexts in which the evidence was gathered (goals **ConCompare** and **FnHazCompare**). As there are lots of different pieces of contextual information we need to ensure that it is compatible across the whole range. Some contexts may be compatible (e.g. processor used), but other orthogonal ones incompatible (e.g. operating temperature range is different). In addition, we need to consider whether the failure conditions, and hazards being addressed were the same. This is, in some ways, a more specialised version of context compatibility, but it is the most pertinent one to safety critical system development, hence we have specifically brought it out. This strand should be repeated for all systems where the proven in use evidence has been identified.

V. System Level Argument and Contracts

Once we have our component level arguments, we need a system level argument framework in which they can be used with minimal re-work and change. An important issue is traceability of properties to hazards as well as to evidence. As described earlier, the system level safety argument will link to the component argument fragments and should address the following:

1) Hazard analysis and risk assessment.
2) Traceability from high-level safety requirements to low-level properties of the components within the system.

We note that there is not optimal safety argument structure in which all properties can be made explicit (the argument may be directed per analysis, per component or per hazard, making one more explicit than the other). Merging these approaches for CBSE, the most pragmatic solution (which supports isolated development of component argument fragments for each component) is to have some top town analysis driven argument, supported by arguments about contributions from each component. A typical top-level argument for this is shown in Figure 9. The argument is first broken down over each hazard, and then over each safety requirement which addresses the hazards. Component properties which address each safety requirement are then to be matched, via safety argument contracts as discussed at the end of this section. Thus we have the relationships "Hazard is addressed by Safety Requirements", "Safety Requirements are met by Component Properties" . There are a number of elements to be instantiated, as per the other patterns. In the top-level goal the system itself needs to be described, as does the meaning of "acceptably safe". Typically, the latter is given in terms of overall system failure probability, or within the guidance of a particular standard. The rest of the argument relies on instantiation for each hazard and for each safety requirement.

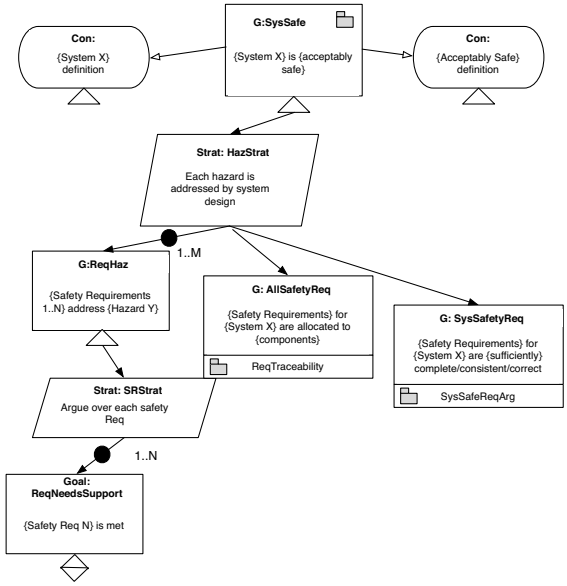

Fig. 9. **System Level Argument Pattern**

There are some caveats and drawbacks to this approach. First, there may be a large number of hazards, and a large number of safety requirements which may need to be collected together and grouped. Some discussion of methods to do are discussed by Hawkins in [15], however it does rely on a certain amount of engineering judgment on the part developer and there is no fixed guidance on how to group relevant requirements. Similarly, the properties

relating to a particular safety requirement may overlap, and the argument fragments, as constructed may not provide a direct mapping (see Figure 10). This paper doesn't provide any explicit guidance on this issue either, and it is an area for future development. We do note, however, that the CBSE and modelling aspects of the development process should be able to provide a more formal, and direct, process for matching component functionality to system requirements, and should be able to give some idea of the exact matches. An exact match within the argument itself is not actually required, as long as we have confidence that all the relevant properties are actually there as required. For example, suppose that component A has 10 grouped properties, 3 of which are needed to meet a safety requirement. Further, suppose that component B has 15 grouped properties, 2 of which are needed for the same safety requirement. The evidence for both sets of properties is via analysis, which is argued about for the whole group (per component). It doesn't matter that the argument doesn't directly show the individual traceability (it would be unreadable if it did), as long as we have assurance that the properties meeting the requirement are as described. We do need some traceability recorded at some point though, as shown in the goal **AllSafetyReq** (figure 9), with supporting evidence such as documentation or use of a requirements tool.

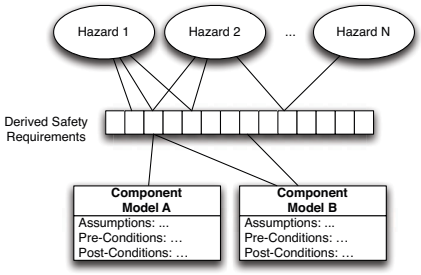

Fig. 10. **Relationship of Components, SRs and Hazards**

Finally, for completeness with respect to our overall process, we may also need to create a safety argument contract to link a safety requirement to a component property, and can use the general pattern proposed in [8]. The top level goal for a safety argument contract links to the bottom level goal in Figure 9 (**ReqNeedsSupport**). It requires instantiation for each safety requirement or possibly group of safety requirements (noting the comments in the previous paragraph). The strategy proposed is that all contributions to a safety requirement (assumed to be of the form of failures, as any condition leading to a system level hazard will arise from a failure by definition) are examined. These are separated as noted in figure 1, such that we look for causes, effects and mitigations. As discussed in [8] the inherited contexts for each component which meets the safety requirement must be assessed for compatibility and contradiction.

VI. Conclusions

This paper has presented an approach to safety arguing for a CBSE approach, with the aim of supporting the reuse of certification data and helping assess compatibility of data and highlight design issues. This is based around the production of a number of argument fragments per component, which can be re-used and joined to a hazard directed system level safety argument. We have presented a number of fragment templates, designed to give a consistent structure to the arguments, and provide guidance as to the argument content. We have further presented an outline for a system level argument, and a template for SACs joining the fragments to the system level argument.

Future work will develop methods for grouping safety requirements into appropriate groups, such as related to timing or a specific hazard. Also, we will consider the ability to re-use arguments cross-domain (e.g. from railway to avionics). Although the patterns are generic, once instantiated they will contain domain specific data. We need to expose or identify data which is context sensitive to help highlight issues when moving between domains.

Acknowledgement

We thanks the Swedish Foundation for Strategic Research (SSF) SYNOPSIS Project and the EU Artemis funded pSafeCer project for supporting this work.

References

[1] J. Fenn, R. Hawkins, P. Williams, T. Kelly, M. Banner and Y Oakshott, "The who, where, how, why and when of modular and incremental certification," in *2nd IET International Conference on System Safety*. IET, November 2007.
[2] P. Conmy, "Safety analysis of computer resource management software," Ph.D. dissertation, University of York, 2005.
[3] J. Lions. Flight 501 failure: Report by the inquiry board. [Online]. Available: http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html
[4] *Design Guidance and Certification Considerations*, 1st ed., RTCA, EUROCAE, 2007.
[5] *ISO 26262 Road Vehicles - Functional Safety*, ISO, 2011.
[6] Automotive open system architecture. [Online]. Available: http://www.autosar.org
[7] Philippa Conmy, Iain Bate, "Safe composition of real time software," in *Proceedings of the 9th IEEE Symposium in High Assurance Systems Engineering*, 2005.
[8] J. Fenn, R. Hawkins, T. Kelly, P. Williams, "Safety case composition using contracts - refinements based on feedback from an industrial case study," in *Proceedings of 15th Safety Critical Systems Symposium (SSS'07)*. Springer, February 2007.
[9] Bastian Zimmer, Susanne Bürklen, Michael Knoop, Jens Höfflinger, Mario Trapp, "Vertical safety interfaces - improving the efficiency of modular certification," *Computer Safety, Reliability, and Security, Lecture Notes in Computer Science*, vol. 6894, pp. 29–42, 2011.
[10] *ISO/IEC 19507 - Object Constraint Language*, ISO, 2012.
[11] I. Habli, "Model-based assurance of safety-critical product lines," Ph.D. dissertation, University of York, 2009.
[12] N. Leveson, *Safeware*. Addison-Wesley, 1995.
[13] Origin Consulting on behalf of Contributors, "GSN community standard."
[14] P. Graydon, I. Habli, R. Hawkins, T. Kelly, J. Knight, "Arguing conformance," *IEE Sofware*, vol. 29, no. 3, 2012.
[15] R. Hawkins and T. Kelly, "A software safety argument pattern catalogue," University of York, Tech. Rep., 2011. [Online]. Available: http://www-users.cs.york.ac.uk/~rhawkins/papers/swsafetyargumentpatterncatalogue.pdf