A model-based assurance case construction approach for system control software

Dajian Zhang*, Minyan Lu, Nan Wu

School of Reliability and System Engineering, Beihang University, Beijing, P. R. China

Received 12 June 2014, www.tsi.lv

Abstract

As the massive damage caused by the failures of system control software becomes increasingly prominent, people pay more attention to the construction of assurance case to demonstrate the dependability level of system control software. In this paper, a new assurance case construction approach for system control software is proposed. Based on the metamodel of modular GSN, we give the basic procedure and tree structure deductive algorithm of the approach, and verify our work using Brake Control software used in an aircraft. The results show that the approach can develop assurance case effectively and efficiently.

Keywords: Software, Dependability, assurance case, GSN, modularization

1 Introduction

With the wide deployment of software in critical control systems whose potential failure may cause huge damage, the dependability of the control software has become a major factor for proper system operation. Therefore, it is of great importance to study on the dependability assurance of this software. Demonstrating the expected dependabiltiy properties of this software to provide sufficient confidence for potential users is a key issue [1]. Traditional software development and certification are generally based on a prescriptive standards, such as DO-178B [2], IEC 15608 [3], ISO/IEC 15408 [4-6]. However, these certification approaches have some deficiencies such as unclear rationale underlying some process activities, lack of organization between evidence, highly prescribed technical activities [7-8]. Therefore, a new goal-basded assurance case approach is proposed [9]. Through a clear argument structure and flexible and effective organization of evidence, it can demonstrate the system meet its original requirements in an explict and structured way. This approach can overcome the deficiencies of traditional approaches and receives growing attention [10].

Assurance case is originally used in the safety area [11], and gradually extended to other dependability area [12-14]. It is defined as “a documented body of evidence to provide a compelling justification that the systems performing a specific task satisfies specified critical properties in a specific environment” [15]. An assurance case generally consists of three elements: claim, argument, and evidence. How to represent the structure of argument efficiently and concisely is a key problem in assurance case research area. Many approached are proposed [16-17]. Goal Structuring Notation (GSN) is a popular one among these approaches [18]. It combines rich graphical notations with modularization thinking to present argument in an intuitive, explicit way. It can clearly exhibit the logical relationship between product-oriented and process-oriented evidence by establishing the argument structure model and can be used in qualitative or quantitative analysis to achieve the evaluation of software dependability level [19]. However, the rich elements in GSN also bring confusion in its usage. Because of a lack of guidance on how to use this powerful tool systematicly and unambiguously, the modular elements are often abused or misused by developers of assurance case. Therefore, the result of the argument is strongly influenced by subjective factors, which leads to the low effectiveness and the reduction of confidence placed in the argument conclusion.

This paper proposes a structured development approach for modular GSN in order to guide the construction of assurance case for control software. Based on the analysis of the GSN modular argument elements, an extended GSN metamodel is proposed. According to the metamodel, we give the progression algorithm for constructing the core structure of assurance case and the standard procedure of the argument construction. We illustrate our contributions by application to a Brake Control software system. The results show that our approach can provide explicit guidance and help to standardize the development process of assurance case. It also reduces subjectivity and ambiguity in the process, and improves the effectiveness.

* ***Corresponding author*** e-mail: greatdjz@163.com

2 Model-Based Assurance Case Construction Approach

2.1 GSN MODEL MODULARIZATION

On the basis of in-depth study of GSN basic concepts, we propose a modular GSN meta-model, as is shown in figure 1-4.

Figure 1 shows the correlation of macroscopic concepts in assurance case. As is shown above, every assurance case has an assurance subject. Assurance case varies from safety assurance, security assurance, reliability assurance, dependability assurance, etc. Every assurance subject has a central assurance objective, this assurance objective is proposed in the form of claim, and is the top-level objective of assurance case. An assurance case consists of three basic parts: claim, argument and evidence. Assurance case is presented in the form of structured argument and GSN is one of these arguments.

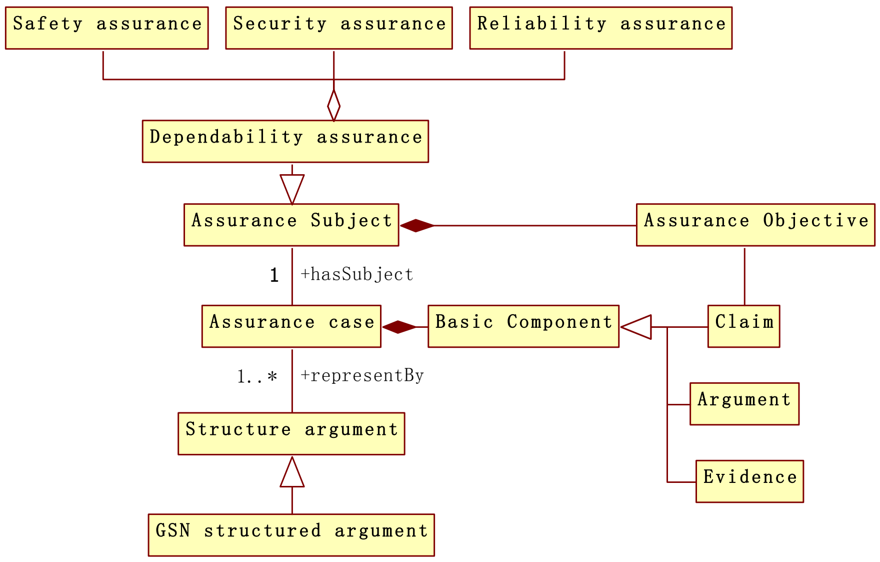

FIGURE 1 Macroscopic concept of assurance case

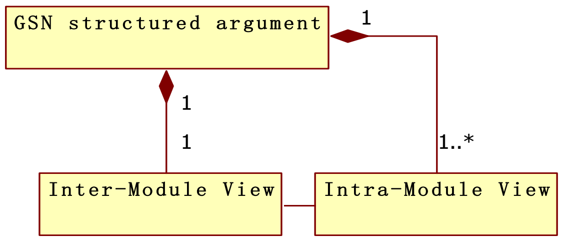

FIGURE 2 Macroscopic composition of GSN structured argument

Figure 2 shows the macro composition of GSN structured argument. GSN structured argument can be treated as two abstract granularities: the macroscopic and abstract argument between modules, and specific inter-module argument. In general, for relatively simple software systems, conducting fine-granularity argument using basic GSN nodes can meet the requirement. However, when we are arguing a complex large scale system, adopting basic argument structure will make the argument structure too big and complicated to manage, especially when a system consists of many modules. Introducing coordination mechanism of two-level abstract of module view and inter-module view can deal with this problem. Module view displays the relationship between modules, it shows the overall argument structure in a higher level of abstraction. Every module in the module view represents a specific argument structure, and an inter-module view in parallel.

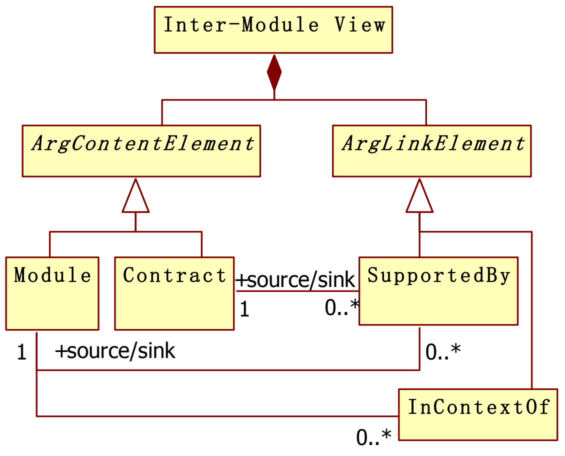

FIGURE 3 Basic concepts and correlation of module view

Figure 3 shows basic concepts and correlation of module view. Module and contract are basic elements in module view, module is a high-level abstraction of specific argument structure, a contract shows the relationship between modules and defines how a goal in the module is supported by the argument in another module. Modules and contracts are connected by “supportedby” and “incontexof” elements to build the macro view of the argument.

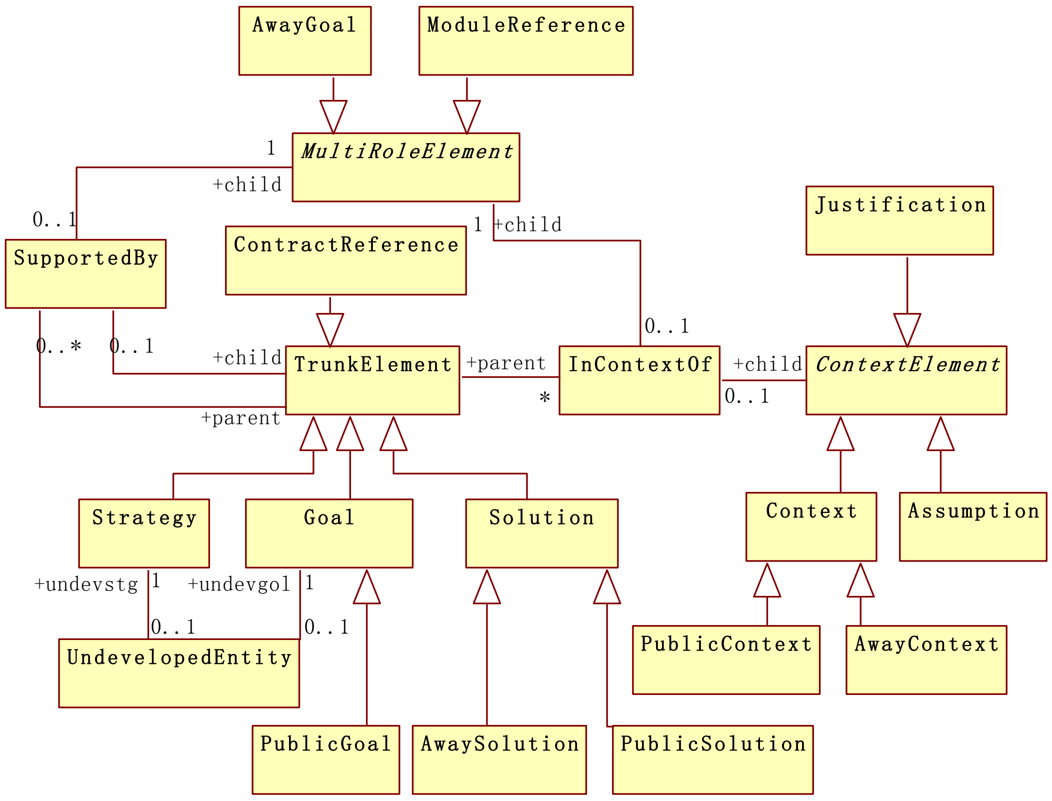

FIGURE 4 Concept and correlation of argument view

Figure 4 shows the concept and correlation of specific argument view. Elements in specific argument view can be divides to standard GSN elements and extended modularization elements. In standard GSN elements, goal, strategy and solution forms the backbone element of argument, undeveloped bodies provides effective support for the development phased in argument through the attachment to Goal and Strategy; context, assumption and justification are ancillary elements providing background information for the argument, “supportedby” and “incontexof” elements are connectors in the argument, “public elements” (public goal, public solution, public context) and “away elements” (Away solution, away the context and away goal) correspond each other, and together they provide a general mechanism for the share of inter-module argument elements. “Public elements” are open external interface of the argument, indicating these elements can be referenced by other argument modules. When referenced by other argument modules, they must be presented in the form of “away elements”. Module reference elements and contract reference elements corresponds module elements and contract elements, respectively. They can be considered as the projection of module and contract element in the specific argument view. The roles different elements play in the argument are different, contract reference element can be considered as a backbone element, “modulereference element” is more special. Similar with “away elements”, they come in a wide variety of roles in the argument, they can be referred to as the backbone elements in argument and reasoning, and they can provide background information for argument and reasoning, so in this paper we will define these as “MultiRoleElement”.

2.2 MODEL-BASED GSN ARGUMENT CONSTRUCTION APPROACH

Based on the meta-model given above, we define a Model-Based GSN Argument Construction Approach. Because the modular GSN argument includes two levels: macro and micro argument, we adopt the principle of building the argument from macro to micro, and then back to macro.

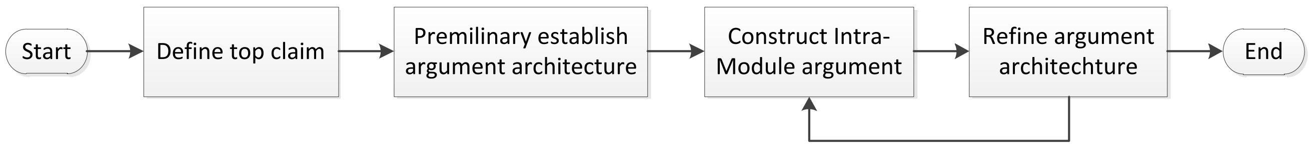
Start
Define top claim
Premilinary establish argument architecture
Construct Intra-Module argument
Refine argument architechture
End

FIGURE 5 Basic procedure of modular GSN Construction Approach

Figure 5 shows the basic procedure of modular GSN construction approach. Firstly, define the top-level claim according to our argument subject, which is the ultimate goal of our argument. Then establish the preliminary overall argument structure according to top-level claim. We can establish the argument structure from different angles according to the features of top-level claim and evidence acquired. Through the investigation and analysis of the literature on assurance case, we found some typical argument structures, such as functional arguments, workflow process arguments, life cycle arguments, system structure arguments and risk mitigation arguments, etc. Argument structure is shown in the form of module view. This is only the preliminary established macroscopic module view structures, but the specific interface between modules is difficult to specify at the moment, specific argument structure refinement is needed to establish step by step. Next, we need refine each module one by one. According to the meta-model given in the figure above and the concepts related to instantiation, we can gradually establish argument using top-down approach. Along with the unfolding of argument structure, modular elements are added to the specific structure of argument. The introduction of modular elements means that the shared interface between this module and other modules are changed. In this case, we need to backtrace the overall argument, change the module view accordingly, which is step 4. This process might involve two cases: 1. the share interface are changed between this module and an existing module, this case is likely to cause changing relations between the two modules in the module view. 2. This module has an interaction with a module excluded in the module view, in this case, we need to create a new module in the module view to reflect this change.

Establishing the specific argument view, namely step 3, is the focus of the modular GSN argument construction. The introduction of modular elements greatly enhanced the expression ability of GSN, however, the abuse of modular elements can lead to the unclear role of elements, bringing chaos in the argument view structure and further influence the macro module view. These will bring difficulties to users to understand and communicate on assurance case. This paper defines a modular development approach using formal methods. This approach can clearly demonstrate the role and the timing of use of each modular element in constructing the argument, eliminating the ambiguity on understanding, and implements a systematic construction of argument.

First, we will define some primitives to describe the process, through the combination of these primitives we can describe the establishment process of the argument. The process describing primitives are as followed:

Declare primitives:

Declare(SetInstance, ConceptName)

- SetInstance{instanceName$_1$, instanceName$_2$, …, instanceName$_n$}, instanceName$_i$ is the name of the instance;
- ConceptName is the name of the concept;

This primitive declares the instance set "SetInstance" of the concept "ConceptName"

Relationship defining primitives:

Define(SetInstance, LinkType, SourceInstance)

- SetInstance {instanceName$_1$, instanceName$_2$, …, instanceName$_n$}, instanceName$_i$ is the name of the instance;
- LinkType is enumeration type variable, enum{Supportedby, InContextof};
- SourceInstance is the name of the instance;

This primitive defines the relationship between instance set "SetInstance" and instance source "SourceInstance". If SetInstance $\notin \emptyset$, then this primitive defines the LinkType argument relationship from SourceInstance to SetInstance; if SetInstance = $\emptyset$, then this primitive does not define any relationship.

Judge primitives:

ifContextNeeded(InstanceName)
Determine if instance "InstanceName" needs background information
ifAssumptionNeeded(InstanceName)
Determine if instance "InstanceName" needs assumption information
ifJustificationNeeded(InstanceName)
Determine if instance "InstanceName" needs judge information
ifDecomposeNeed (InstanceName)
Determine if instance "InstanceName" needs to be decomposed

Based on the primitives above, we put forward a GSN modular constructing process as followed:

```
BEGIN
Declare({topGoal}, Goal)
Step_DefineContextInfo(topGoal)
Step_DecomposeGoal(topGoal)

Step_DecomposeGoal(aGoal::Goal)
  ifNeedDecompose(aGoal)
        Delare({aStrategy}, Strategy)
     Step_DefineContextInfo(aStategy)
     Define({aStrategy}, Supportedby, aGoal)
     Declare({subGoal$_1$, subGoal$_2$, …, subGoal$_n$}, Goal)
     Define({subGoal$_1$, subGoal$_2$, …, subGoal$_n$},
     Supportedby, aGoal)
        for each subGoal$_i$, i        $\in \{1,…,n\}$
          Step_DecomposeGoal(subGoal$_i$)
        Declare({awayGolname$_1$, awayGolname$_2$, …,
awayGolname$_n$}, AwayGoal)
        Define({awayGolname$_1$, awayGolname$_2$, …,
awayGolname$_n$}, SolutionBy, aGoal)
     RefreshModView({awayGolname$_1$, awayGolname$_2$,
…, awayGolname$_n$}, SolutionBy)
        Declare({modname$_1$, modname$_2$,…,modname$_n$},
Module)
```

```
            Define({$modname_1$, $modname_2$,…,$modname_n$}, SolutionBy, aGoal)
        RefreshModView({$modname_1$, $modname_2$,…,$modname_n$}, SolutionBy)
        Declare({$contrname_1$, $contrname_2$, …, $contrname_n$}, Contract)
            Define({$contrname_1$, $contrname_2$, …, $contrname_n$}, SolutionBy, aGoal)
        RefreshModView({$contrname_1$, $contrname_2$, …, $contrname_n$}, SolutionBy)

    else
            Step_GiveSolution(aGoal)
    EXIT

Step_GiveSolution(aGoal::Goal)
            Declare({$solname_1$, $solname_2$, …, $solname_n$}, Solution)
            Define({$solname_1$, $solname_2$, …, $solname_n$}, SolutionBy, aGoal)

Step_DefineContextInfo(aElement::TrunkElement)
    ifContextNeeded(aElement)
        Declare({$contname_1$, $contname_2$, …, $contname_n$}, Context)
        Define({$contname_1$, $contname_2$, …, $contname_n$}, ContextBy, aElement)
        Declare({$awaycontname_1$, $awaycontname_2$, …, $awaycontname_n$}, AwayContext)
        Define({$awaycontname_1$, $awaycontname_2$, …, $awaycontname_n$}, ContextBy, aElement)
        RefreshModView({$awaycontname_1$, $awaycontname_2$, …, $awaycontname_n$}, ContextBy)
        Declare({$awayGolname_1$, $awayGolname_1$, …, $awayGolname_1$}, AwayGoal)
        Define({$awayGolname_1$, $awayGolname_1$, …, $awayGolname_1$}, ContextBy, aElement)
        RefreshModView({$awayGolname_1$, $awayGolname_1$, …, $awayGolname_1$}, ContextBy)
    ifAssumptionNeeded(aElement)
        Declare({$assumname_1$, $assumname_2$, …, $assumname_n$}, Assumption)
        Define({$assumname_1$, $assumname_2$, …, $assumname_n$}, ContextBy, aElement)
    ifJustificationNeeded(aElement)
        Declare({$justname_1$, $justname_2$, …, $justname_n$}, Jusitification)
        Define({$justname_1$, $justname_2$, …, $justname_n$}, ContextBy, aElement)
        Declare({$awayGolname_1$, $awayGolname_2$, …, $awayGolname_n$}, AwayGoal)
        Define({$awayGolname_1$, $awayGolname_2$, …, $awayGolname_n$}, ContextBy, aElement)
        RefreshModView({$awayGolname_1$, $awayGolname_2$, …, $awayGolname_n$}, ContextBy)
```

3 Application

This chapter takes brake system software on airplane for example to illustrate to construction of modular GSN argument. This software is the core control software of the landing gear brake control system on the airplane, which collects information like wheel speed sensor signal and braking instruction signal, and realizes the function of braking, skid resistance, and ground protection. It is the key software to ensure the safety of taking-off and landing of the plane, so it must have a high-level of safety and reliability.

(1) Define the top-level claim

The theme of this example is to ensure the dependability of braking software, we will consider this matter from two angles: safety and dependability. Due to space limitations, this section only demonstrates the argument of the safety. Therefore, we set the top-level claim as “the braking software on the airplane is safe”.

(2)Preliminary establish the structure of argument

According to the top-level claim established above, we can preliminary establish the structure of argument. The nature of software safety is “the ability of not causing an accident of the software”, concerns about the safety of the software are derived from system accidents, system accidents are caused by system hazards. Therefore, to analyse software safety, we must look from the system level. We must consider the role of the software as a component in accident of the system, and the contributions they make to system hazards. Analysis of these contributing factors, proposal of safety requirements in related software, eliminating or retarding the danger caused by software in the system, these are the keys to ensure software safety. Therefore, in this example, we preliminarily established the three-tier argument structure of “top-level claim-system hazard-safety function”. Argument structure is shown in figure 6.

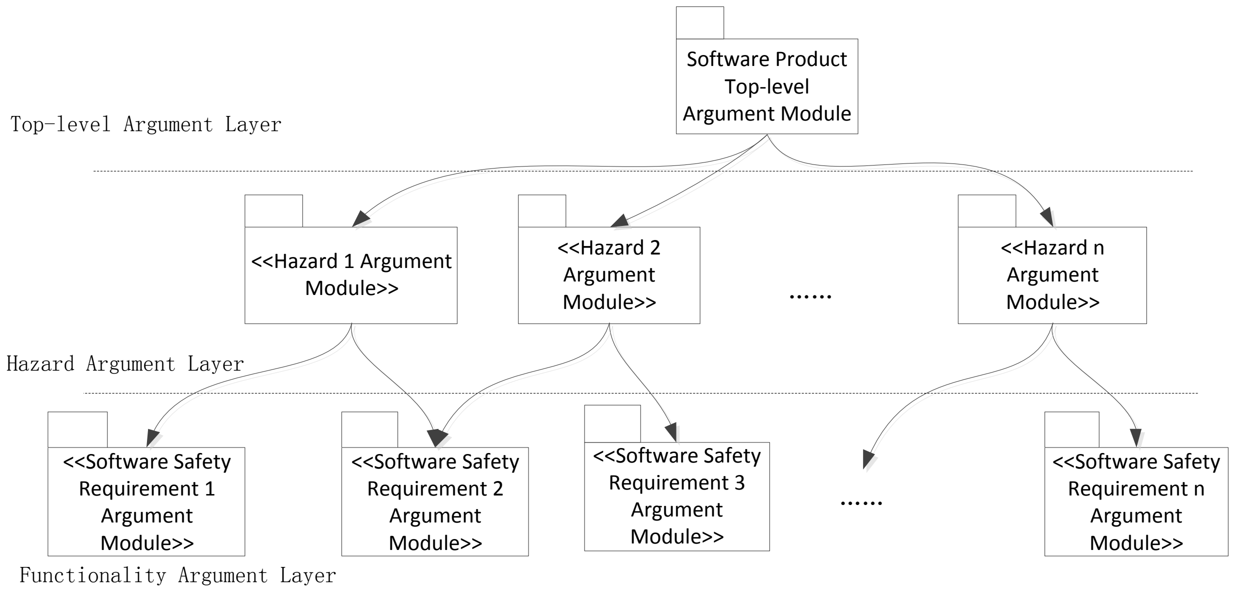
Software Product Top-level Argument Module
Top-level Argument Layer
<<Hazard 1 Argument Module>>
<<Hazard 2 Argument Module>>
......
<<Hazard n Argument Module>>
Hazard Argument Layer
<<Software Safety Requirement 1 Argument Module>>
<<Software Safety Requirement 2 Argument Module>>
<<Software Safety Requirement 3 Argument Module>>
......
<<Software Safety Requirement n Argument Module>>
Functionality Argument Layer

FIGURE 6 Safety argument structure of braking system software

(3) Build concrete argument and elaborate argument view

After the initial argument structure, we need to further refine each module. According to the build process presented in the above section, we can systematic develop internal argument of each module. Take the top-level argument module in software products for example:

First, declare top-level goal Declare({topGoal}, Goal)

We have declared a goal instance topGoal=Goal(“The final implementation of ABS software meets the software safety demands”), in which topGoal is the ultimate goal of important argument in this module.

Then, define background information of the goal Step_DefineContextInfo(topGoal). Realization of software product defines and explains through requirements, design documentation, and source code. Therefore, we define the background information of topGoal as Declare({contname$_1$, contname$_2$, contname$_3$}, Context), in which contname$_1$=Requirements of ABS software,contname$_2$=Design Specification of ABS software,contname$_3$=Source Code of ABS software.

After finishing the goal information definition, we can determine whether the goal can be further decomposed. According to the macro module structure, we further decompose top-level goal adopting the way of risk-orienting. Therefore, we will enter the iterative process of goal decomposition Step_DecomposeGoal(topGoal). Define argument strategy as Delare({aStrategy}, Strategy),aStrategy={“Argument over hazard introduced by ABS software”}. Then, define relevant background information for the strategy. We defined two context elements, Declare({contname$_1$, contname$_2$}, Context),contname$_1$={“Hazard list of ABS software”},contname$_2$={“Hazard Identification method of ABS software”}. We also declared an Assumption element Declare({assumname$_1$}, Assumption), Assumption information is given in the argument strategy assumption$_1$={“Hazards are independent and can be argumented respectively”}.

Based on the information in argument strategy background, we listed three system hazards to be respectively argued. According to the macro argument structure, each hazard should be argued in a separate module, and we must use awayGoal instances to show interface between top-level module and hazard argument module. We will declare three, like Declare({awayGolname$_1$, awayGolname$_2$, awayGolname$_3$}, AwayGoal),awayGolname$_1$={“Hazard ‘Airplane can’t decelerate by braking function’ is managed adequately”},awayGolname$_2$= {“Hazard “Tire blowout” is managed adequately”},awayGolname$_3$= {“Hazard ‘sideslipping and off tracking’ is managed adequately”}. After finishing the declare of awayGoal element, we didn’t declare other modular modules. By the modular building approach above, decomposition process of the goal is over.

Graphical results of modular argument are shown in figure 7.

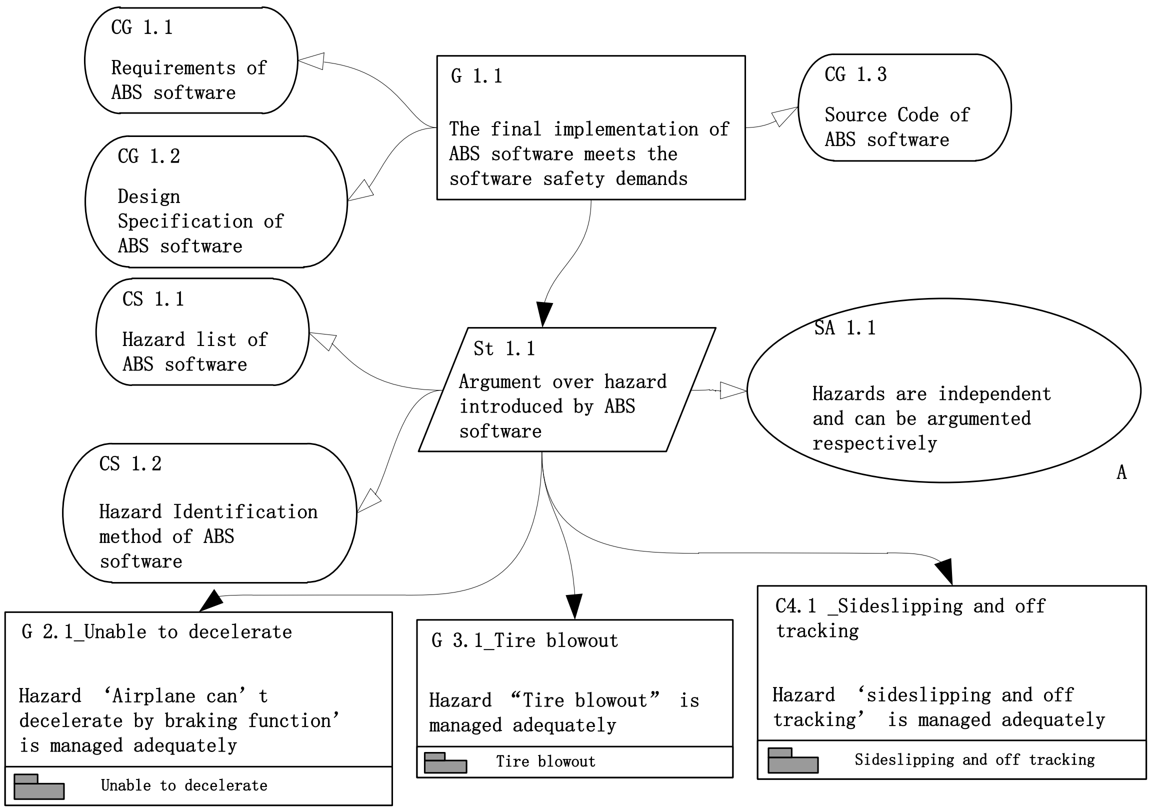
CG 1.1
Requirements of ABS software
G 1.1
The final implementation of ABS software meets the software safety demands
CG 1.3
Source Code of ABS software
CG 1.2
Design Specification of ABS software
CS 1.1
Hazard list of ABS software
St 1.1
Argument over hazard introduced by ABS software
SA 1.1
Hazards are independent and can be argumented respectively
A
CS 1.2
Hazard Identification method of ABS software
G 2.1_Unable to decelerate
Hazard 'Airplane can' t decelerate by braking function' is managed adequately
Unable to decelerate
G 3.1_Tire blowout
Hazard "Tire blowout" is managed adequately
Tire blowout
C4.1 _Sideslipping and off tracking
Hazard 'sideslipping and off tracking' is managed adequately
Sideslipping and off tracking

FIGURE 7 Results of modular argument

According to the top-level argument module, the updated module view is shown in figure 8:

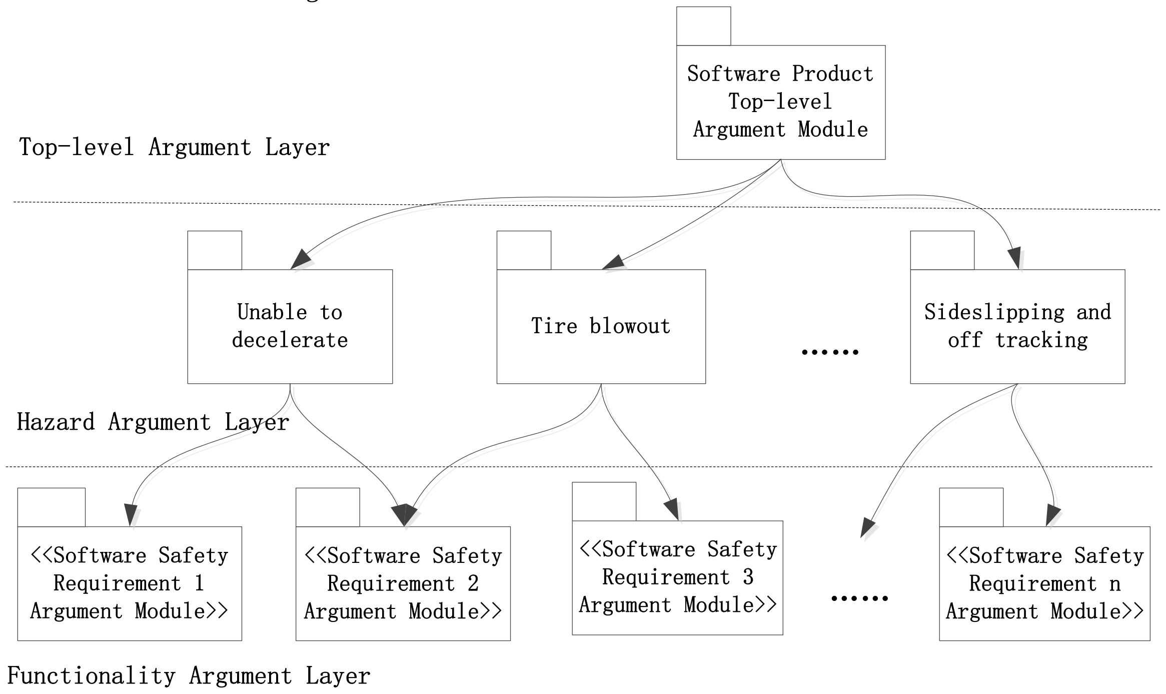
Software Product Top-level Argument Module
Top-level Argument Layer
Unable to decelerate
Tire blowout
......
Sideslipping and off tracking
Hazard Argument Layer
<<Software Safety Requirement 1 Argument Module>>
<<Software Safety Requirement 2 Argument Module>>
<<Software Safety Requirement 3 Argument Module>>
......
<<Software Safety Requirement n Argument Module>>
Functionality Argument Layer

FIGURE 8 view module

4 Conclusion

This paper has presented a systematic construction approach for modular software assurance case. On the basis of in-depth analysis of the argument modelling technique, GSN, we extract the basic concepts and terminology of modularization, and summarize the relationships between concepts and constraints that must be met. A new modular GSN metamodel is proposed, which is a comprehensive, reusable description for the internal logic of modular GSN, and can be seen as the basis of a normative argument construction process for modular assurance case. According to the metamodel, we give the progression algorithm for constructing the core structure of assurance case and present the implementation process of modular GSN argument construction in a "macro-micro-macro" iterative way. Our approach can help the assurance case developers gradually extract and analyse the argument elements, and provides a modelling process guidance for assurance case developer. We apply our approach in an ABS software system, which is a typical control software, to examine the feasibility and effectiveness. Results show that this approach can greatly enhance the development efficiency of dependability assurance case, and can improve systematicness, comprehensiveness and scientificity of the assurance case itself, thus providing strong support for ensuring the software product can reach its desired dependability level.

At present, our approach does not include the concept of GSN pattern and its related elements, and there is also lack of tool supporting. These will be important directions of our future work.

References

[1] Jackson D, Thomas M, Millett L I 2007 *Software for Dependable Systems: Sufficient Evidence?* National Research Council

[2] DO-178B *Software Considerations in Airborne Systems and Equipment Certification*

[3] IEC 61508 *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems*

[4] ISO 15408 *Information technology — Security techniques — Evaluation criteria for IT security — Part 1: Introduction and general model*

[5] ISO 15408 *Information technology — Security techniques — Evaluation criteria for IT security — Part 2: Security functional requirements*

[6] ISO 15408 *Information technology — Security techniques — Evaluation criteria for IT security — Part 3: Security assurance requirements*

[7] Bloomfield R, Bishop P 2010 Safety and Assurance Cases: Past, Present and Possible Future – an Adelard Perspective *Proceedings of the Eighteenth Safety-Critical Systems Symposium*

[8] Habli I, Hawkins R, Kelly T 2010 Software safety: relating software assurance and software integrity *International Journal of Critical Computer-Based Systems* **1**(4) 364-83

[9] Bishop P, Bloomfield R, Guerra S 2004 The future of goal-based assurance cases *Proceedings of Workshop on Assurance Cases of 2004 International Conference on Dependable Systems and Networks*

[10] IEEE P15026-2 *IEEE Standard for Systems and software engineering - Systems and software assurance - Part 2: Assurance case*

[11] Kelly T P 1998 *Arguing Safety – A Systematic Approach to Managing Safety Cases* Department of Computer Science University of York

[12] Miller A, Gupta R 2008 *Assurance Cases for Reliability: Reducing Risks to Strengthen ROI for SCADA Systems Recent Advances in Reliability and Quality in Design* Ed H Pham Springer: London pp 465-89

[13] He Y, Johnson C W 2012 Generic security cases for information system security in healthcare systems *7th IET International Conference on System Safety, incorporating the Cyber Security Conference*

[14] Nakazawa J, Matsuno Y, Tokuda H 2011 Evaluating degree of systems' dependability with semi-structured assurance case *Proceedings of the 13th European Workshop on Dependable Computing(Pisa, Italy)ACM* pp 111-2

[15] Ankrum T S, Kromholz A H 2005 Structured assurance cases: three common standards *Proceedings of Ninth IEEE International Symposium on High-Assurance Systems Engineering (HASE 2005) IEEE*

[16] Bishop P, Bloomfield R 1998 A methodology for safety case development *Safety-Critical Systems Symposium (SAFECOMP)*

[17] Cyra L, Górski J 2011 Support for argument structures review and assessment *Reliability Engineering & System Safety* **96**(1) 26-37

[18] Kelly T, Weaver R 2004 The Goal Structuring Notation – A Safety Argument Notation *Proceedings of Dependable Systems and Networks*

[19] Langari Z, Maibaum T 2013 Safety cases: A review of challenges *Proceedings of 1st International Workshop on Assurance Cases for Software-Intensive Systems (ASSURE)*

Authors

Dajian Zhang, born in 1982, China

Current position, grades: Ph.D. candidate
University studies: Beihang University
Scientific interest: software safety, software reliability
Publications: 2
Experience: He received his Bachelor degree in School of Information and Computing Science from Beijing Jiaotong University and now he is a Ph.D. candidate in School of Reliability and System Engineering of Beihang university.

Minyan Lu, born in 1963, China

Current position, grades: Professor
University studies: Beihang University
Scientific interest: software dependability engineering, reliability engineering
Experience: She received her doctor degree in School of Reliability and System Engineering from Beihang University and now she is a professor in School of Reliability and System Engineering of Beihang university.

Nan Wu, born in 1990, China

Current position, grades: postgraduate
University studies: Beihang University
Scientific interest: software reliability engineering
Publications: 1
Experience: He is a postgraduate in School of Reliability and System Engineering of Beihang university.