Extending Argumentation to Goal-Oriented Requirements Engineering

Ibrahim Habli, Weihang Wu, Katrina Attwood, and Tim Kelly

Department of Computer Science, The University of York, York YO10 5DD
{Ibrahim.Habli,Weihang.Wu,Katrina.Attwood,
Tim.Kelly}@cs.york.ac.uk

Abstract. A key goal in safety-critical system development is to provide assurance that the critical requirements are sufficiently addressed. This goal is typically refined into three sub-goals, namely that the safety requirements are validated, satisfied and traceable. The achievement of these sub-goals is typically communicated by means of a safety argument supported by items of evidence (e.g. testing, review or analysis). In this paper, we explore the relationships between goals, requirements, and arguments. We discuss how argumentation is used to assure the decomposition and traceability of requirements in safety-critical applications. Particularly, we focus on the achievement of goals related to both the requirements artefacts and the underlying requirements process.

1 Introduction

Goal-modelling techniques have long been recognised as an effective support to requirements engineering processes. The work of van Lamsweerde and others [1, 2], for example, has demonstrated a powerful goal-based method to support requirements elicitation, refinement, trade-off analysis and documentation. A goal describes the objective that a system should meet, which may be classified further in terms of functional and quality goals. Goals can be refined hierarchically into sub-goals through various refinement strategies such as AND/OR decomposition, design decisions and domain-specific analysis results. The goal refinement process stops when all identified goals have been satisfied or achieved. Goals may be violated, however, due to unexpected behaviours of a system or its environment. The notion of obstacles [3] or anti-goals [4] has recently been introduced in order to integrate potential violations into goal modelling.

Goal-based techniques have also been used to model the relationships between the system under definition and the environment in which it will operate. For example, the i* technique has been used in early-stage requirements engineering, to capture aspects of user motivation, business and organisational goals in systems characterised by a high degree of human-computer interaction, such as Air Traffic Control [5, 6]. Chung et al [7] have also developed a goal-based framework for clarifying and prioritising non-functional requirements and managing trade-offs between them. At the design level, early work on argument-based design rationale developed a set of generic models of design processes in terms of three common elements [8]: issues/questions, positions/options, and arguments/criteria. There have been several applications of design rationale to software engineering. The richest extension to date is the REMAP (REpresentation and MAintenance of Process knowledge) model developed by Ramesh [9], in which the notions of requirements/goals, assumptions, constraints and design objects have been incorporated into the Issue-Based Information System (IBIS) framework.

In the safety-critical system domain, one of the key goals is to provide assurance that the critical requirements are sufficiently addressed. As a minimum, it should be demonstrated that these requirements are validated, satisfied and traceable. Particularly, software standards in the safety domain are shifting towards goal-based approaches where the validation, satisfaction and traceability of safety requirements are the primary goals for demonstrating that a safety-critical software system is acceptably safe. To justify that a software system is acceptably safe, a safety case is typically submitted. A safety case is defined in the UK Defence Standard 00-56 as [10]:

> *"A structured argument, supported by a body of evidence that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given operating environment."*

Underlying the descriptions of the safety case is a view of the safety case consisting of three principal elements: Goals, Argument and Evidence [11]. The argument communicates the relationship between the evidence and goals. Argument without supporting evidence is unfounded, and therefore unconvincing. Evidence without argument is unexplained – it can be unclear that (or how) goals have been satisfied.

In this paper, we explore the relationships between goals, requirements, and arguments based on our experience in the software safety domain. We discuss how argumentation is used to assure the decomposition and traceability of software requirements in safety-critical applications. In particular, we focus on the achievement of goals related to both the *requirements artefacts* and the underlying *requirements engineering process*. To create goal-based arguments, we use the Goal Structuring Notation (GSN) [11]. GSN is a graphical notation for the construction of safety and assurance arguments. Nonetheless, GSN is generic and can be used to structure and present goal-based arguments that demonstrate requirements validation and satisfaction.

The rest of paper is structured as follows. Section 2 presents an overview of GSN and how it can be used to demonstrate how existing requirements are decomposed and managed. Section 3.1 presents an argumentation approach to refining safety goals and anti-goals. 3.2 shifts the discussion towards the assurance of goals related to the underlying requirements process. Section 3.3 focuses on one process aspect, namely requirements traceability. Finally, Section 4 presents a summary and conclusions.

2 The Goal Structuring Notation (GSN)

GSN explicitly represents the individual elements of goal-based arguments (requirements, goals, evidence and context) and (perhaps more significantly) the relationships that exist between these elements (i.e. how individual requirements are supported by specific claims, how claims are supported by evidence and the assumed context that is defined for the argument). The principal symbols of the notation are shown in Fig. 1 (with example instances of each concept).

When the elements of the GSN are linked together in a network they are described as a ‘goal structure’. The principal purpose of any goal structure is to show how goals (claims about the system) are successively broken down into sub-goals until a point is reached where claims can be supported by direct reference to available evidence (solutions). As part of this decomposition, using the GSN it is also possible to make clear the argument strategies adopted (e.g. adopting a quantitative or qualitative approach), the rationale for the approach and the context in which goals are stated (e.g. the system scope or the assumed operational role).

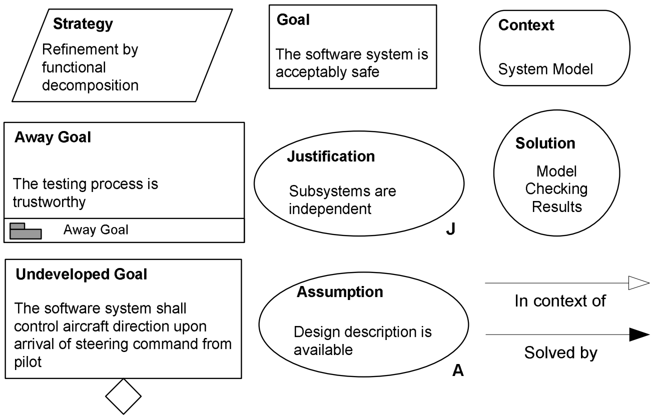
Strategy
Refinement by functional decomposition
Goal
The software system is acceptably safe
Context
System Model
Away Goal
The testing process is trustworthy
Away Goal
Justification
Subsystems are independent
J
Solution
Model Checking Results
Undeveloped Goal
The software system shall control aircraft direction upon arrival of steering command from pilot
Assumption
Design description is available
A
In context of
Solved by

Fig. 1. Elements of the Goal Structuring Notation

The GSN technique adds a rich semantics to this goal-breakdown structure, by forcing the explicit recording of and justification for the strategies used to refine and relate the goals in a semi-formal argument structure. Since both the goals and these refinement strategies hold true only in a defined context (i.e. within the scope of a particular system, or under certain operational conditions), the notation's syntax requires that explicit reference is made to the evidential basis on which the goals and the refinement depend. Within Europe, GSN has been adopted by a growing number of companies within safety-critical industries (such as aerospace, railways and defence) for the presentation of safety arguments within safety cases.

Arguably, GSN can be used in a broader and more general context. It can provide a means for decomposing requirements and recording traceability links between individual claims and sub-claims. These claims are represented as goals, and equate to the requirements statements and specifications. The notation also records the strategies used to decompose the goals. These strategies map to the ‘satisfaction argument’, in that they seek to provide a basis for the relationship between the goals. As well as presenting a clear record of the goal-decomposition strategy, however, GSN allows this strategy to be validated by the use of an apparatus of justifications and assumptions. These justifications and assumptions are attached to goals and strategies within the structure, as well as by explicit references to artefacts such as system architectural models or contextual information.

3 Modelling Requirements Artefacts and Processes Using GSN

This section presents GSN arguments addressing (1) the refinement of safety goals and anti-goals and (2) the assurance of goals related to the underlying requirements process, with a particular emphasis on requirements traceability.

3.1 Modelling Goals and Anti-goals

Although several different goal formulation techniques exist, it is possible to identify four common elements in the 'goal' concept:

- Artefact: The artefact is the composite system or its parts onto which a goal is applied.
- Context: The context addresses the pre-conditions that a goal refers to and evolves over.
- Stimulus: The stimulus is the trigger condition for the initiation of a goal.
- Response: The response captures the desired properties (i.e., postconditions) that the artefact should hold over time. Quality requirements (e.g., deadline or failure rate) can be specified in this part if they exist.

A GSN goal represents a requirement goal, and can thus be expressed using natural language in the following form:

"The <artefact> shall <respond> upon <stimulus> when <context>"

This goal formulation is consistent with the SEI's quality attribute scenario framework [12]. It can be applicable to both functional and quality goals. As an example, consider a wheel braking system (WBS) on an aircraft [13]. We assume there are a number of top-level system goals that can be stated in terms of aircraft functionality e.g., controlling the aircraft on the ground and safety (Fig. 2). Despite their high level of abstraction, these system goals can be expressed in stimulus-response form. Each functional goal can be decomposed further into a set of sub-goals and these should evolve separately given that they are independent. The goal decomposition may be guided by the decision to use various mechanisms for speed reduction. Goal structures can thus be constructed. Safety goals cannot simply be decomposed via functional goals or system structures; their refinement is based upon the results of deviation analysis and the chosen mitigation. For example, in the WBS, late output from a controller is likely to be safety-significant. A performance goal is thus derived and added into the safety goal structures. Fig. 2 shows a part of the goal structure in which the core functionality of WBS is elicited. All the goals in this structure are expressed using the above form (the context elements '*RefArf_SatisfactionArg*' and '*ReqDev_SWReq*' are addressed separately in the next

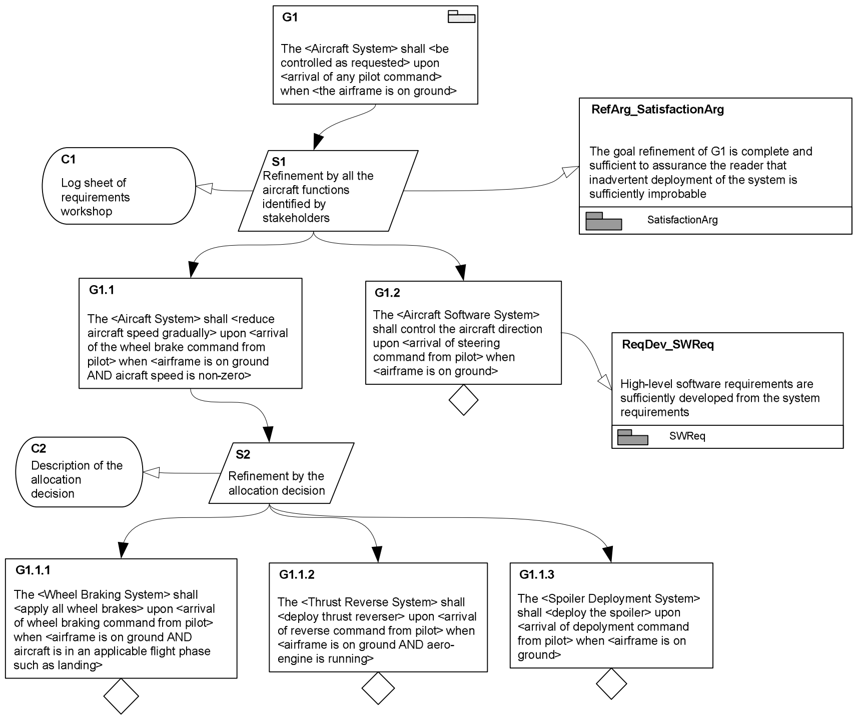
G1
The <Aircraft System> shall <be controlled as requested> upon <arrival of any pilot command> when <the airframe is on ground>
RefArg_SatisfactionArg
The goal refinement of G1 is complete and sufficient to assurance the reader that inadvertent deployment of the system is sufficiently improbable
SatisfactionArg
C1
Log sheet of requirements workshop
S1
Refinement by all the aircraft functions identified by stakeholders
G1.1
The <Aircaft System> shall <reduce aircraft speed gradually> upon <arrival of the wheel brake command from pilot> when <airframe is on ground AND aicraft speed is non-zero>
G1.2
The <Aircraft Software System> shall control the aircraft direction upon <arrival of steering command from pilot> when <airframe is on ground>
ReqDev_SWReq
High-level software requirements are sufficiently developed from the system requirements
SWReq
C2
Description of the allocation decision
S2
Refinement by the allocation decision
G1.1.1
The <Wheel Braking System> shall <apply all wheel brakes> upon <arrival of wheel braking command from pilot> when <airframe is on ground AND aircraft is in an applicable flight phase such as landing>
G1.1.2
The <Thrust Reverse System> shall <deploy thrust reverser> upon <arrival of reverse command from pilot> when <airframe is on ground AND aero-engine is running>
G1.1.3
The <Spoiler Deployment System> shall <deploy the spoiler> upon <arrival of depolyment command from pilot> when <airframe is on ground>

Fig. 2. A top-level goal structure for the WBS example

two sections). The expression language used is a structured natural language, and some expression can be very abstract at this level. For example, both the stimulus and response parts of the top-level goal G1 are very general and need to be refined. This should be acceptable, however, in the early development lifecycle in which many requirements are volatile and unclear.

On the other hand, an anti-goal is the negative correlative of a goal: a condition that, if true, would immediately prevent the system from achieving the corresponding goal. Goals and anti-goals are complementary and thus capture the possible desired and undesired end states of a composite system. A common example of an anti-goal is the loss of a system function where the function is a goal. Nevertheless, the simple negation of a goal in terms of propositional logic cannot guarantee the sufficient completeness of the corresponding anti-goals. A less obvious but perhaps more severe anti-goal would be inadvertent application of that function. Given some goal formulation, it is important to ensure the exhaustiveness of deviations from that goal, at least from the viewpoint of safety. In the safety community, the possible deviations of a system are often characterised in terms of deviation or failure modes. Previous work in York has developed a collection of deviation modes for software systems: SHARD guidewords [14]. We interpret the SHARD modes with respect to the goal formulation in the following Table 1.

By allocating the SHARD modes onto a formulated goal and interpreting them using the above table, we can achieve a high level of confidence in the exhaustiveness of the set of anti-goals elicited. Note that anti-goals do not necessarily have safety implications, although they should always be evaluated with respect to possible safety-related consequences. Let us return to the WBS example. As soon as the system goals of WBS are formulated, the identification of anti-goals can start by considering the SHARD deviations first without information about the elaborated scenarios. In this example, only omission and commission modes are applicable. Table 2 illustrates an anti-goal by negating the context part – wheel braking when the context is not as intended. The definition of the stimulus part is trivial in this case. The anti-goal elicited is expressed at an abstract level.

Table 1. The anti-goal interpretation using SHARD guidewords

SHARD	**Anti-Goal Interpretation**
Omission	Response part does not hold while stimulus and environment parts hold
Commission	Stimulus or context parts do not hold while response part holds
Timing	Timing constraint specified in the response part is violated while the other parts hold
Value	Value constraint specified in the response part (e.g., accuracy or cost) is violated while the other parts hold

Table 2. An example anti-goal formulation

Portion of Goal	**Possible Value**
Artefact	WBS
Context	NOT (Airframe is on ground AND aircraft is in landing/taxiing/RTO flight phase)
Stimulus	N/A
Response	All wheel brakes are applied

By expanding the negation operation on the context part using Boolean logic, we can derive a set of well-refined anti-goals: e.g., wheel brakes applied when the aircraft is taking off or when the aircraft is in air (both of these conditions can lead, at worst, to total loss of control). It must be stressed that the expansion here cannot be achieved solely by formal Boolean logic and may need the help of domain experts. For the example of inadvertent wheel braking when the aircraft is taking off, we may need to distinguish further whether the aircraft is taking off before the decision speed V1, as the safety consequences before and after this threshold is passed would be different. Obviously, it would be impossible to identify these two anti-goals by the use of formal logic alone, without the requisite domain knowledge.

When all anti-goals are identified and refined (say, eight anti-goals for the WBS example), they should be linked to the anti-goals of the parent goal of the WBS (i.e., aircraft deceleration) in a bottom up manner, thereby forming an anti-goal structure. The anti-goal structure in the WBS example is shown in Fig. 3. The anti-goal structure should refer to the corresponding functional goal structure. It should be noted that the expression languages used in functional goals, safety goals and anti-goals are slightly different. The functional goals are simply operational and thus

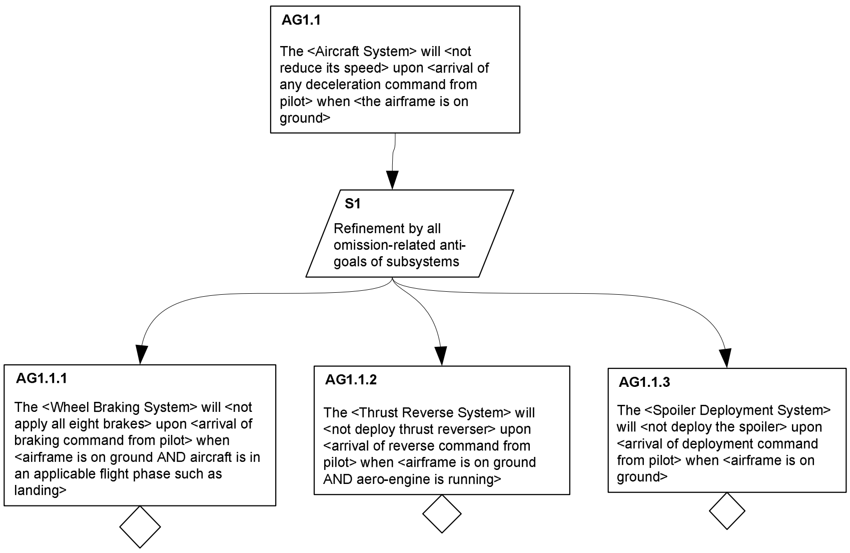
AG1.1
The <Aircraft System> will <not reduce its speed> upon <arrival of any deceleration command from pilot> when <the airframe is on ground>
S1
Refinement by all omission-related anti-goals of subsystems
AG1.1.1
The <Wheel Braking System> will <not apply all eight brakes> upon <arrival of braking command from pilot> when <airframe is on ground AND aircraft is in an applicable flight phase such as landing>
AG1.1.2
The <Thrust Reverse System> will <not deploy thrust reverser> upon <arrival of reverse command from pilot> when <airframe is on ground AND aero-engine is running>
AG1.1.3
The <Spoiler Deployment System> will <not deploy the spoiler> upon <arrival of deployment command from pilot> when <airframe is on ground>

Fig. 3. The anti-goal structure for the WBS example

'shall' statements are suitable; the anti-goals are hypotheses about states and events of the system and thus 'will' statements should be used. The construction of anti-goal structures will prompt the refinement of the goal structure in which decisions need to be made regarding the mitigation of these anti-goals.

3.2 Assuring Goals Concerning the Requirements Process

The previous section has presented an argument supporting the decomposition of requirements goals in isolation from the underlying requirements process, i.e. the level of review, independence, traceability, competency of the requirements analysts and organisational agility. Uncertainties about the trustworthiness of this process may weaken confidence in the allocated and refined requirements. To this end, it is important to satisfy goals about the requirements *process* (i.e. not just the requirements *artefacts*). These goals address attributes such as process completeness, consistency and robustness since flaws in the requirements process may result in flaws in the requirements themselves. Such types of requirements flaws may only be discovered by independent reviews, for example. Factors that need to be addressed by an argument justifying the requirements process include issues such as:

- Is the software requirements team independent from the systems requirements team?
- Is requirements validation carried out using a repeatable and traceable technique?
- Is requirements validation performed on a stable and identifiable version of the allocated system requirements?
- Do the software engineers understand their relationship with the system requirements?

Fig. 4 shows a purely process-based argument that provides evidence addressing goals about the requirements process of the "Aircraft Software System" (G1.2 in Fig. 2). The process goals considered in that argument concern the clarity of the notation, the suitability of the validation methods, the consistency of the configuration, and the competency of the software engineers. Research has shown that domain knowledge is one of the most significant factors in achieving low rates for hazardous failure in safety-critical software development [15]. Validation of the requirements – and the consequent discovery of operational requirements errors – is best performed by domain experts. Therefore, regardless of how well the software requirements are structured, refined and documented, it is of equal importance to show the competency of personnel and it is therefore important that it is included as a separate goal in the process assurance argument. The next section focuses on assuring requirements traceability, a process goal central to all software safety standards.

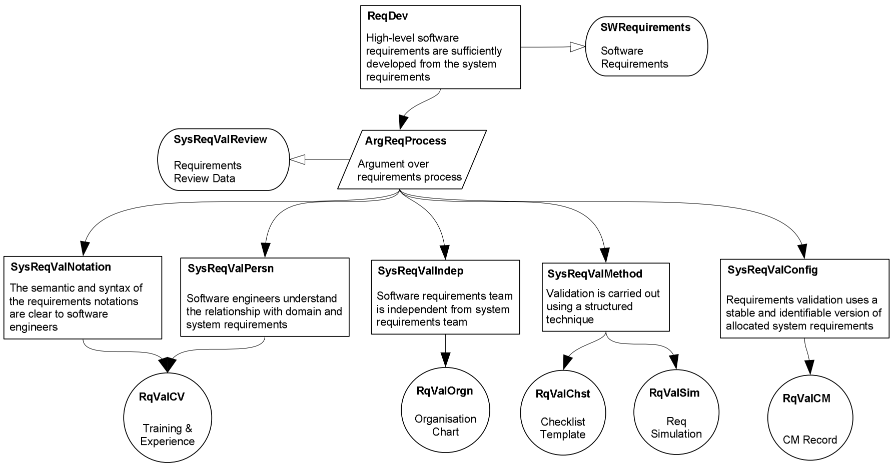
ReqDev
High-level software requirements are sufficiently developed from the system requirements
SWRequirements
Software Requirements
SysReqValReview
Requirements Review Data
ArgReqProcess
Argument over requirements process
SysReqValNotation
The semantic and syntax of the requirements notations are clear to software engineers
SysReqValPersn
Software engineers understand the relationship with domain and system requirements
SysReqValIndep
Software requirements team is independent from system requirements team
SysReqValMethod
Validation is carried out using a structured technique
SysReqValConfig
Requirements validation uses a stable and identifiable version of allocated system requirements
RqValCV
Training & Experience
RqValOrgn
Organisation Chart
RqValChst
Checklist Template
RqValSim
Req Simulation
RqValCM
CM Record

Fig. 4. Software Requirements Validation Argument

3.3 Goal-Based Arguments for Requirements Traceability

Zave and Jackson [16] observe that satisfaction of a requirement (R) can be demonstrated only by a sufficient combination of domain knowledge (K) and specifications (S): S, K $\vdash$ R. Jackson suggests that traceability links between requirements and specifications should be supported by textual 'correctness arguments' which explain how the specifications and domain behaviour combine to provide assurance that the engineered system satisfies the requirement in the application domain [17]. 'Satisfaction arguments' are a development from these 'correctness arguments' and provide assurance that the traceability relationship between requirements at different levels of abstraction is valid within a given application domain [18]. For example, it is essential for the successful reuse of requirements across system families that there is some assurance that the satisfaction relationships remain valid for requirements and specifications in the reuse domain. Failure to observe this principle risks the late, and therefore costly, delivery of erroneous or untenable requirements.

The following example demonstrates how GSN can be used to document satisfaction arguments, and to indicate where changes in design commitments or contexts challenge the satisfaction of the requirements. The decomposition in Fig. 2 does not attempt to justify the decomposition strategy employed. Instead, it contains an ‘away goal’ reference to a justification claim depicted in Fig. 5 (*Satisfaction Argument*). This goal is the top-level claim of the argument in Fig. 5, which is the satisfaction argument justifying the requirements decomposition. The argument strategy is a two-pronged one: the left-hand side of the goal structure argues that the checks, taken together, are sufficient to satisfy the top-level requirement, while the right-hand side (not fully developed here) argues that all possible failure modes have been considered and are adequately mitigated by the checks. The GSN structure makes clear what evidence is required to demonstrate the satisfaction of the top-level requirement (PSSA).

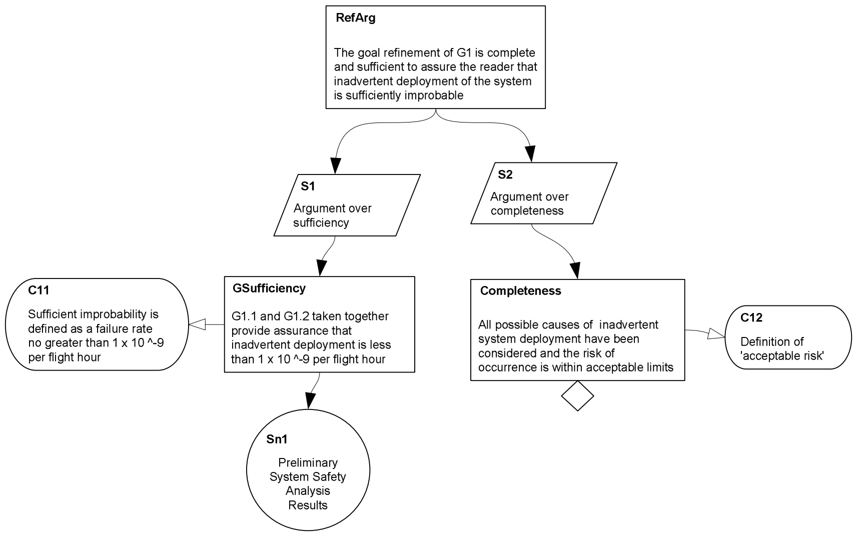
RefArg
The goal refinement of G1 is complete and sufficient to assure the reader that inadvertent deployment of the system is sufficiently improbable
S1
Argument over sufficiency
S2
Argument over completeness
C11
Sufficient improbability is defined as a failure rate no greater than 1 x 10 ^-9 per flight hour
GSufficiency
G1.1 and G1.2 taken together provide assurance that inadvertent deployment is less than 1 x 10 ^-9 per flight hour
Completeness
All possible causes of inadvertent system deployment have been considered and the risk of occurrence is within acceptable limits
C12
Definition of 'acceptable risk'
Sn1
Preliminary System Safety Analysis Results

Fig. 5. Satisfaction argument for the refinement of the G1 in Fig. 2

The GSN structure provides a straightforward means for assessing which aspects of the satisfaction relationship are threatened by the design change. GSN satisfaction arguments thus allow for the clear record of domain information and assumptions, and indicate which information sources are required for adequate requirements traceability. The severity of the impact of requirements or contextual change can be assessed by reference to the satisfaction arguments on the requirements decomposition.

5 Summary and Conclusions

In this paper we have discussed how goal-based argumentation can be used to provide assurance for the decomposition and traceability of safety-critical requirements. We have presented an integrated approach that addresses the achievement of goals related to both the requirements artefacts and the underlying requirement process. The key benefit of adopting GSN is that it improves comprehension of the requirements satisfaction and validation argument amongst all of the key project stakeholders (i.e. system developers, safety engineers, independent assessors and certification authorities). In turn, this improves the quality of the debate and discussion amongst the stakeholders and reduces the time taken to reach agreement on the development and analysis approaches being adopted.

References

1. Lamsweerde, A.v., Dardenne, A., Fickas, S.: Goal-directed Requirements Acquisition. Science of Computer Programming 20, 3–50 (1993)
2. Lamsweerde, A.v.: Goal-Oriented Requirements Engineering: A Guided Tour. In: RE 2001. Proceedings of 5th IEEE International Symposium on Requirements Engineering, pp. 249–263. IEEE Computer Society Press, Los Alamitos (2001)
3. Lamsweerde, A.v., Letier, E.: Integrating Obstacles in Goal-Driven Requirements Engineering. In: Proceedings of the 20th International Conference on Software Engineering, pp. 53–62. IEEE Computer Society Press / ACM Press (1998)
4. Lamsweerde, A.v.: Elaborating Security Requirements by Construction of Intentional Anti-Models. In: Proceedings of the 26th International Conference on Software Engineering, pp. 148–157. IEEE Computer Society Press, Los Alamitos (2004)
5. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In: RE 1997. Proceedings of the Third IEEE International Symposium on Requirements Engineering, Washington D.C., USA, Jan 6-8, 1997, pp. 226–235. IEEE Computer Society Press, Los Alamitos (1997)
6. Maiden, N., Jones, S.: Dependability in RESCUE: A Concurrent Engineering Approach to the Specification of Requirements for Air Traffic Management. In: DSN2004. Proceedings of the Workshop on Interdisciplinary Approaches to Achieving and Analysing System Dependability, Washington D.C., USA, June 29, 2004 (2004)
7. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic, Boston, etc. (1999)
8. Shum, S.B.: Design Argumentation as Design Rationale. In: The Encyclopedia of Computer Science and Technology, Marcel Dekker Inc., New York, pp. 95–128 (1996)
9. Ramesh, B., Dhar, V.: Supporting systems development by capturing deliberations during requirements engineering. IEEE Trans. on Software Engineering 18(6), 498–510
10. UK Ministry of Defence, 00-56 Safety Management Requirements for Defence Systems, Part 1: Requirements, Issue 3, UK Ministry of Defence (August 2004)
11. Kelly, T.P.: Arguing Safety - A Systematic Approach to Safety Case Management. DPhil Thesis, University of York, York (1999)
12. Barbacci, M., Ellison, R., Lattanze, A., Stafford, J., Weinstock, C., Wood, W.: Quality Attribute Workshops (QAWs), Third Edition. Technical Report (CMU/SEI-2003-TR-016) Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University (2003)

13. ARP 4761: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment, Society of Automotive Engineers, Inc. (1996)
14. Fenelon, P., McDermid, J., Nicholson, M., Pumfrey, D.: Towards Integrated Safety Analysis and Design. ACM Computing Reviews 2(1), 21–32
15. McDermid, J.A.: Software Safety: Where's The Evidence? In: Proceedings of the Sixth Australian Workshop on Industrial Experience with Safety Critical Systems and Software, Australian Computer Society (2001)
16. Zave, P., Jackson, M.: Four Dark Corners of Requirements Engineering. ACM Transactions on Software Engineering and Methodology 6(1) (1997)
17. Jackson, M.: Problem Frames: Analysing and Structuring Software Development Problems. Addison-Wesley, London (2001)
18. Hull, M., Jackson, K., Dick, J.: Requirements Engineering. Springer, London (2002)