MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

JUSTIFYING SOFTWARE SYSTEMS SAFETY USING ARGUMENTS

LICENSE THESIS

Graduate: Nicoleta Catalina MARC
Supervisor: Lect.dr.eng Adrian GROZA

2014

MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Nicoleta Catalina MARC**

JUSTIFYING SOFTWARE SYSTEMS SAFETY USING ARGUMENTS

1. **Project proposal:** *In safety-critical applications it is necessary to justify the software conformance with specifications and standards. Certification bodies require the construction of assurance cases, that contain claims supported by evidence obtained during development and testing of the system. It is important to build well-structured and coherent safety cases, because wrong construction and reasoning in safety arguments can undermine system's safety claims and lead to failures. We developed a tool that facilitate the construction and assessment of safety cases. The tool supports the Goal Structuring Notation standard for creation of safety arguments. The GSN diagrams are translated in description logic, in order to formally check various properties of the safety case.*

2. **Project contents:** *Presentation page, Introduction, Project Obectives and Specifications, Bibliographic research, Analysis and Theoretical Foundation, Detailed Design and Implementation, Testing and Validation, UserâĂŹs manual Conclusions and Further Work, Bibliography, Appendices*

3. **Place of documentation:** Technical University of Cluj-Napoca, Computer Science Department

4. **Consultants:** Lect.dr.eng. Adrian Groza
5. **Date of issue of the proposal:** November 1, 2014
6. **Date of delivery:** July 3, 2014

Graduate: Nicoleta Catalina Marc

Supervisor: Lect.dr.eng. Adrian GROZA

MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnata Marc Nicoleta Catalina , legitimat(ă) cu seria AX nr. 377045 CNP 2911205012661, autorul lucrării Justifying software systems safety using arguments

elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la Facultatea de Automatică şi Calculatoare, Specializarea Calculatoare in limba Engleza din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea Iulie a anului universitar 2013-2014, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

În cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data _____

Nume, Prenume _____

Semnătura

List of Tables

List of Figures

Contents

Chapter 1

Introduction

Chapter 1 describes the problem statement and its solution.

In the past few years the number of safety critical systems has grown, the question is how do we know if they are safely-made and secure, any system that presents a certain level of risks must prove that its behavior should be trusted. Nowadays software safety assurance is often demonstrated by compliance with national or international safety standards. The assurance cases are used to decide if the system is safe and secure solely from the provided evidence. Their usage has grown in the last years and in certain domains, like the nuclear field, it is mandatory to build a safety case. This proves that, currently, assurance cases are the answer to our question. Assurance cases have evolved from the concept of safety cases and they contain claims supported by the evidence obtained during development and testing of the system. It is very important to build well-structured and coherent safety cases because wrong construction and reasoning in safety arguments can undermine a systems safety claims and lead to failures of the system. Over the last years, many documents have appeared about how to describe and graphically structure an assurance case. One of these documents is the Goal Structuring Notation, also known as GSN. The Goal Structuring Notation is an argumentation notation used to structure and graphically represent a safety argument. Even though we have a solid standard for representing safety cases, it doesn't reduce the risk of building incomplete and bad-structured assurance cases because of bad assessment of tha safety case, i.e the number of claims could be very high and therefore will be hard to follow by the safety engineer. A solution to this problem is to develop a tool that facilitates the construction and assessment of safety cases. The main feature of the tool is to translate the GSN graphical notation into description logic in order check the GSN model for consistency. Hence, the GSN model for a safety critical application can be specified both in graphical notation and in description logic. The advantage is that the specific reasoning services of description logic are enacted to verify the compliance of the case with the GSN standard and also to signal possible argumentation flaws.

1.1 Organization of the paper

Chapted I presents an introduction in arguing software safety domain.

In Chapter II we introduce the theme and the objectives of our system.

In Chapted III are presented all the references for the project within the project domain and related work.

Chapter IV presents the elements of a safety argument, introduces the techincal instrumentation used: the Goal Structuring Notation plus description logic.

Chapter V details the architecture of the system and implementation.

In Chapter VI are presented the needed resources and steps for installing the application, and how to use it.

Chapter VII focuses on methods for testing and validating our tool.

Finally, in Chapter VIII we present the conclusion and further work.

Chapter 2

Project Obectives and Specifications

2.1 Problem Statement and Solution

For many systems that present a certain level of resk safety is a good practice to justify, prior to deployment, if and why the sofware behavior should be trusted. Because structured and evidence-based arguments are more and more used to describe the assurance of the system, many international standard adopted this strategy.

As explained in table 2.1, the problem arises in assessing complex safety cases and proving that the system is sufficiently secure through the sufficient solutions and evidence are provided for each stated claim in the safety case. The found solution is using ontology reasoning serivices because the ontology provides a source of well-defined terms that can be used in descriptions of safety case nodes. The safety case knowledge can be represented as a ontology and after queries can be run on that ontology.

Table 2.1: Problem Statement and Solution

The problem of	building incomplete safety cases because of bad safety case assessment
affects	system's safety and the safety engineer
the impact of which is	incomplete description of systems assurance
a successful solution would be	improve the assement of the tool using reasoning services

2.2 Goals and Objectives

2.2.1 Bussiness Goals and Objectives

Nowadays many international standards require for systems that present a certain level of risk to justify, prior to deployment, why software behavior is to be trusted. Because

of this, the bussiness goals and objectives for this project will focus on implementing a tool that:

- improves assement of safety cases
- reduces the risks of not building well-structured and complete assurance cases
- is easy to use
- eases the work of the safety engineer
- enables the evaluation of the realism of the cases

2.2.2 Project Goals and Objectives

The project main goal is to develop a system that:

- accomplish project business goals and objectives
- supports automated construction and assessment of safety cases
- that can be used to justify the correctenes of the users critical systems and if systems obeys the international standards
- translates the safety case into description logic
- uses reasoning services for better assesment of safety cases
- validates the safety case
- generates documentation and reports for the safety case

2.3 Functional requirements

In the use case diagram from figure 2.1 we have captured the requirements of the system. Using our tool, the user should be able to:

1. build and edit safety case diagram
2. export any diagram as image (jpg or png)
3. get a file conatining the translated the diagram into A-box
4. load or set the current Abox for querying
5. query the diagram from a special console used only for this

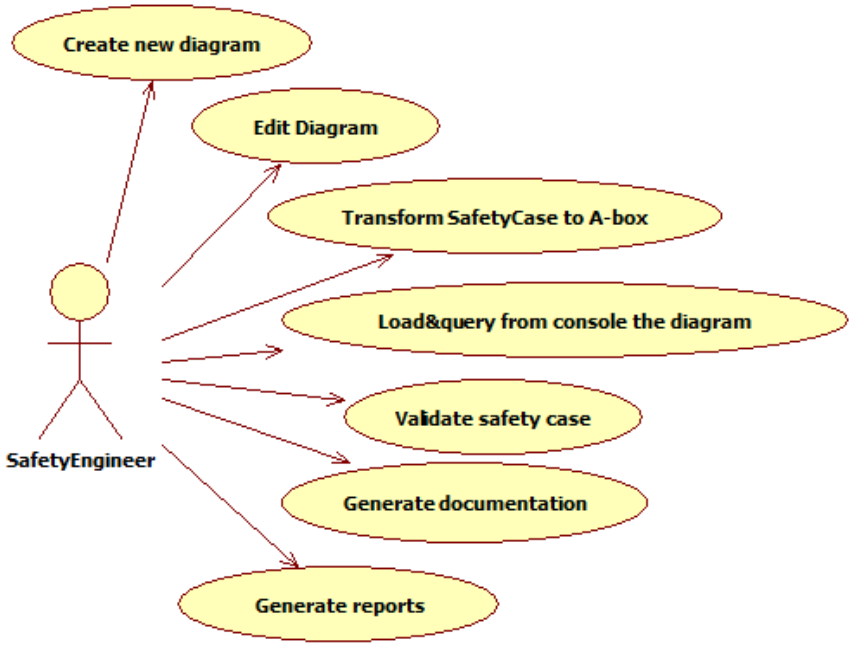

Figure 2.1: System'usecase

6. validate his safety case and get a file with the status of the validation

7. Generate documentation and reports

2.4 Non-functional requirements

Usability:
The user interface shall be very easy to use and intuitive. Two goals should be strived for one click away functionality intuitive interface - zero training. Also the system should prevent the user before making errors and if the errors are made the the user will we notified about the errors.

Extensibility:
Take into consideration further work, extending and adding product features.

Documentation:
An Administration Guide and a User Guide should be developed in .pdf format.

Operating constraints:

Reasoning engine RacerPro are needed in querying the safety case, plus java jRacer library for creating a java racer client connection to the engine. NaturalOwl engine is used to generate documentation of de description logic in natural language.

Reusability:

Ontology code should be in a separate plug-in so that it can be reused. Do the same for the command console code.

Chapter 3

Bibliographic research

In order to build the tool we have studied several tools used for building safety cases, among them ACedit [1] and AdvoCATE [2] Both tools are used to create safety cases and use the GSN for structuring the safety arguments.

ACedit [1] is an open-source editor used to create Assurance Cases based on the Goal Structuring Notation standard and the OMG Argumentation Metamodel. The tool can be used only for creating and editing a safety case, it lacks at the assessment of the safety-case. Out tool is an improvement of this tool, being added new features like the option of querying the diagram using ontologies, better validation, creation of reports and documentation, usage of safety metrics for the assessment of the safety cases. Figure 3.1 represents a screenshot of this tool.

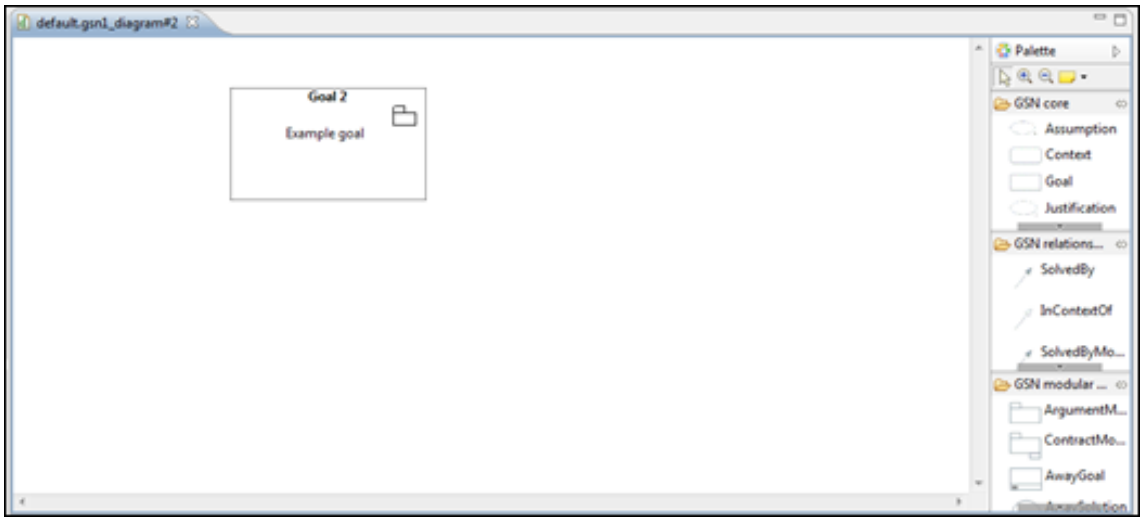

Figure 3.1: Acedit tool screenshot

AdvoCATE [2] is an Assurance Case Automation ToolsEt, to support the automated construction and assessment of safety cases. The tool is more complex then ACedit.

AdvoCATE is an assurance case automation toolset, and has been build to support the automated construction and assessment of safety cases. The main features of the tool are: i) create and edit of assurance cases; ii) import and export of a variety of safety case formats currently those produced from the ASCE, CertWare, and D-Case tools; iii) assemble automatically fragments of safety cases; iv) Computation of metrics, direct measurement and evaluation of the safety case

Figure 3.2 represents a scrrenshot of this tool.

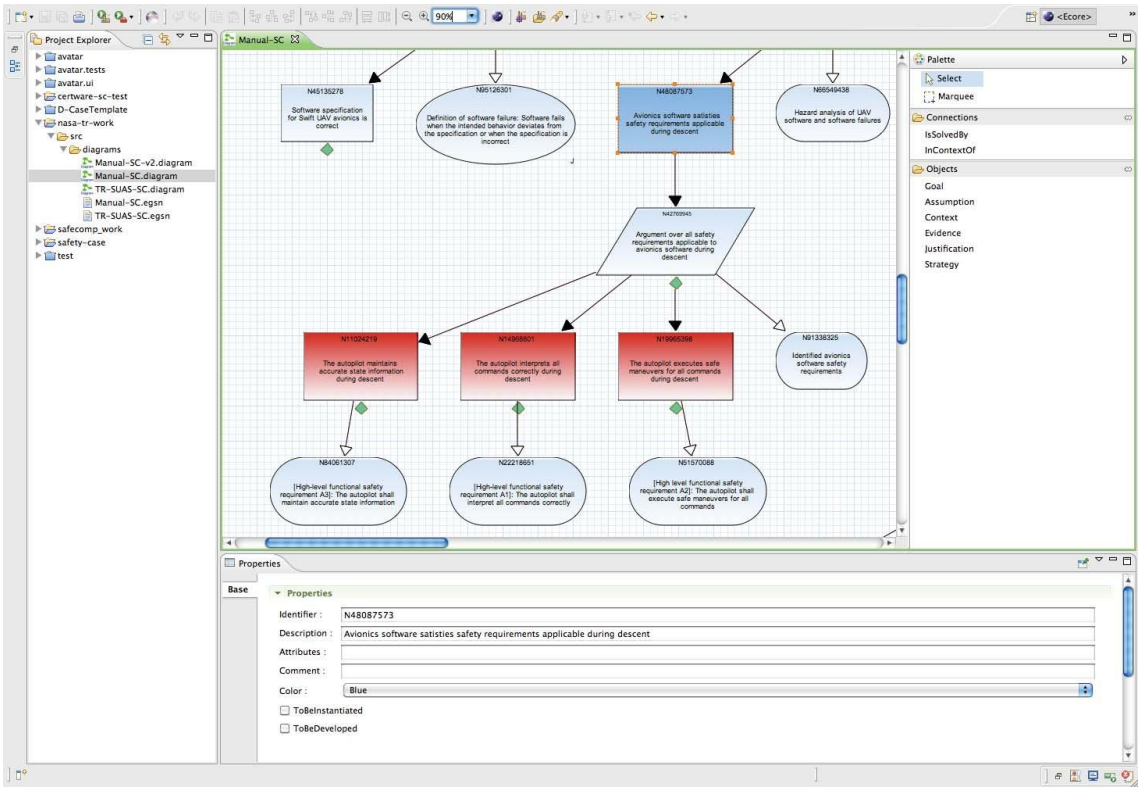

Figure 3.2: AdvoCATE tool screenshot

The novelty of our approach is that the assurance case is automatically translated in description logic. The advantage is that the specific reasoning services of description logic are enacted to verify the compliance of the case with the GSN standard and also to signal possible argumentation flaws. The difference between our tool and this one is the fact that using ours the user will be able to build simple or complex queries for interrogating the diagram at any time during the development and not being limited only to metrics, this is a plus at the assessment of the diagram. D-case Editor is another tool for constructing safety cases, developed by DEOS(Dependeble Embedded Operating System for Practical Uses). Similar to our tool, it is an Eclipse plugin based on the Eclipse GMF framework and supports GSN standard. It is a prototype of a dependability case editor and has a

GSN pattern library function and ptototype checking function. Figure 3.3 represents a scrrenshot of this tool.

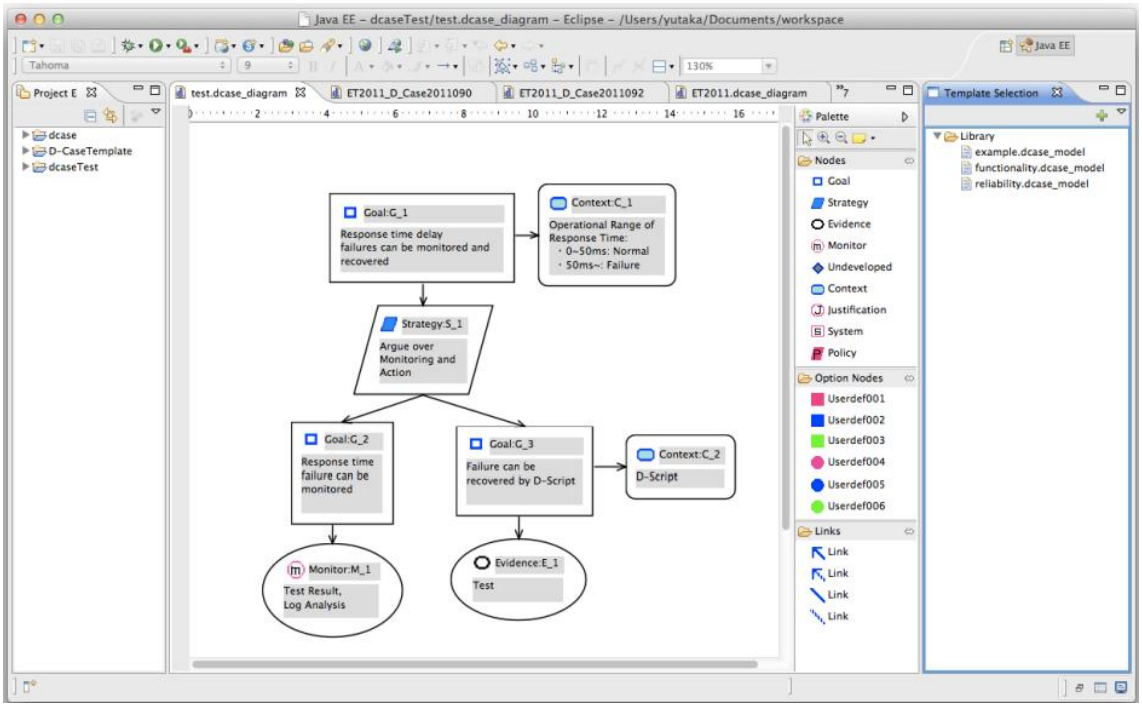

Figure 3.3: D-case editor tool screenshot

A tool for integrating informal and formal reasoning has been proposed in [3]. The focus is on tracing requirements from natural language representation towards formal properties verified using model checking. The tool is a plugin for Eclipse developed on top of ProR plugin for tracing natural language requirements and Rodin for modelling properties in the Event-B language.

For implementing the reasoning feature of our system we studied about description logics from the book [4].It presents description logic plus modelling and technical details, the syntax of knowledge bases of the description logic $\mathcal{SROIQ}$, presents the $\mathcal{ALC}$ description logic.

In paper [5] is presented the RacerPro system, a knowledge representation system based that implements a highly optimized tableau calculus for a very expressive description logic. Advantages of using this engine is that it offers reasoning services for multiple T-boxes and for multiple A-boxes as well and it provides us an API with Racer Client for Java.

For safety cases building and assesment we studied the following articles: [6] and [7]. In the paper [6] is presented a new way to structure assurance cases using assured safety arguments. According to the authors any safety arguments has two components:

- a safety argument that documents the arguments and the evidence used to establish direct claims of system safety
- a confidence argument that justifies the sufficiency of confidence in this safety argument.

This decomposition gives greater clarity of purpose and helps avoiding the introduction of unnecessary arguments and evidence. Many difficulties are encountered when having a single argument elements that documents both direct arguments of risk mitigation and supporting arguments, for example:

- Because there is too much information in just one argument, the arguments will become to large and and unwieldy, leading to difficulty in reviewing them because of the size and lack of focus.
- It is more difficult to see the incompleteness or poor structure in the safety argument
- Arguments tend to be indirect and unfocused, and the link between elements of the argument and risk is often lost.
- Arguments become difficult to build, and weaknesses of the argument are sometimes not evident and so are easily overlooked.

These difficulties are serious since they all detract from the basic purposes of us- ing safety cases. Linking the two arguments provides a mechanism for guiding analysis of the interrelationship between safety and confidence. Both papers [6] and [7] claim that any safety argument should focus on identification, management and mitigation of hazards associated with the system. All the elements of a safety case are part of the causal chain of hazard. The safety case is based on claims, any claim is broken into sub-claims until is reached a point when the claims can be proved by the development or assessment of an artefact as evidence. Every claim and strategy adopted tu support the claim shoulf be very clearly formulated and state the context in which the argument is made. The solution proposed by the authors for this is to represent graphically the safety argument because it is clearer than through narrative text because in a narrative text is more difficult for the reader to identify the individual elements and structure of the argument. In paper [8] the

authors propose a method for assessing software safety and security standards by capturing and criticising their arguments. This method assumes following three steps: argument capture, argument criticism, and issue sentencing. In the first step, argument capture, the standard requirements are ideentified through analysis and also the specific claims assumed by those requirements and the evidence supporting the claims.The standard's text should be used in the captured argument as much as practical.Throughout this process, the standard should be captured as accurately as possible. The second step called argument criticism, focuses on reviewing the fragments of argument by following the next phases:

1. Take into consideration misinterpretations of the argument

2. Try to draw out implicit or explicit assumptions
3. Judge the necessity of each assumption
4. Search for errors the argument
5. Identifies where "independent" lines of reasoning depend upon common sub-arguments
6. Take into account strengthening the argument
7. Determines whether negative experience with similar systems might provide counterevidence
8. Judges the strength of the argument

In the last step, issue sentencing, the analyst shoul re-examines each identified issue to determine what hazards from the standard are reflected More about safety argument strategis have been studied from paper [9]. In his papers he investigates the issues of autonomy for safety-critical systems, the ways of safety assurance and the structure of safety arguments. He proposes two approches for autonomous vehicle safety. The first one is the classical one uses hazard analysis approach and is based on safety barriers. This solution aims to identify event sequences leading to accidents and ways to control risks, the safety is perceived in a binary way. The second solution is based on the dynamic risk assesment approach, i.e. design a system which is able to perceive and interpret risk factors and evaluate if fowarding will lead to a safe state or ends with an accident. Autonomous systems rise problems for safety analysis and safety assurance, and therefore for certification. ISO 26262 standard and safety cases of systems obeying this standard have been studied in papers [10] and [11] The standard for vehicles requires building a safety case for electrical/electronic that presents a certain level of risks in order to prove that the system requirements are complete and satisfied by evidence.

In the paper [12] the user presents the case study of Therac 25 machine that had massively overdosed six people. Therac 25 is a computer controlled therapy machine that can treat the patient with relatively low energy electron beams or with X-ray. From the safety analysis of the system, it can be seen that the developers focus more on the technology from olders versions of this machine than on the changes introduced by the new machine, they didn't go throug all phases of the project development, residual software errors have been omitted from the analysis, many hardware safety stops were removed because they thought that the software was better. Another bad practice commited by them: not performing risk assessment for the software, reuse of older machine's software without checking for faults and making the system hard to use by the others (the error messages were very difficult to understand because they didn't specified the exact cause, they were like "MALFUNCTION 47" OR "VILT" and therefore the errors were ignored by the practitioners). The case presented by the author is just a start because it doesn't

contain very detailed the system requirements, hazard identification and a risk analysis haven't been performed, but a top level safety case can be build. The most important requirements established by the author are: more detailed error messages, the maximum limit for MeV electron treastments should be set,occurence of errors should stop the machine, the computer should select the correct energy and modes.

Chapter 4

Analysis and Theoretical Foundation

4.1 Safety Case

Safety cases are used in risk safety domain, in case of systems that presents a certain level of risks in order to prove that the system requirements are complete and satisfied by evidence.

In this thesis the safety case is defined in the following terms:

" *A safety case should communicate a clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context* "

Assurance cases are build using safety arguments, each safety arguments should focus on identifying and reducing the hazards associated with the system. The safety argument is based on claims, any claim is broken into sub-claims until is reached a point when the claims can be proved by the development or assessment of an artefact as evidence. Every claim and strategy adopted to support the claim shoulf be very clearly formulated and state the context in which the argument is made so that the graphical representation of the safety case will be well structured.

The elements of any safety case are represented in figure 4.1, they are:

- Claim: the statement or assertion
- Evidence: represents the provided evidence, facts assumptions or subclaims to support the claim
- Argument: it is the linkage between the claim and the evidence
- Inference: the mechainsm providing the transformational rules for the argument

The most important aspects of an safety case are:

- argument: any safety case builds an argument based on the provided evidence, according to this argument someone can reasonably determine that the system is relatively safe

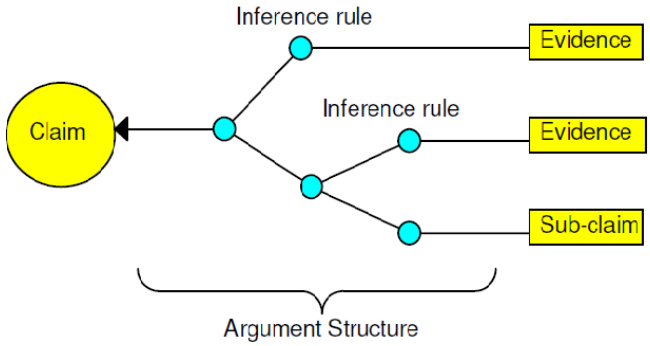

Figure 4.1: Elements of a safety case

- clear: any safety case must be clear because all the information provided must be understood by unexperienced people.
- system: a safety case can be build for any system
- acceptably: it is very hard to prove that a system is 100% safety, the goal of an assurance cases goal is to convince the others that the system is safety enough to be used.
- context: The context of using the system must be specified in the safety case, because context-free safety is impossible to argue.

4.1.1 ISO26262 standard

Many safety standards require now the construction of safety cases for systems that may present a certain level of risk.

THe ISO 26262 standard states that any electrical/electronic product must ensure an acceptable level of safety and requires building a safety case, but it does not tell you the steps of building it [13]. Fig. 4.3 shows how such an analysis is performed in order to comply to the ISO26262 requirements, according to [14]. The figure presents only the "hazard analysis and risk assessment" component. The top level goal *Goal1* is to show if the product ensures a sufficient and acceptable level of safety.

The user should structure the safety case into product assurance cases and process related assurance cases.

In figures 4.3, 4.4, 4.5 is developed only the "hazard analysis and risk assessment" claim of the product and is shown the corresponding process-based (Goal 2) and product-based(Goal 6) arguments.

The process-based goal *Goal2* in refined in Fig. 4.4. The goal claims that the process adopted to develop the product is correct and successfully completed. *Goal2* is divided, taking in account the roles (*Strategy1*) and activity steps(*Strategy2*), in 3 sub-goals:

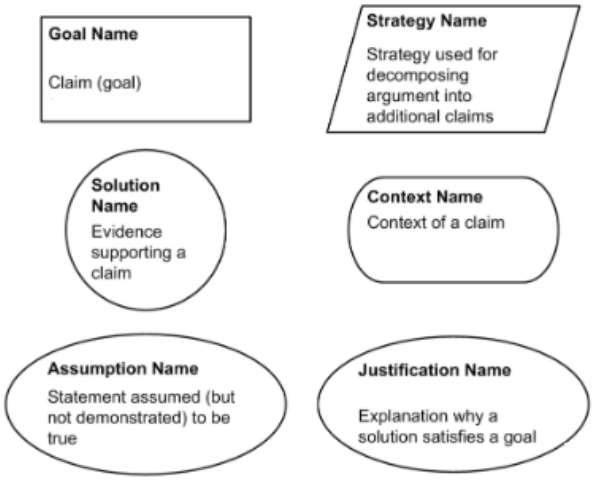

Figure 4.2: Goal Structuring Notation elements

Figure 4.3: Partial goals structure

Goal3,*Goal4* and *Goal5*. *Goal4* claims that the hazards regarding the adapted process of building the product have been identified and classified, using the Hazard identification and analysis using HAZOP technique(HAZard and Operability analysis) to provide the evidence, representing *Evidence2* node, while *Goal5* claims that all the hazard have been carefully analyzed backward and forward, providing as solution hazard identification and analysis using HAZOP technique(HAZard and Operability analysis) represented as *Evidence3* and Failure Modes and Effects Analysis (FMEA) procedure and Fault Tree Analysis technique (FTA) as *Evidence4*.

The product-based goal *Goal6* is justified in Fig 4.5. This claims that the system has the required safe behavior, if something fails then the system should be able to fail in a safe way.The goal is divided in two goals: *Goal7* and *Goal8*. *Goal7* claims that all the hazards regarding the product have been found, while *Goal8* states that the the effects and causes of hazardous events have been analyzed. The goals have as solution techniques

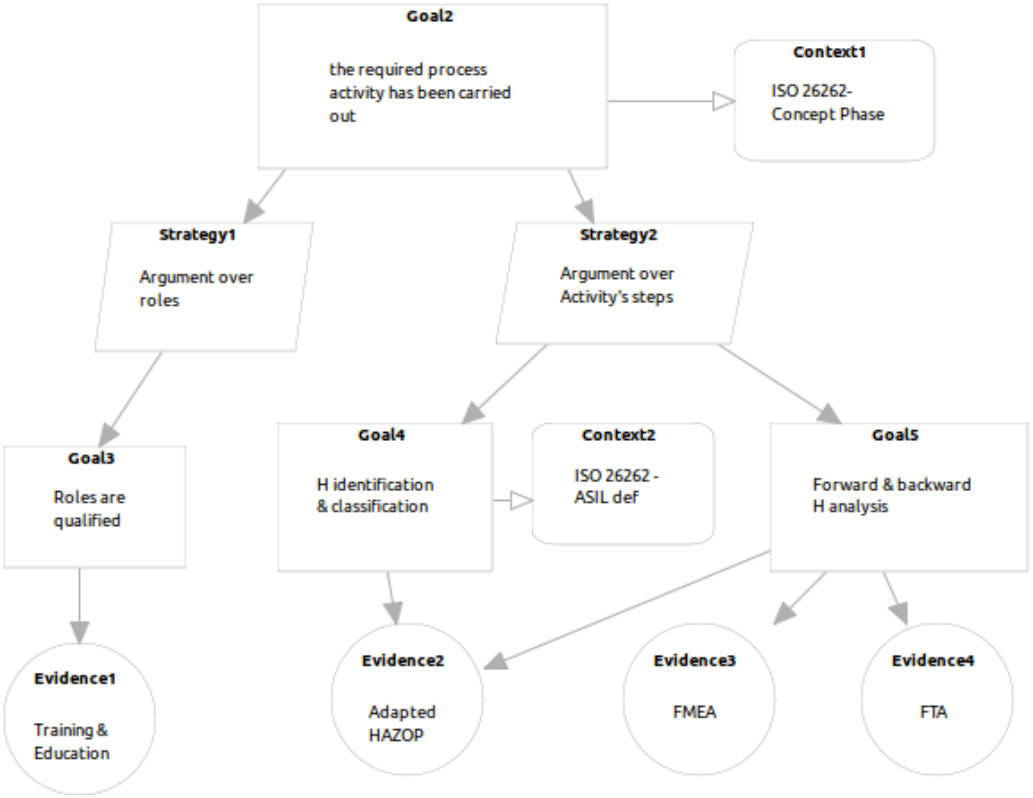

Figure 4.4: Goal structure for the process based argument.

the same nodes *Evidence2*, *Evidence3*, *Evidence4*.

4.2 Goal Structuring Notification

Our tool uses Goal STRucturing Notation to represent assurance cases structure, mainly because inn last years, GSN has been more and more used in risk-based safety domain. GSN presents a new argumentation notation for structuring and representing graphically safety arguments

4.2.1 Elements of GSN

Having a well defined structured, GSN standard increases the chance of identifying gaps in proving the goal and providing evidence.

According to GSN standard, the main elements of an safety argument are: goal, strategy, solution, context, assumption, justification. The Goal represents the statement that needs to be proved, it can be divided in sub-goals until its reached the level where the

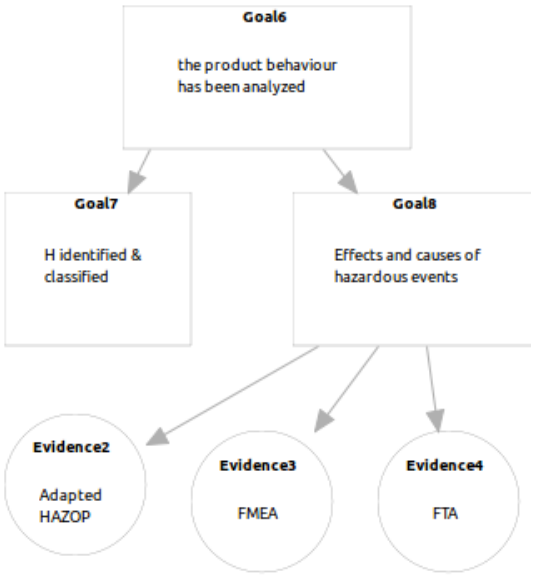

Figure 4.5: Goal structure for the product based argument.

sub-goal can be supported by evidence. Strategy element is used to describe the method aproached by the system to prove the claim.

A Solution node provides the evidence or references to the evidence supporting the goal. Context element provides information relevant for the corresponding goal, while the assumption element will provide statements that are already true. Justification elements is used to explain why the provided evidence is enough to prove the goal. These elements can be related to each other using two relations: inContextOf , only between goal and context, and solvedBy. Figure 4.2 contains elements of the GSN standard represented as follows: goals with rectangular, strategies with paralelograms, evidence and solutions are represented by circle, assumptions and justifications with ellipse, context by a rectangular with rounded corners.

The supportedBy relation is rendered as an arrow with the solid and allows inferentiol or evidential relationships to be documented. It is used to establish the following connections: goal-to-goal, goal-to-strategy, goal-to-solution, strategy-to-goal, goal-to-solution, goal-to-justification. The inContextOf relation is represented by an arrow with empty head and is used to declare a contextual relationship. This relation permits connection between goal or strategies with context, justification or assumption elements.

In figure 4.6 we have an example of goal structure composed with GSN elements. It can be seen that we have a main goal, the top level goal, which is solved by other goals, the support goals, and strategies. Each goal,in turn, can happen in different contexts or different assumptions about the goal can be made, and the goal could be divided in ther sub-goals, strategies or could be supported by evidence

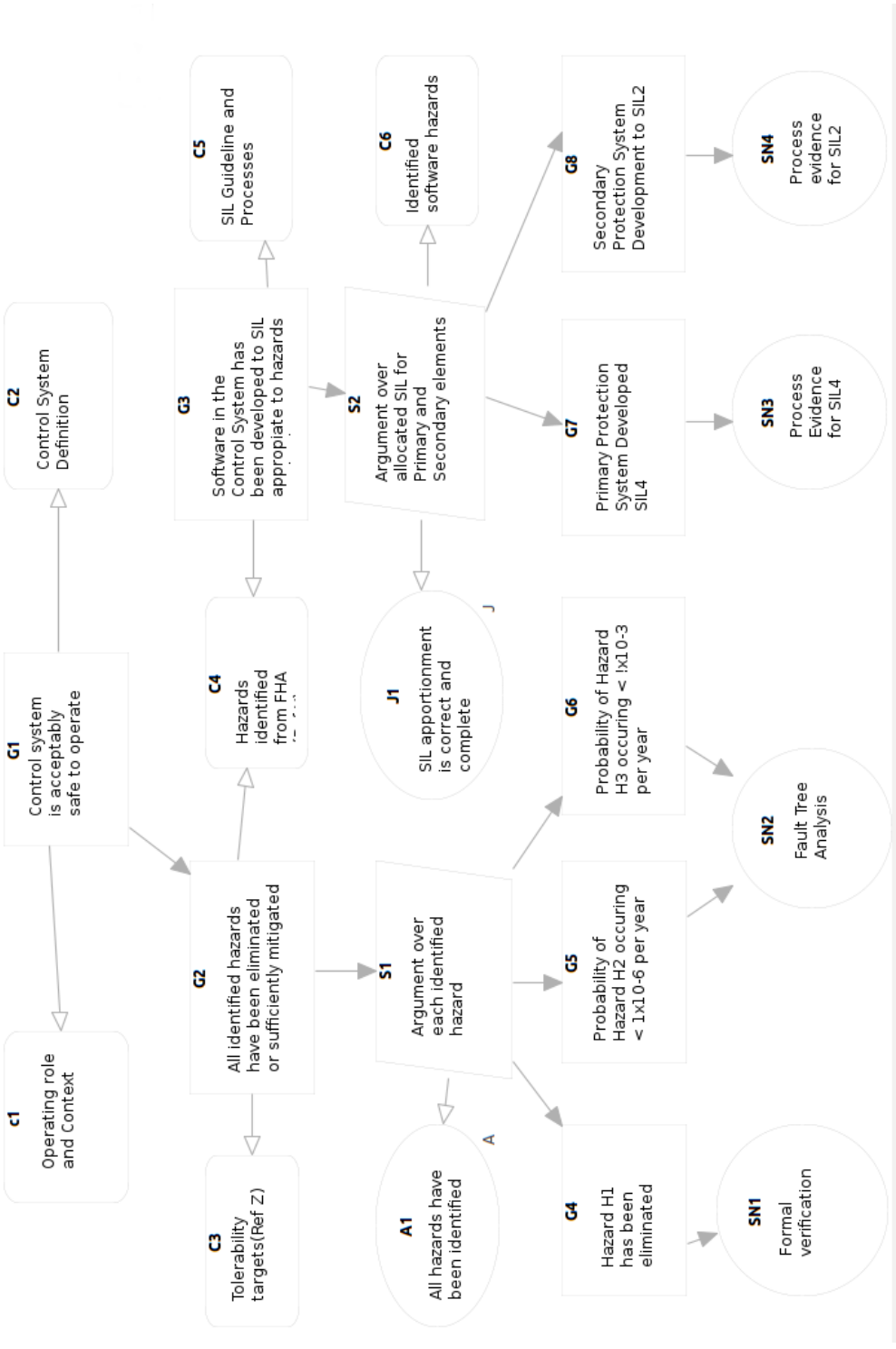

Figure 4.6: Example of safety case represented using GSN.

4.3 Description Language

DLs are based on a vocabulary, also called signature, containing individual names, concept names (unary predicates) and role names (binary predicates).

In the description logic $\mathcal{ALC}$, concepts are built using the set of constructors formed by negation, conjunction, disjunction, value restriction, and existential restriction [4], as shown in table 4.1. Here, C and D represent concept descriptions, while r is a role name. The semantics are defined based on an interpretation $I = (\Delta^I, \cdot^I)$, where the domain Δ^I of I contains a non-empty set of individuals, and the interpretation function $\cdot^I$ maps each concept name C to a set of individuals $C^I \in \Delta^I$ and each role r to a binary relation $r^I \in \Delta^I \times \Delta^I$. The last column of table 4.1 shows the extension of $\cdot^I$ for non-atomic concepts.

Table 4.1: Syntax and semantics of $\mathcal{ALC}$.

Constructor	*Syntax*	*Semantics*
negation	$\neg C$	$\Delta^I \setminus C^I$
conjunction	$C \sqcap D$	$C^I \cap D^I$
disjunction	$C \sqcup D$	$C^I \cup D^I$
existential restriction	$\exists r.C$	$\{x \in \Delta^I \mid \exists y : (x,y) \in r^I \wedge y \in C^I\}$
value restriction	$\forall r.C$	$\{x \in \Delta^I \mid \forall y : (x,y) \in r^I \rightarrow y \in C^I\}$
individual assertion	$a : C$	$\{a\} \in C^I$
role assertion	$(a,b) : r$	$(a,b) \in r^I$

An ontology consists of terminologies (or TBoxes) and assertions (or ABoxes).

Definiţie. A terminology *TBox* is a finite set of terminological axioms of the forms C $\equiv$ D or C $\sqsubseteq$ D.

Definiţie. An assertional box *ABox* is a finite set of concept assertions *a:C* or role assertions *(a,b):r)*, where C designates a concept, r a role, and a and b are two individuals. Usually, the unique name assumption holds within the same *ABox*.

A concept C is satisfied if there exists an interpretation I such that $C^I \neq \emptyset$. The concept D subsumes the concept C, represented by $C \sqsubseteq D$, if $C^I \subseteq D^I$ for all interpretations I. Constraints on concepts (i.e. `disjoint`) or on roles (`domain`, `range` of a role, `inverse` roles, `transitive` properties), number constraints (`max`, `min`), role inheritance (`parent` role) can be specified in more expressive description logics[1].

[1]This paper provides only some basic terminologies of description logics to make it self-contained. For a detailed explanation about families of description logics, the reader is referred to [4].

4.4 Modeling the GSN Standard in Description Logic

The relationship *supportedBy*, allows inferential or evidential relationships to be documented. The allowed connections for the *supportedBy* relationship are: goal-to-goal, goal-to-strategy, goal-to-solution, strategy to goal. Axiom A_1 specifies the range for the role *supportedBy*:

$$(A_1) \quad \top \quad \sqsubseteq \quad \forall supportedBy.(Goal \sqcup Strategy \sqcup Solution)$$

Axiom A_2 specifies the domain of the role *supportedBy*, axiom A_3 introduces the inverse role *supports*, and A_4 constraints the role *supportedBy* to be transitive.

$$\begin{array}{lllll} (A_2) & \exists supportedBy.\top & \sqsubseteq & Goal \sqcup Strategy \\ (A_3) & supportedBy^{-} & \equiv & supports \\ (A_4) & supportedBy & \sqsubseteq & supportedBy \end{array}$$

```
(define-primitive-role supportedBy :domain (or Goal Strategy)
:range (or Goal Strategy Solution) :inverse supports :transitive t)
(define-primitive-role inContextOf :domain (or Goal Strategy)
 :range (or Context Justification Assumption))
```

Inferential relationships declare that there is an inference between goals in the argument. Evidential relationships specify the link between a goal and the evidence used to support it. Axioms A_5 and A_8 specify the range of the roles *hasInference*, respectively *hasEvidence*, while A_6 and A_9 the domain of the same roles. Definitions A_7 and A_{10} say that the *supportedBy* is the parent role of both *hasInference* and *hasEvidence*, thus inheriting its constraints.

$$\begin{array}{lllll} (A_5) & \top & \sqsubseteq & \forall hasInference.Goal \\ (A_8) & \top & \sqsubseteq & \forall hasEvidence.Evidence \\ (A_6) & \exists hasInference.\top & \sqsubseteq & Goal \\ (A_9) & \exists hasEvidence.\top & \sqsubseteq & Goal \\ (A_7) & hasInference & \sqsubseteq & supportedBy \\ (A_{10}) & hasEvidence & \sqsubseteq & supportedBy \end{array}$$

```
(define-primitive-role has-inference :parent supportedBy :domain Goal
 :range Goal)
(define-primitive-role has-evidence :parent supportedBy :domain Goal
 :range (or Evidence solution))
```

Goals and sub-goals are propositions that we wish to be true that can be quantified as quantified or qualitative, provable or uncertainty.

$$\begin{array}{lllll} (A_{11}) & QuantitativeGoal & \sqsubseteq & Goal \\ (A_{13}) & ProvableGoal & \sqsubseteq & Goal \\ (A_{12}) & QualitativeGoal & \sqsubseteq & Goal \\ (A_{14}) & UncertaintyGoal & \sqsubseteq & Goal \end{array}$$

A sub-goal supports other high level goals. Each safety case has a top level *Goal*, which does not support other goals.

$$(A_{15}) \quad SupportGoal \equiv Goal \sqcap \exists supports.\top$$
$$(A_{16}) \quad TopLevelGoal \equiv Goal \sqcap \neg SupportGoal$$

```
(equivalent SupportGoal (and Goal (some supports *top*)))
(equivalent TopLevelGoal (and Goal (not SupportGoal)))
```

For each safety argument, the elements is instantiated and a textual description is attached to that individual by enacting the attribute *hasText* with domain *Statement* and range *String*:

$$(A_{17}) \quad \top \sqsubseteq \forall hasText.String)$$
$$(A_{18}) \quad \exists hasText.Statement \sqsubseteq \top$$

Three individuals *gt*, *gp*, and *gu* of type goal and their textual descriptions are instantiated by assertions f_1 to f_6:

$$(f_1) \quad gt : TopLevelGoal$$
$$(f_2) \quad (gt, \text{"}The\ system\ meets\ its\ requirements\text{"}) : hasText$$
$$(f_3) \quad gp : ProvableGoal$$
$$(f_4) \quad (gp, \text{"}Quick\ release\ are\ used\text{"}) : hasText$$
$$(f_5) \quad gu : UncertaintyGoal$$
$$(f_6) \quad (gu, \text{"}The\ item\ has\ a\ reliability\ of\ 95\%\text{"}) : hasText$$

Intermediate explanatory steps between goals and the evidence include statements, references, justifications and assumptions:

$$(A_{20}) \quad Explanation \sqsubseteq Statement \sqcup Reference \sqcup Justification \sqcup Assumption$$

where these top level concepts are disjoint:

$$(A_{21}) \quad Statement \equiv \neg Reference$$
$$(A_{22}) \quad Statement \equiv \neg Justification$$
$$(A_{23}) \quad Statement \equiv \neg Assumption$$
$$(A_{24}) \quad Reference \equiv \neg Justification$$
$$(A_{25}) \quad Reference \equiv \neg Assumption$$
$$(A_{26}) \quad Justification \equiv \neg Assumption$$

The evidences or solutions form the foundation of the argument and will typically include specific analysis or test results that provide evidence of an attribute of the system. In our approach, the evidence consists in model checking the verification for a specification of the system.

A not verified goal is a goal which has at least one piece of evidence that is not formally proved.

$$(A_{27}) \quad NotVerifiedGoal \equiv Goal \sqcap \exists hasEvidence.NotVerifiedEvidence$$

$$(A_{28}) \quad NotVerifiedEvidence \equiv Evidence \sqcap \exists hasTestResult.(False \sqcap Unknown)$$

4.5 Functionality

4.5.1 Diagram translation and Reasoning in safety arguments

The main idea of translating the builded safety case diagram into A-box is presented in figure 4.7.

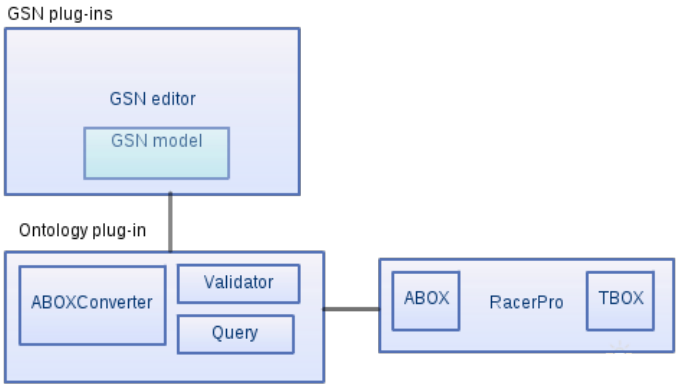

Figure 4.7: System Architecture

We translate the GSN standard into description language, representing the Tbox and for each diagram builded the user can transform the diagram into description language, representing the A-box part. Both parts are loaded in the RacerPro reasoning engine which will be used used by our tool to query and validate the Abox against the GSN Tbox.

Querying the Diagram Scenario

The flow of this action is represented in figure 4.8

Below is presented the steps necessary, without interruptions, for querying from console a selected diagram.

Happy flow:

1. User opens or creates a diagram
2. Select translating the diagram in abox
3. System returns the racer file contaning the abox
4. User loads the diagram

5. User selects the command console

6. User enters query in console

7. System displays the results in the console

In the extended flow are presented the interruptions that might occur when trying to complete this action.

Extended flow:

4.1 Racer engine is not found

4.1.1. System notifies the user that RacerPro was not found

4.1.2. The user starts manually the engine

6.1 Console is not visible

6.1.1 User open window->show perspectives->console

6.1.2 System displays the console view

6.1.3 User selects Command Console from the console view

6.1.4 System displays the command console

6.1 Racer engine stopped working

6.1. System notifies the user that RacerPro was not found

6.2. The user starts manually the engine

6.1 Query is incorrect!

6.1. System notifies in the console the user to re-enter the query

6.2. User re-enters query

4.5.2 Validating the Safety Case

When analysing the diagram by querying the RacerPro engine the safety engineer can simply identify the goals from the diagram that are still undeveloped or not supported by evidence, goal descriptions or retrieve explanation why a goal belongs to a specific concept, check the consistency of the Abox. In this way, the safety engineer can repair the problems and validate.

The following formal verifications are provided by the *SafeEd* system:

1. Every node can be traced back to the top-level claim. That is, there are no ”dangling” nodes or sets of nodes.

Table 4.2: Retrieving information about the safety case.

Query	RacerPro query
Top level goal	$(concept - instances\ TopLevelGoal)$
Support goals	$(concept - instances\ SupportGoal)))$
Evidence supporting goal g_2	$(retrieve - individual - fillers\ g_2\ hasEvidence)$
Undeveloped Goals	$(concept - instances\ UndevelopedGoals)$
Generate OWL	$(save - kb"PATH/kb.owl"\ : syntax : owl)$
Check if Abox is consistent	$(abox - consistent?)$
Get all contexts of a specific goal	$(individual - fillers\ g_1\ inContextOf)$

2. Each "leaf" node should either evidence or a reference to some previously reviewed assurance case

3. Circular reasoning: identified by the RacerPro engine in the form of cycle concepts

Validating the Diagram Scenario

The flow of this action is represented in figure 4.9

Below is presented the steps necessary, without interruptions, for validating using Racer the selected diagram.

Happy flow:

1. User opens or creates a diagram

2. User select from ProjectExplorer tab the selected diagram

3. Selects validation of the wanted diagram

4. Systems returns the results in validation.log file

In the extended flow are presented the interruptions that might occur when trying to complete this action.

Extended flow:

3.1 File containing diagram translation is not found

3.1.1. System notifies the user that racer file was not found

3.1.2. The user selects translating the diagram

3.1.3. User ree-selects validation

3.2 Racer engine is not started

3.2.1. System notifies the user that RacerPro was not found

3.2.2. The user starts manually the engine

3.2 Diagram is not loaded

3.2.1. System notifies the user that diagram is not loaded

3.2.2. The user loads the diagram in racer engine

4.5.3 Generation of Safety Case Metrics

Complementarily to supporting semantic reasoning, our system provides also quantitative assessment of a safety case through several metrics developed.

The metrics are developed with the LISP API of RacerPro system. For instance, the number of not verified goals for safety case given as the ABox $sc1$ is computed with:

$(length\ (concept - instances\ NotVerifiedGoal))$

or the number of undeveloped goals:

$(length\ (concept - instances\ UndevelopedGoal))$

The main use case of metrics is to assess the progress during different stages of validating the safety case. Given large safety cases, one can monitor the rate to which the number individuals of type *NotVerifiedGoal* decreases.

4.5.4 Generating Natural Language Reports on the Safety Case

Our tool supports the generation of documentation and reports for the safety case. There will be generated three types: validation report, to do or assement report and diagram documentation.

The to-do reports contains things that still needs to be done, assessment and validation of the safety case. The validation report includes:

- nodes that do not have a description;
- elements that are not linked directly or indirectly through other elements of the diagram to the top level goal;
- goals that do not have evidence or solution;
- incomplete goals that have undeveloped sub-goals.

With this report the safety engineer knows at any moment what still needs to be added to the safety case to have a complete and well-build safety case. The assessment report provides also quantitative information of the diagram, in terms of number of nodes and their types. The documnation file will include the assessment report and description of every goal node of the diagram, including attributes and relations. Having the diagram and diagram documentation facilitate the work of the safety engineer or certification auditors.

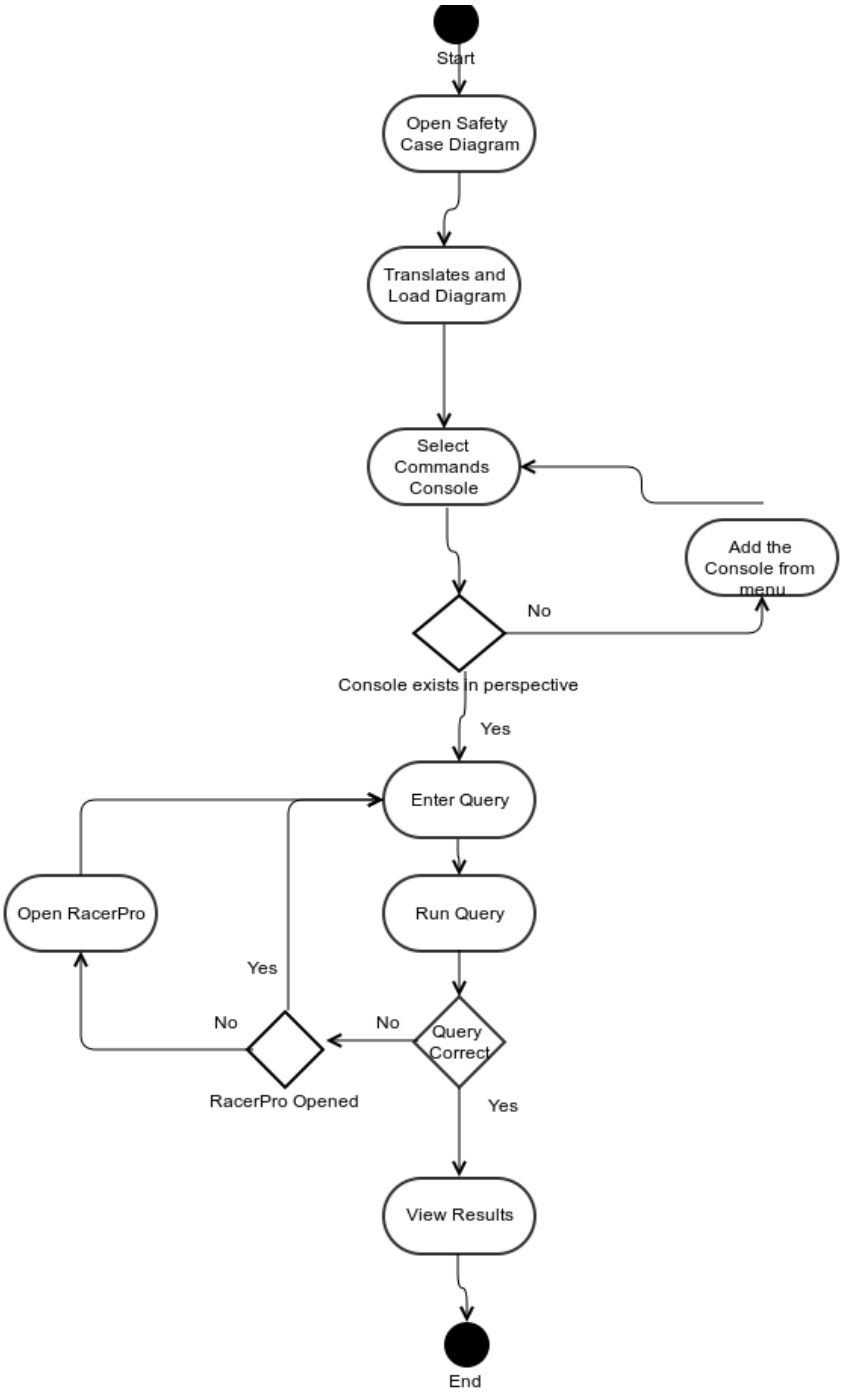

Figure 4.8: Flow Chart Querying the diagram

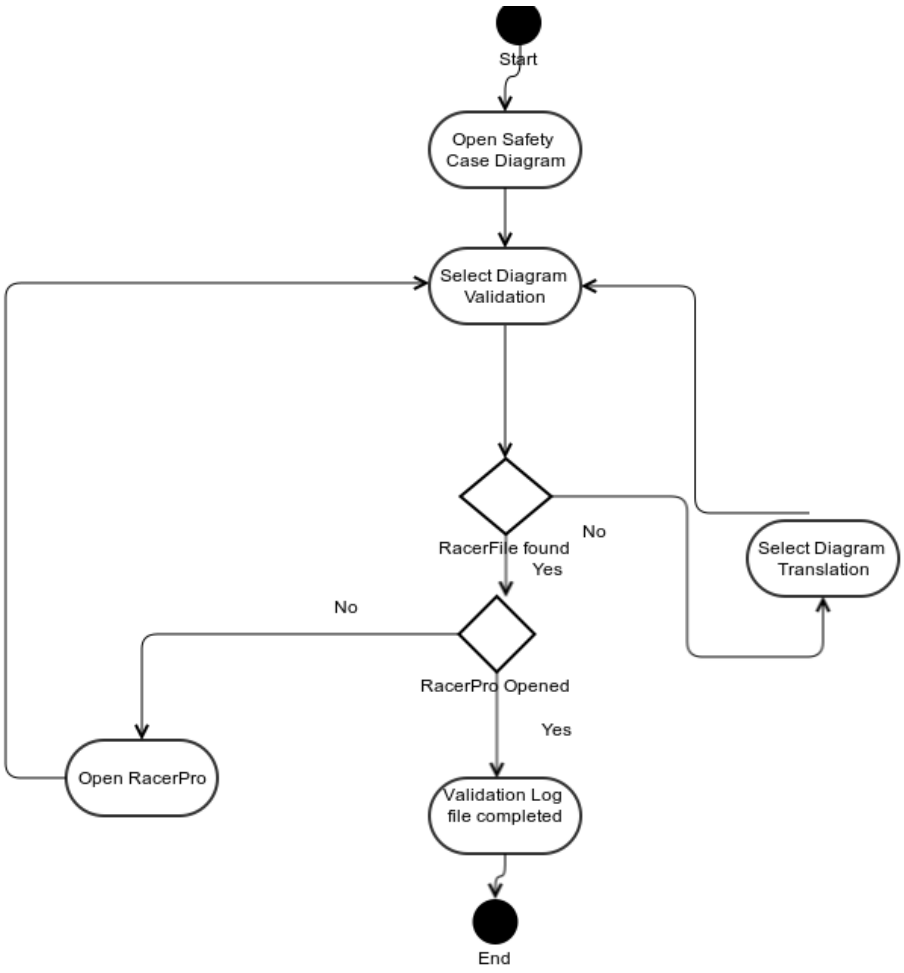

Figure 4.9: Flow Chart for validating the diagram

Chapter 5

Detailed Design and Implementation

Out tool is an improvement of Acedit [1] tool. New features like the option of querying the diagram, generation of reports and documentation, usage of safety metrics for the assessment of the safety cases have been added and others tool features have been improved, for example the validation of a diagram.

5.1 System Architecture

The tool consist of a set of Eclipse plug-ins.

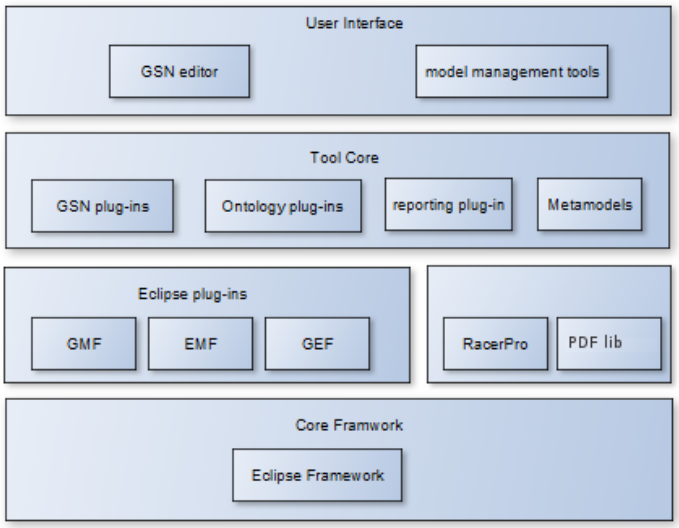

Figure 5.1: System Archtecture

As presented in figure 5.1, the tool is structured on layers as follows:

- **first layer:** is the one at the bottom and consists of the core framework of the tool;

- **second layer:** consists of several eclipse plug-ins used to implement the tool (EMF, GMF, GEF) plus the libraries used for establishing connections with RacerPro engine and NaturalOwl engine;
- **third layer:** contains the GSN and ARM metamodels, plus tool plug-ins through which all tool functionality is provided;
- **last layer:** represents the user interface layer and consists of the GSN editor and the model management tools(GSN editing wizards);

From figure 5.2 can be seen that we kept the main component of the ACedit, consisting of the GSN plug-ins regarding the graphical editor, eliminated the validation plug-in and we added a new set of plug-ins which provides the semantic reasoning facility, validation and the others features.

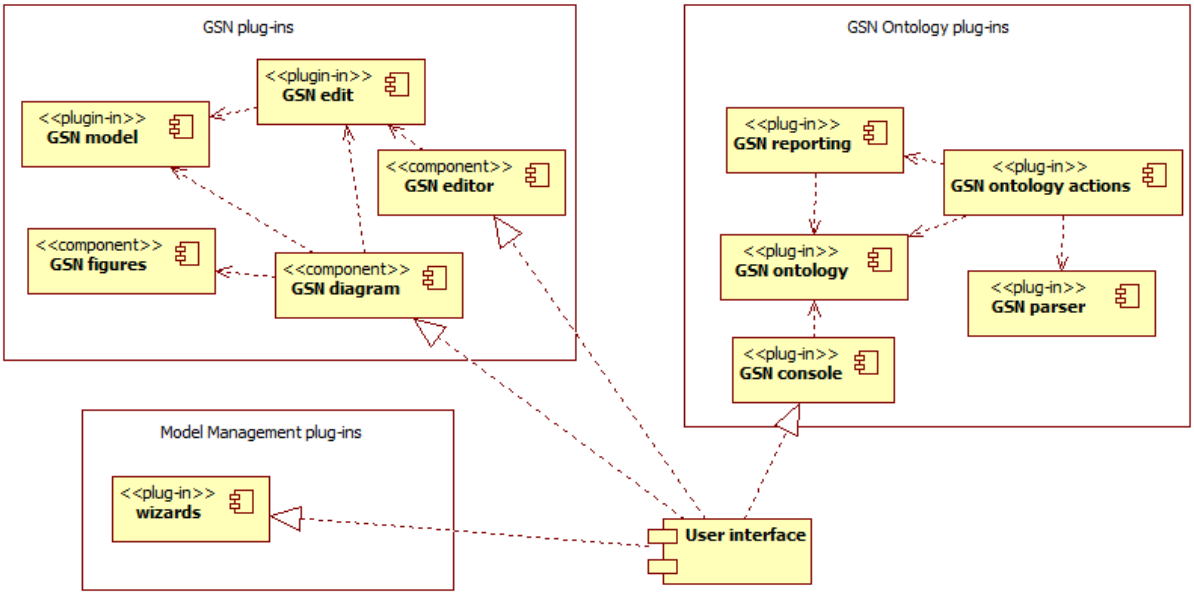

Figure 5.2: Component Diagram

5.1.1 GSN editor

The GSN editor has two views:

- the diagram view: deals with creation of GSN diagram accordingly to the mapping between metamodels and their graphical notation
- the model view: deals with creation of GSN metamodels

For communication between this two views it has been used the Model View Controller pattern as follows: the model part of the MVC is represented by the model view and is the GSN model instance created by the user, the view part is represented by the diagram view,i.e. the editor's GUI, and the controller represents the logic implemented in the others plug-ins. This communication is alsoo represented in figure 5.3

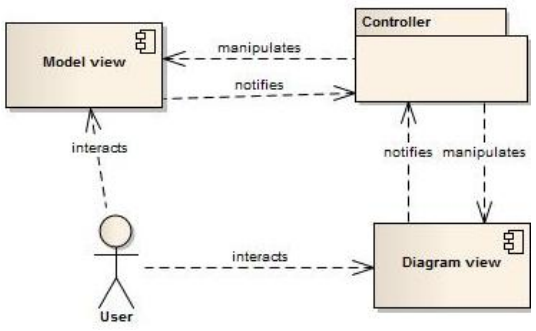

Figure 5.3: Component Diagram

5.2 Implementation details

5.2.1 GSN plug-ins

In this set of plug-ins is implemented GSN editor functionality and interface using specific eclipse frameworks designed for this:

Eclipse Modeling Framework (EMF) is a Java framework and code generation facility provided by Eclipse for building tools based on a domain model. The framework provides the EMFatic language with which the domain models can be turned into efficient, correct and easily customizable code. EMFatic language is similar to Java syntax and is designed to support navigation and editing of Ecore models. In our case, the GSN metamodels are defined using the EMFatic language.

Graphical Editing Framework (GEF) is a framework provided by Eclipse for creating graphical editors for various diagrams. It is used to represent graphically a safety case and also it permits the addition and manipulation of a safety case, providing a visual representation of the relationships between the safety case model elements.

Graphical Modelling Framework (GEF) is an Eclipse framework used for developing graphical editors based on EMF and GEF. This has been used to develop the runtime infrastructure of our tool.

5.2.2 GSN ontology plug-ins

The **GSN console plug-in** contains the implementation of the console used to run queries on the diagram called Command Console. It consist of two packages, gsn1.console is the package implementing the console and gsn1.racercommands is the one which deals with the processing and execution of the commands.

We create a new type console called "Commands Console" implemented in class CommandConsole. This class extends the IOConsole class provided in org.eclipse.ui.console package. The console will display text from the I/O streams. It can have multiple output streams connected to it and only one input stream connected to the keyboard.

In the plugin.xml file we declare a new extension point that contributes to the org.eclipse.ui.console.consoleFactories factory, extending that plug-in with the console factory class called Factory. The console factory extension is responsible for openining a console in the console view. Extensions appear on a menu in the console view and are opened when their openConsole method is called. The gsn.console package that contains the console was developed so that it can be re-used by developers for building I/O consoles having other purposes.

For processing the commands and queries from the console we applied the Factory pattern. There is the factory class called CommandFactory and the interface ICommand implemented by each command. The StartCommand class deals with starting a new RacerPro process if at that moment there's no process of the engine. The QueryCommand class is used to send a query to the racer engine and process the results.

The class diagram of this plug-in can be seen in figure 5.4.

The **GSN actions plug-in** adds in the diagram's file menu all the actions that can be done with the file: transforming into A-box, validate it, load it in the racer engine, generate to-do report and documentation. It is composed of two packages, one is containing the popup actions and the second one contains their core functionality of the popup actions from the menu. The actions have a corresponding class from the gsn.actions.package and action in the extension point extending the org.eclipse.ui.popupMenus class from the plugin.xml file in order to be visible in the view. Their core functionality has been separated from the interface, all classes being implemented in the package gsn.actions.core. All the 3 types of actions for creating reports use the same class DocumentationActionCore which has a separated method for each one of them because they don't have the same template. All the others actions are linked to a class implementing the action functionality.

In figure 5.6 are represented all the classes and the relations between them from this plug-in.

The **GSN parser plug-in** is the eclipse plug-in used to translate the builded safety case diagram into the A-box part. We have a class called XMLparser that has methods for parsing each node from the diagram-model file(which is a xml file) transforms it into a concept instance or relation between two instances or adds an attribute and at the end saves all data into a file representing the an A-box. Because each node of the diagram is

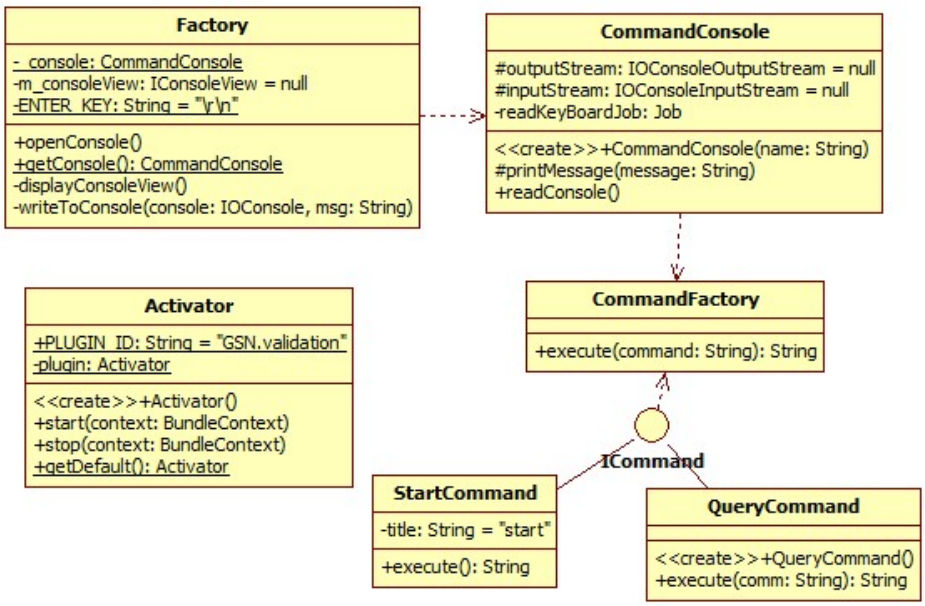

Figure 5.4: Class diagram for console plug-in.

represented through generated ids at not by the node identifier from diagram, when we parse the diagram_model file create Shape objects and save only the nodes that are shape or shape attribute, if we find connectors that depending on the ids representing the targets we search for those shapes and add the connection at the parent shape. This class uses org.w3c.dom and javax.xml.parsers libraries for parsing the XML file.

The Ontology class is contains methods that build the syntax to be added in the Abox file, depending on the node type.

In figure 5.6 are represented all the classes and the relations between them from this plug-in.

The **GSN ontology plug-in** is implementing the communication of our tool with the RacerPro engine.

The class called OntologyConnection is build following Singleton pattern, and is used to create a racer client and establish a connection to the racerPro reasoner using the JRacer java library provided by Racer.

For creating the connection we need to check if the engine is started, for this we have the class RacerProcess which checks if there exist any process of the engine, if not then locates the execution file of the racer and starts a new process. If no file has been found then the user will be shown a message with the location from where he can download the engine.

This OntologyConnection class makes it possible interrogating the buid A-box file, mainly by receiving the text command from the console and calling answerQuery(String

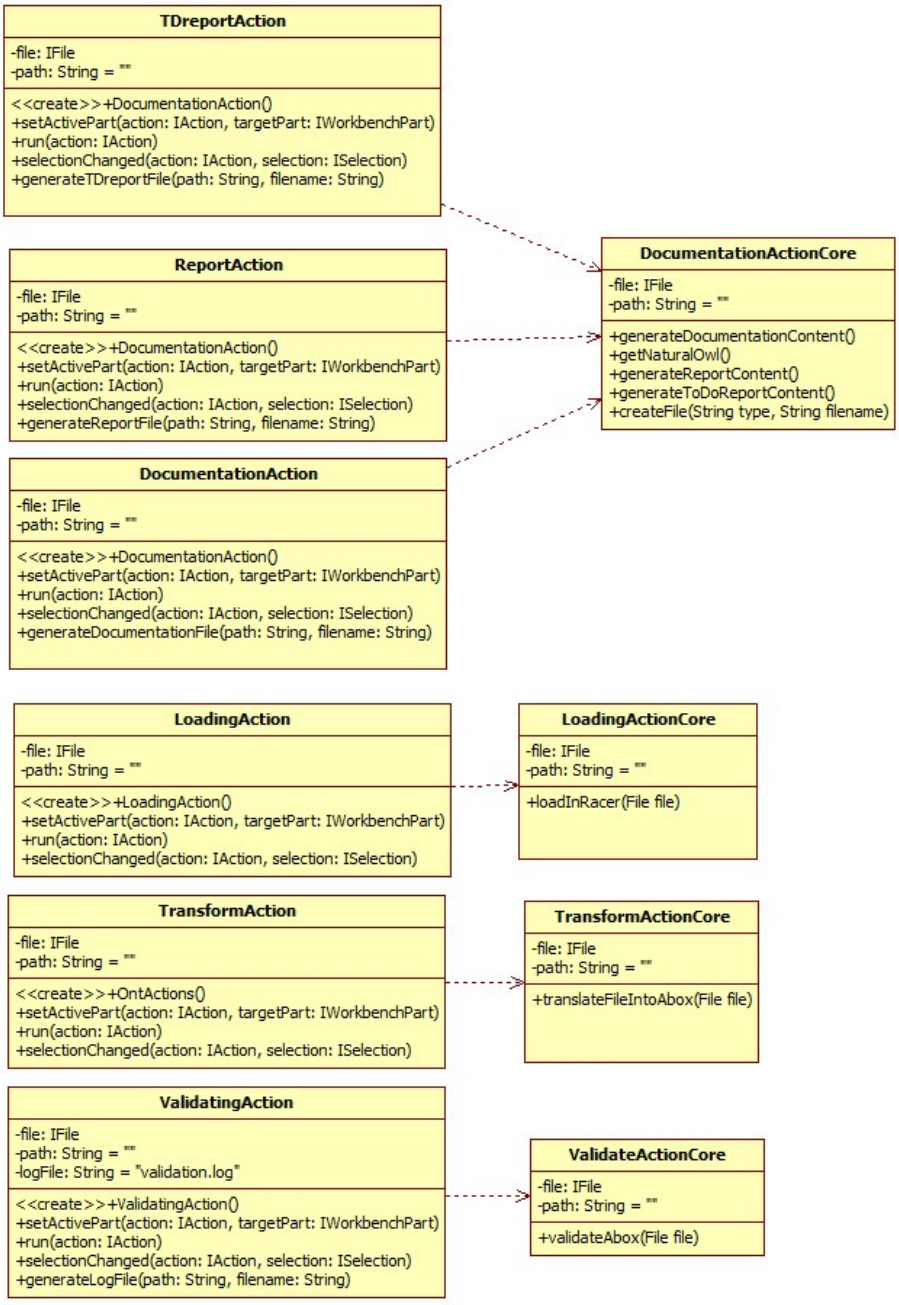

Figure 5.5: Class diagram for ontology actions plug-in.

command) method which will return the result of the sent query. Also this class has the method getTboxFile() which instantiates an GSNTbox object and aks for creating the t-box file modeled on GSN standard.

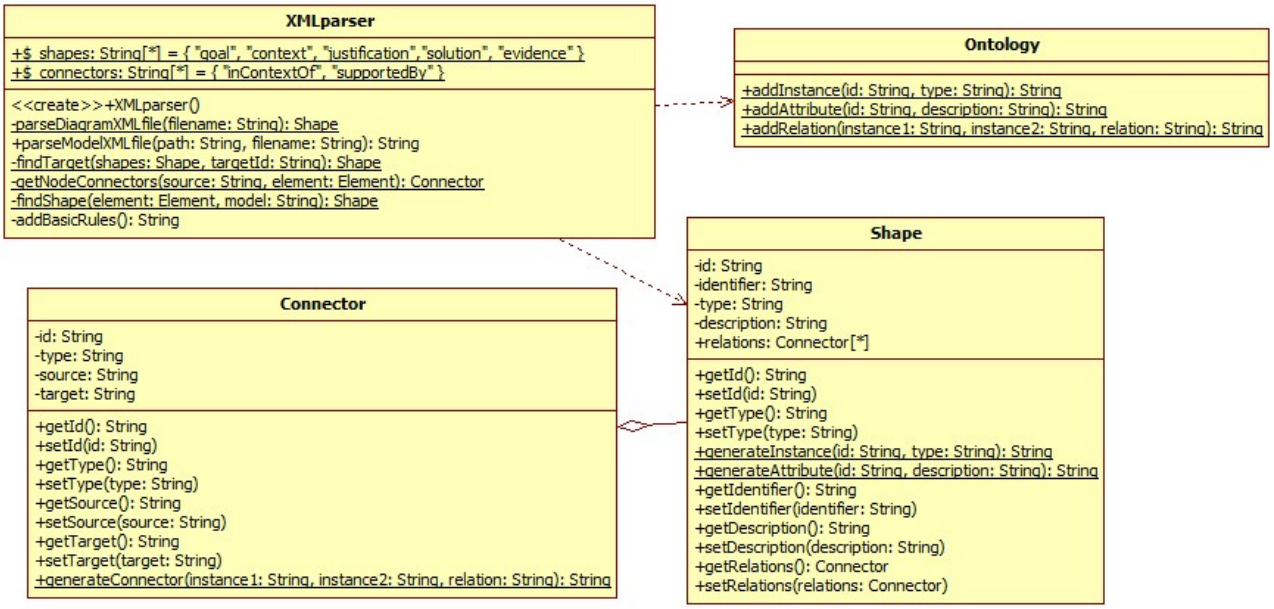

Figure 5.6: Class diagram for parser plug-in.

The GSNTbox class is a class that builds the T-box part of the ontology. It has static methods where all the concept, roles and attributes are defined, returning list of strings that are merge in the getContent() method of the class. A new file is generated with all this info and it will be saved at running in the eclipse main folder.

In figure 5.7 are represented all the classes and the relations between them from this plug-in. This plug-in is used in all the other plug-ins excepting the parser plug-in.

The **GSN reporting plug-in** contains three types of reports: validation, ToDo and documentation reports generated in natural language. All three reports are pdf type and are generated using iText library. This library is used to creare, adapt, inspect and maintain pdf files.

The data from the reports is taken using Lisp Racer Cilent library provided by Racer is used for generating metrics in the reports and by directly interogating the A-box with fixed queries depending on each report type. Each report has a different template.

The validation report contains the same output of the validation action, the difference is that it is saved in a new file and not in a log file.

The To do report contains what has been done, what still needs to be done and what is missing from the safety case diagram.

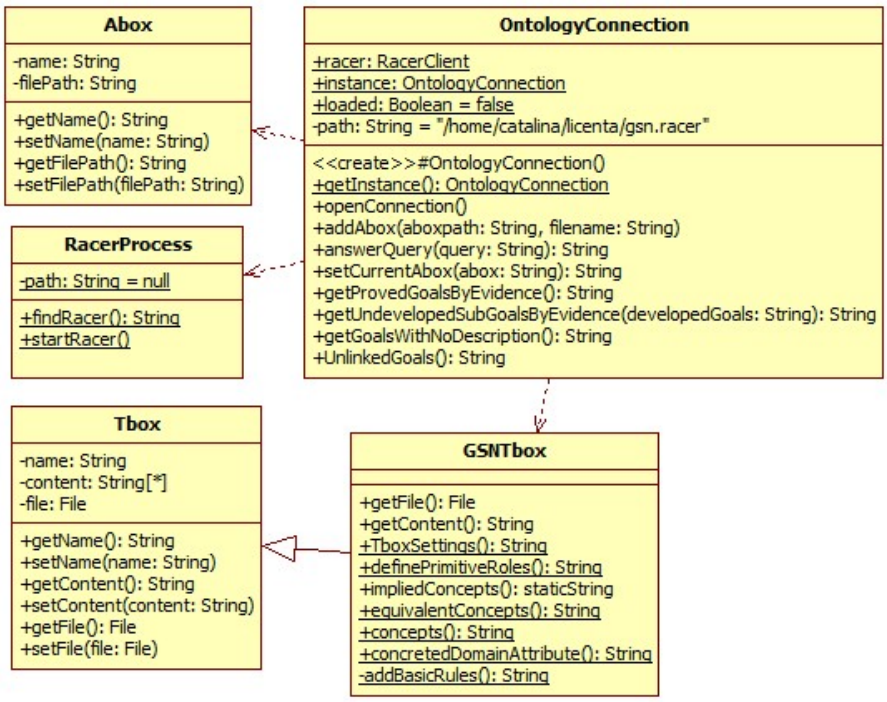

Figure 5.7: Class diagram for ontology plug-in.

5.3 User Interface

The workspace of the system is presented in Fig. 5.8. A safety project (top-left) consists of several assurance cases, developed either as a graphical diagram (files with gsn extension) or as an abox in description logic (files with racer extension). In case of need the system automatically translated between these two input formats. For a selected diagram file the user can transform into abox, validate the diagram and generate reports.

The main window (top-cencer) depicts the active gsn diagram. The elements of the GSN standard are represented as follows: goals with rectangular, strategies with paralelograms, evidence and solutions are represented by circle, assumptions and justifications with ellipse, context by a rectangular with rounded corners, the supportedBy relation is an arrow with the head filled, while the inContextOf is represented by an arrow with empty head.

The title and description of a node can be entered by clicking on the node in the head part for the title, and in the field with the placeholder ‘description‘. The diagram is constructed by using a drag-and-drop pallet (top-right).

The built-in actions performed on the diagram are visible when clicking right on the diagram or model file.

The command console (bottom-center) shows the reasoning performed on the active

diagram above. In the command line, specific queries for interrogating ontologies can be added and the reasoning engine will return the results for each query. The syntax of the queries corresponds to the RacerPro tool. In Fig. 5.8, the four queries exemplified are:

- retrieving all the goals in the diagram,
- identifying the top level goal,
- listing all pieces of evidence supporting the goal g_2, and
- checking the consistency of a diagram with respect to the GSN standard encoded as axioms in description logic.

In the bottom-left corner the red rectangle represent the view part of the diagram visible in the main window.

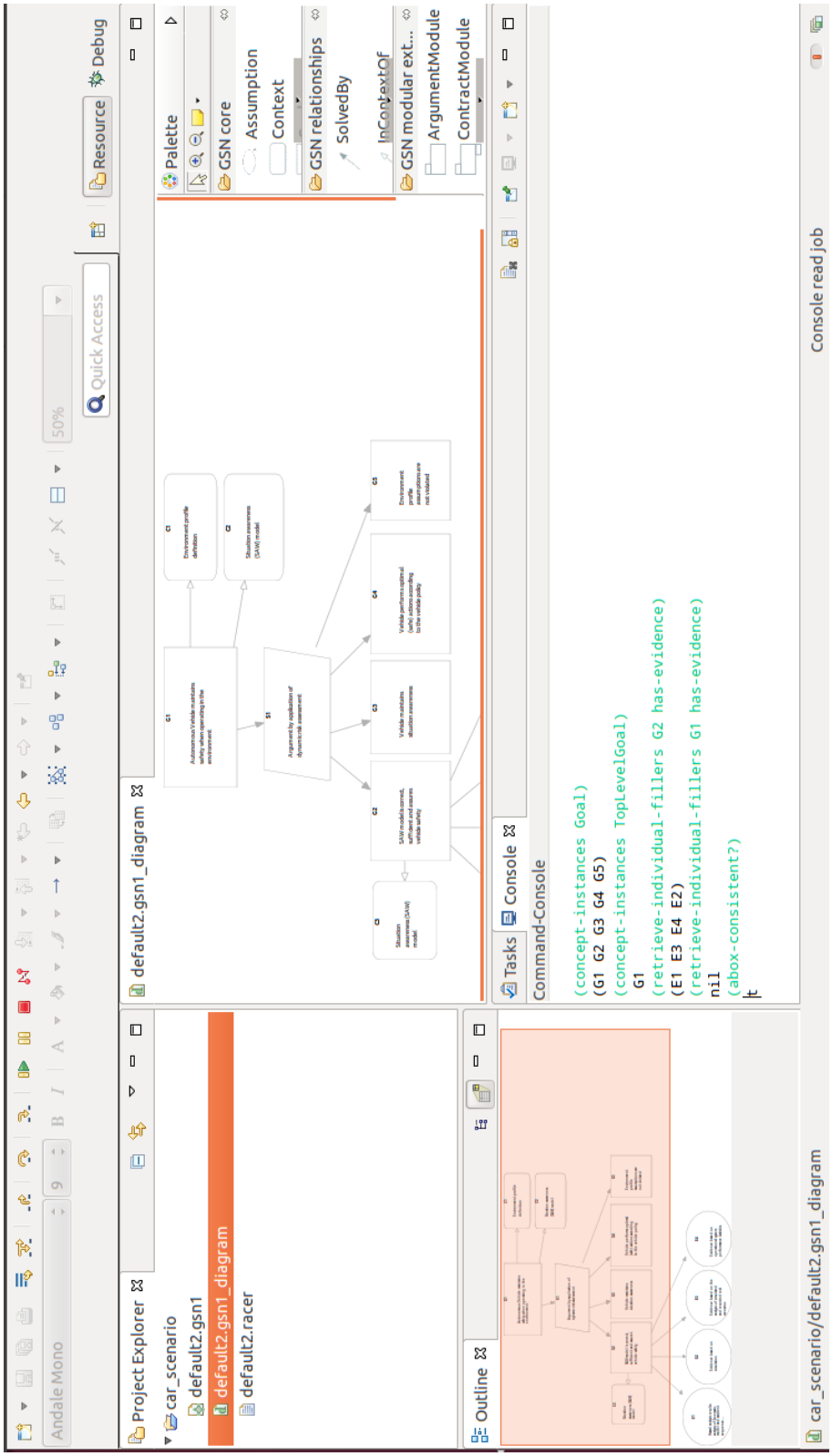

Figure 5.8: Application Interface

Chapter 6

Testing and Validation

6.1 Testing the tool

To test if the application has the desired behaviour and it is well build, we have taken well structured or incomplete safety cases from different domains, builded and validated within our tool. More details about each safety case can be found in the Bibliographic research chapter.

The first safety scenario is taken from from the autonomous driving domain and represents in the case is claimed that any autonomous vehicle should ensure safety when operating in the environment. A GSN diagram built in our *SafeEd* tool is represented in Fig. 6.1.

The top level goal g_1 states that any autonomous vehicle should ensure safety when operating in the environment. The goal holds in two contexts: the existence of an environment formalisation (context c_1), respectively the existence of a mechanism providing situation awareness. One solution for ensuring safety is dynamic risk assessment approach [9]. The corresponding strategy s_1 used to support the goal g_1 is to dynamically assess the risk. The sub-goals g_2, g_3, g_4, and g_5 are used to fulfill the strategy s_1. For instance, the sub-goal g_2 claims the correctness of the model, statement that is supported by various pieces of evidence, including formal verification e_2

The diagram from Fig. 6.1 is translated into the description language in Figure 6.2 and the result of translation into Abox using Racer syntax by our tool is shown in figure Figure 6.3. Here, the facts f_{61} to f_{64} assert the individuals to their corresponding GSN core elements. The structure of the GSN diagram based on the two relationships *supportedBy* and *inContextOf* is formalised by the facts f_{65} to f_{72}. The natural language text describing claims, solutions, contexts or evidences are encapsulated as concrete attributes [5] in Racer syntax (assertions f_{73} to f_{80}).

The table 6.1 contains the results of some basic queries run on the abox from 6.2. It can be seen that the obtained results are very helpful because it spares the safety engineer time which has to look for manually for them if such an assesment feature, like querying the diagram, is not available.

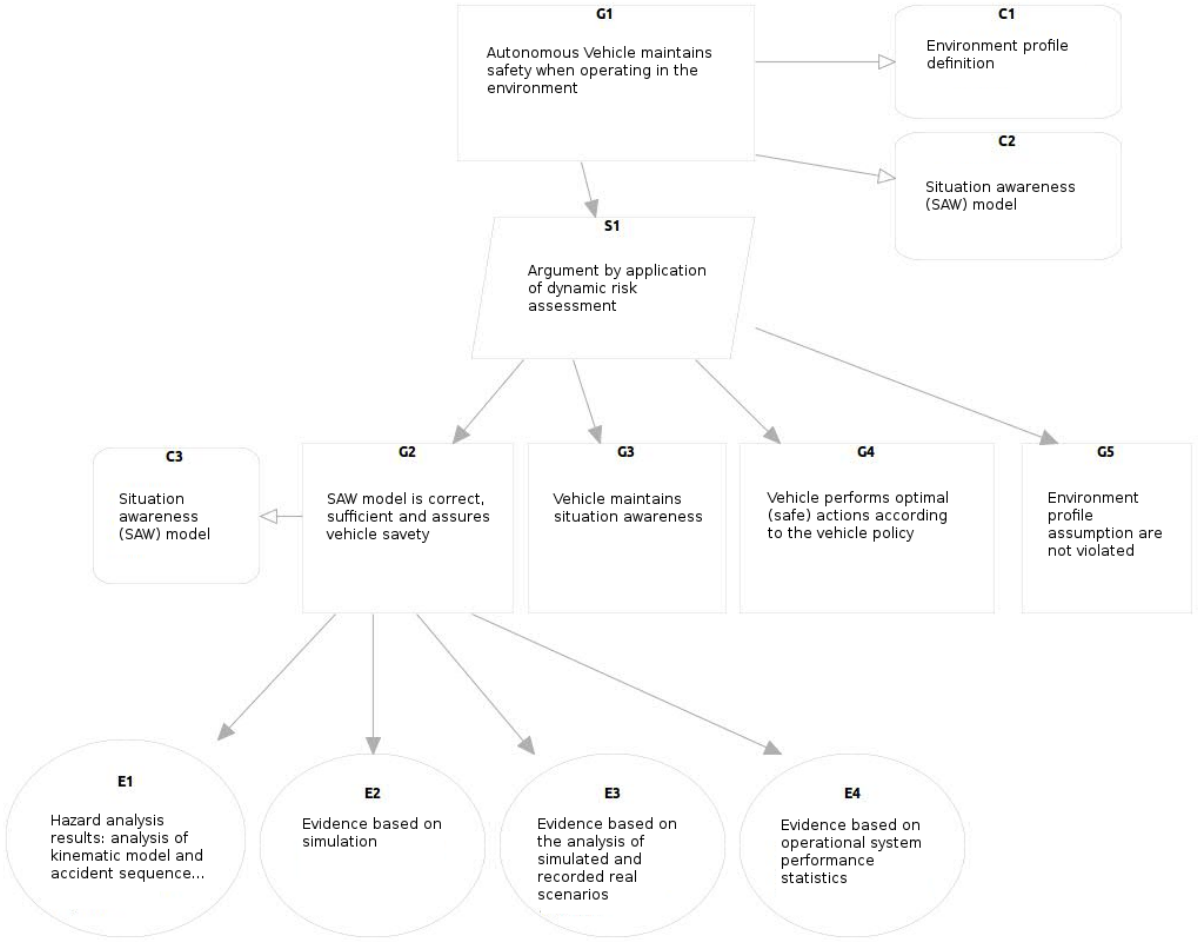

Figure 6.1: Autonomous vehicle scenario.

Table 6.1: Retrieving some information about the safety case represented in figure 6.1.

Query	RacerPro query	RacerPro answer
Top level goal	$(concept - instances\ TopLevelGoal)$	g_1
Support goals	$(concept - instances\ SupportGoal)))$	g_2, g_3, g_4, g_5
Evidence supporting goal g_2	$(retrieve - individual - fillers\ g_2\ has - evidence)$	e_1, e_2, e_3, e_4
Undeveloped Goals	$(concept - instances\ UndevelopedGoals)$	g_3, g_4, g_5
Check if Abox is consistent	$(abox - consistent?)$	
Get all contexts of a specific goal	$(individual - fillers\ g_1\ inContextOf)$	c_1, c_2

The second safety case is taken from medical industry and refers to the Therac 25, a case study in safety failure. In figure 6.4 is a diagram build wit our tool representing the case of Therac 25 presented in [12].

(f_{61}) $g_1 : Goal,\ g_2 : Goal,\ g_3 : Goal,\ g_4 : Goal,\ g_5 : Goal$
(f_{62}) $c_1 : Context,\ c_2 : Context, c_3 : Context$
(f_{63}) $e_1 : Evidence,\ e_2 : Evidence, e_3 : Evidence, e_4 : Evidence$
(f_{64}) $s_1 : Context,\ c_2 : Context, c_3 : Context$
(f_{65}) $(g_1, s_1) : supportedBy$
(f_{66}) $(g_1, c_1) : inContextOf$
(f_{67}) $(g_1, c_2) : inContextOf$
(f_{68}) $(g_2, c_3) : inContextOf$
(f_{69}) $(g_2, e_1) : has - evidence$
(f_{70}) $(g_2, e_2) : has - evidence$
(f_{71}) $(g_2, e_3) : has - evidence$
(f_{72}) $(g_2, e_4) : has - evidence$
(f_{73}) $(g_1,$ "*Autonomous Vehicle maintains safety when operating in the environment*") : *hasText*
(f_{74}) $(g_2,$ "*SAW model is correct, sufficient and assures vehicle safety*") : *hasText*
(f_{75}) $(g_3,$ "*Vehicle maintains situationawareness*") : *hasText*
(f_{76}) $(g_4,$ "*Vehicle performs optimal (safe) actions according to the vehicle policy*") : *hasText*
(f_{77}) $(g_5,$ "*Environment profile assumptions are not violated*") : *hasText*
(f_{78}) $(s_1,$ "*Argument by application of dynamic risk assessment*") : *hasText*
(f_{79}) $(e_1,$ "*Hazard analysis results : analysis ofkinematic model and accident sequence*") : *hasText*
(f_{80}) $(e_2,$ "*Evidence based on simulation*") : *hasText*
(f_{81}) $(e_3,$ "*Evidence based on the analysis of simulated and recorded real scenarios*") : *hasText*
(f_{82}) $(e_4,$ "*Evidence based on operational system performance statistics*") : *hasText*
(f_{83}) $(c_1,$ "*Environment profile definition*") : *hasText*
(f_{84}) $(c_2,$ "*Situation awareness (SAW) model*") : *hasText*
(f_{85}) $(c_3,$ "*Situation awareness (SAW) model*") : *hasText*

Figure 6.2: Translating the GSN diagram in a description logic Abox.

```
(in-abox default2.racer GSN)
(instance G1 Goal)
(attribute-filler G1 "Autonomous Vehicle maintains safety when operating in the environment" claims)
(related G1 S1  supportedBy)
(related G1 C1  inContextOf)
(related G1 C2  inContextOf)
(instance G2 Goal)
(attribute-filler G2 "SAW model is correct,sufficient and assures vehicle safety" claims)
(related G2 E2  has-evidence)
(related G2 E4  has-evidence)
(related G2 E3  has-evidence)
(related G2 E1  has-evidence)
(related G2 C3  inContextOf)
(instance G3 Goal)
(attribute-filler G3 "Vehicle maintains situation awareness" claims)
(instance G4 Goal)
(attribute-filler G4 "Vehicle performs optimal (safe) actions according to the vehicle policy" claims)
(instance G5 Goal)
(attribute-filler G5 "Environment profile assumptions are not violated" claims)
(instance S1 Strategy)
(attribute-filler S1 "Argument by application of dynamic risk assessment " claims)
(related S1 G2  supportedBy)
(related S1 G3  supportedBy)
(related S1 G4  supportedBy)
(related S1 G5  supportedBy)
(instance C1 Context)
(attribute-filler C1 "Environment profile definition" claims)
(instance C2 Context)
(attribute-filler C2 "Situation awareness
(SAW) model" claims)
(instance  Assumption)
(instance C3 Context)
(attribute-filler C3 "Situation awareness (SAW) model" claims)
(instance E1 Evidence)
(attribute-filler E1 "Hazard analysis results: analysis of kinematic model and accident sequence" claims)
(instance E2 Evidence)
(attribute-filler E2 "Evidence based on simulation" claims)
(instance E3 Evidence)
(attribute-filler E3 "Evidence based on the analysis of simulated and recorded real scenarios" claims)
(instance E4 Evidence)
(attribute-filler E4 "Evidence based on operational system performance statistics" claims)
(realize-abox)
(set-current-abox default2.racer GSN)
```

Figure 6.3: Translating the GSN diagram in Abox using our tool.

The main statement is represented by top level goal $Goal_1$ and states that the machine is fault free and can be used. Two strategies have been adopted for proving this goal: the first strategy represented by $Strategy_1$ proposes argumentation by satisfaction of system's safety, while the second strategy represented by $Strategy_2$ proposes argumentation by omission of all identified software hazards supposing we have identified the software hazards, context represented by node $Context_1$. The first strategy is divided into two sub-claims: the system will halt in case of errors, statement represented by $Goal_2$, and that exceeding radions limit will abort the operation, statement represented by $Goal_3$. $Goal_2$ is supported by $Goal_4$ which claims that when a fault occurs the system cannot be resumed, a solution to this would be Black Box testing($Solution_1$) Both $Goal_2$ and $Goal_3$ are supported by the $Goal_5$ which claims that any failure operation will include explanatory error message, a solution for this is declaring and using safe states($Solution_2$). The

second adopted strategy is solved by two statements: computer giving a wrong amount of energy occurs as a result of component failure, $Goal_6$, and computer selecting wrong mode can only occur as a result of component failure, $Goal_7$. Both goals are solved by: making fault tree analysis, $Solution_3$ and analysing the results of hazard test, $Solution_4$.

This case could have been applied when developing the Therac 25 system, leading to discovering of several faults that could have saved lives.

The diagram in Fig. 6.5 is translated into the Abox represented in Fig. 6.2. Here, the facts assert the individuals to their corresponding GSN core elements. The structure of the GSN diagram based on the relationships $has - evidence$, $supportedBy$ and $inContextOf$. The natural language text describing claims, solutions, contexts or evidences are encapsulated as concrete attributes [5] in Racer syntax.

The table 6.2 contains the results of some basic queries run on the abox from 6.5. It can be seen that the obtained results are very helpful because it spares the safety engineer time which has to look for manually for them if such an assesment feature, like querying the diagram, is not available.

Table 6.2: Retrieving some information about the safety case represented in figure 6.4.

Query	RacerPro query	RacerPro answer
Top level goal	$(concept - instances\ TopLevelGoal)$	$goal_1$
Support goals	$(concept - instances\ SupportGoal)))$	$goal_2, goal_3, goal_4,$ $goal_5, goal_6, goal_7$
Evidence supporting goal $goal_7$	$(retrieve - individual - fillers$ $goal_7\ has - evidence)$	$solution_3,$ $solution_4$
Undeveloped Goals	$(concept - instances\ UndevelopedGoals)$	nil
Check if Abox is consistent	$(abox - consistent?)$	t
Get all contexts of a specific goal	$(individual - fillers\ strategy_2\ inContextOf)$	$context_1$

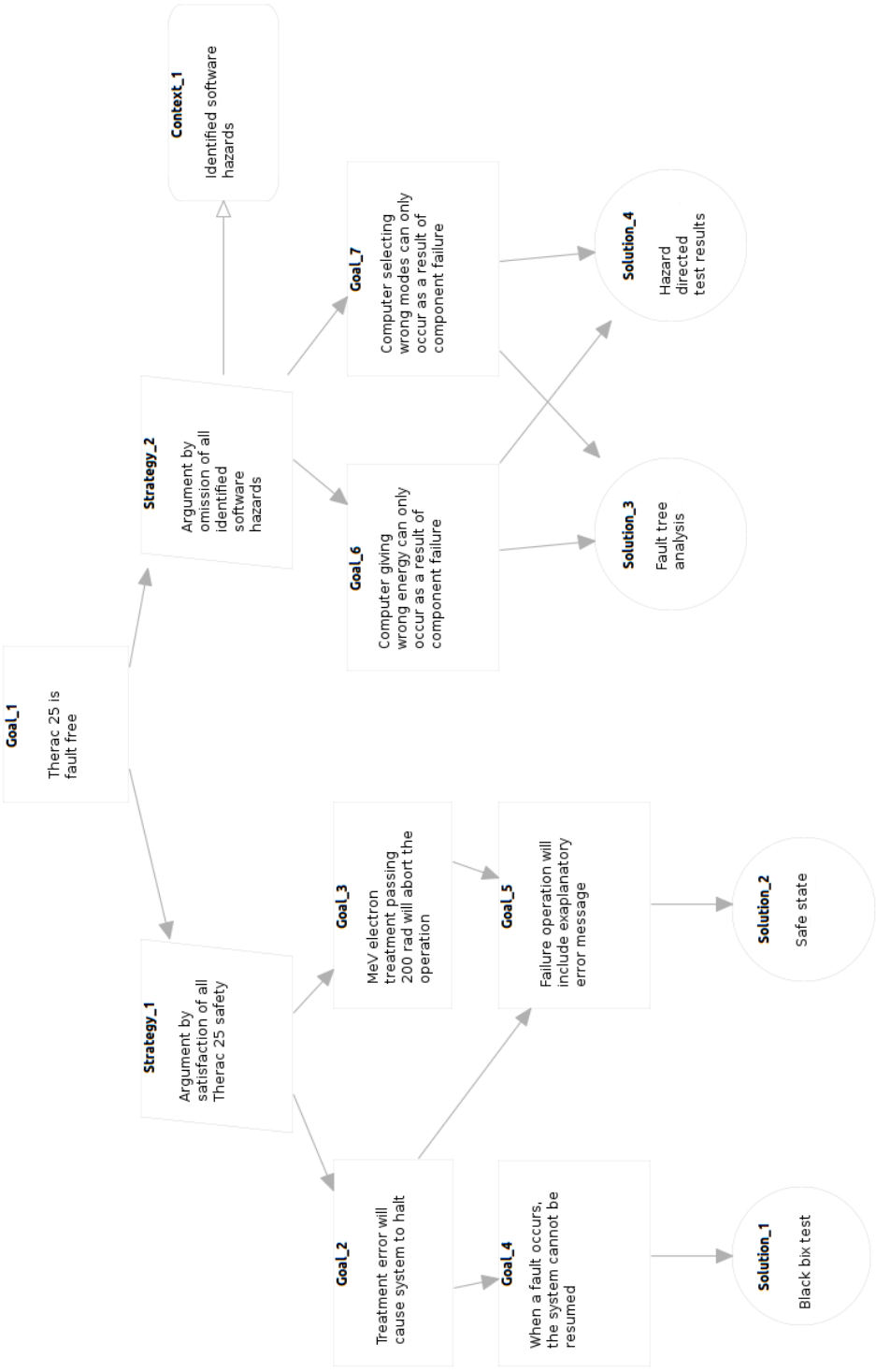

Figure 6.4: Therac 25 safety case.

$(in - abox\ default.racer\ GSN)$
$(instance\ Goal_1\ Goal)$
$(related\ Goal_1\ Strategy_1\ supportedBy)$
$(related\ Goal_1\ Strategy_2\ supportedBy)$
$(instance\ Strategy_1\ Strategy)$
$(related\ Strategy_1\ Goal_2\ supportedBy)$
$(related\ Strategy_1\ Goal_3\ supportedBy)$
$(instance\ Strategy_2 Strategy)$
$(related\ Strategy_2\ Goal_6\ supportedBy)$
$(related\ Strategy_2\ Goal_7\ supportedBy)$
$(related\ Strategy_2\ Context_1\ inContextOf)$
$(instance\ Context_1\ Context)$
$(instance\ Goal_2\ Goal)$
$(related\ Goal_2\ Goal_4\ supportedBy)$
$(related\ Goal_2\ Goal_5\ supportedBy)$
$(instance\ Goal_3\ Goal)$
$(related\ Goal_3\ Goal_5\ supportedBy)$
$(instance\ Goal_6\ Goal)$
$(related\ Goal_6\ Solution_3\ has - evidence)$
$(related\ Goal_6\ Solution_4\ has - evidence)$
$(instance\ Goal_4\ Goal)$
$(related\ Goal_4\ Solution_1\ has - evidence)$
$(instance\ Goal_7\ Goal)$
$(related\ Goal_7\ Solution_3\ has - evidence)$
$(related\ Goal_7\ Solution_4\ has - evidence)$
$(instance\ Goal_5\ Goal)$
$(related\ Goal_5\ Solution_2\ has - evidence)$
$(instance\ Solution_1\ Evidence)$
$(instance\ Solution_2\ Evidence)$
$(instance\ Solution_3\ Evidence)$
$(instance\ Solution_4\ Evidence)$

Figure 6.5: Translating the Therac 25 safety case in a description logic Abox.

The third safety case is for a whole-airspace system divided into several georgraphical regions plus a region of en-route airspace for which ATM rules are applied and implemented on an are-by-area basis. Georgraphical regions that interacts with each other and also with the airspace-wide system. The safety case structure for the system is represented in figure 6.6. The main goal represented as Goal1 states that the airspace is safe for using, in the context of defining the safe term in case of the system. The strategy adopted by proving the goal is to prove that ATM rules are safe, respectively that the rules have been implemented safely. The second strategy is also divided in two: it is state that the rules are implemented safely in every area and each area assumption can't be violated. Evidence for each sub-goal is provided, as is shown in the diagram.

Figure 6.6 represents the translated Abox into Racer syntax, it can be seen that the goal structure is keep in the A-box.

Table 6.3: Retrieving some information about the safety case represented in figure 6.6.

Query	RacerPro query	RacerPro answer
Top level goal	$(concept - instances\ TopLevelGoal)$	$g1$
Support goals	$(concept - instances\ SupportGoal)))$	$g2, g3, g4, g5, g6, g7$
Evidence supporting goal $goal_7$	$(retrieve - individual - fillers\ goal_7\ has - evidence)$	$e3$
Undeveloped Goals	$(concept - instances\ UndevelopedGoals)$	nil
Check if Abox is consistent	$(abox - consistent?)$	t
Get all contexts of a specific goal	$(individual - fillers\ strategy_2\ inContextOf)$	nil

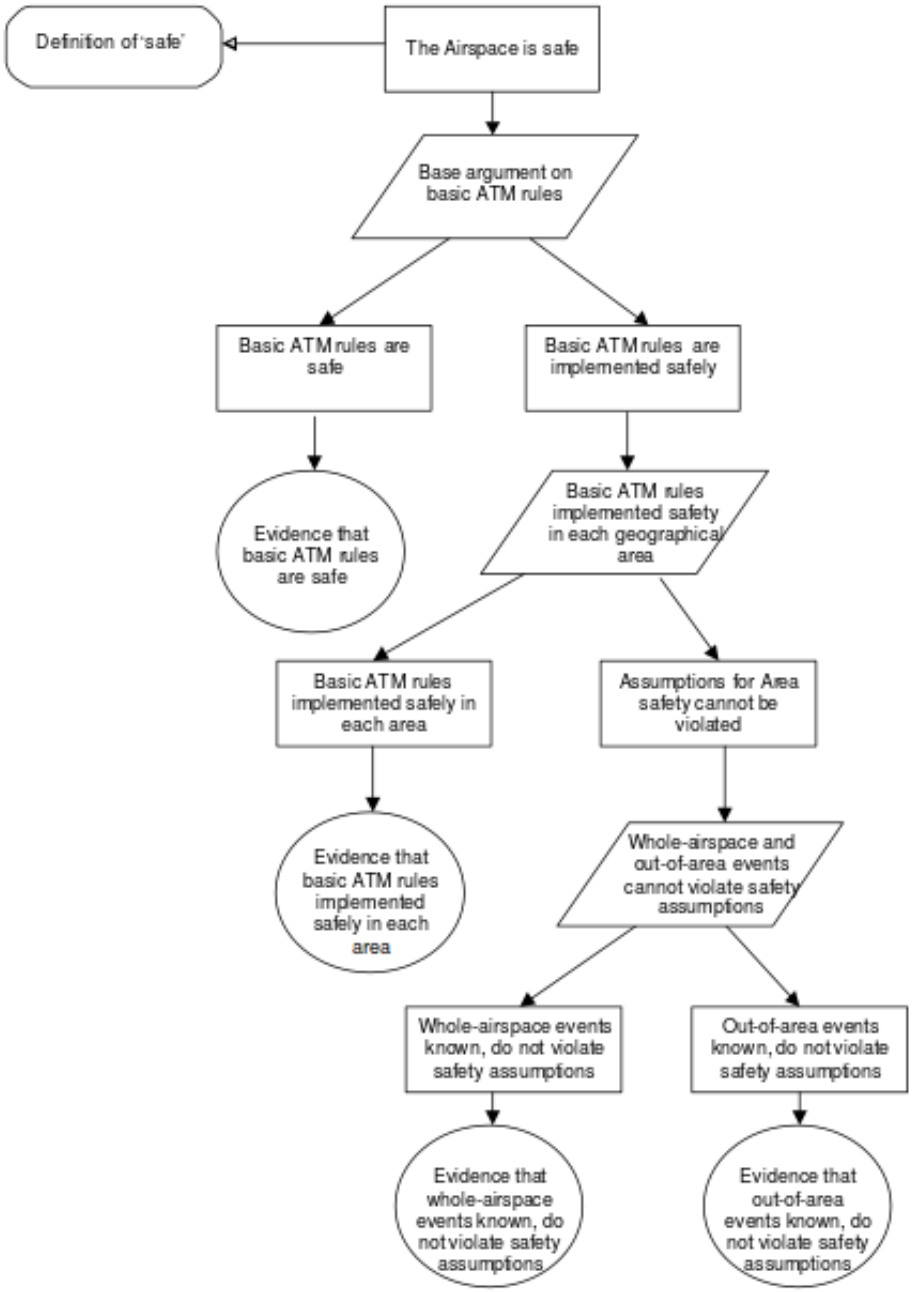

Figure 6.6: Airspace-system safety case.

In table 6.4 we presents the conclusion after building the safety cases presented above in our tool and two other tools. More detailed comparison between our tool and other tool can be found in Bibilographic Research Chapter. From the table we can resume that our tool is better at the management of a safety case, having a plus at validation,

```
(in-abox safetyATM.racer GSN)
(instance G1 Goal)
(attribute-filler G1 "The Airspace is safe" claims)
(related G1 S1  supportedBy)
(related G1 C1  inContextOf)
(instance S1 Strategy)
(attribute-filler S1 "Base argument on basic ATM rules" claims)
(related S1 G2  supportedBy)
(related S1 G3  supportedBy)
(instance G2 Goal)
(attribute-filler G2 "Basic ATM rules are safe" claims)
(related G2 E1  has-evidence)
(instance E1 Evidence)
(attribute-filler E1 "Evidence that basic ATM rules are safe" claims)
(instance G3 Goal)
(attribute-filler G3 "Basic ATM rules are implemented safely" claims)
(related G3 S2  supportedBy)
(instance S2 Strategy)
(attribute-filler S2 "Basic ATM rules implemented safety in each geographical area" claims)
(related S2 G4  supportedBy)
(related S2 G5  supportedBy)
(instance G4 Goal)
(attribute-filler G4 "Basic ATM rules implemented safely in each area " claims)
(related G4 E2  has-evidence)
(instance E2 Evidence)
(attribute-filler E2 "Evidence that basic ATM rules implemented safely in each area" claims)
(instance G5 Goal)
(attribute-filler G5 "Assumptions for Area safety cannot be violated" claims)
(related G5 S3  supportedBy)
(instance S3 Strategy)
(attribute-filler S3 "Whole-airspace and out-of-area events cannot violate safety assumptions" claims)
(related S3 G6  supportedBy)
(related S3 G7  supportedBy)
(instance G6 Goal)
(attribute-filler G6 "Whole-airspace events known, do not violate safety assumptions" claims)
(related G6 E3  has-evidence)
(instance E3 Evidence)
(attribute-filler E3 "Evidence that whole-airspace events known, do not violate safety assumptions" claims)
(instance G7 Goal)
(attribute-filler G7 "Out-of-area events known, do not violate safety assumptions" claims)
(related G7 E4  has-evidence)
(instance E4 Evidence)
(attribute-filler E4 "Evidence that out-of-area events known, do not violate safety assumptions" claims)
(realize-abox)
(set-current-abox safetyATM.racer GSN)
```

Figure 6.7: Airspace-system safety case translated in Abox using our tool

assesment, evaluation and reports generation. A minus would be the weak graphically representation of elements and importing/exporting diagrams from our tool.

The difference between our tool and this one is the fact that using ours the user will be able to build simple or complex queries for interrogating the diagram at any time during the development and not being limited only to metrics, this is a plus at the assessment of the diagram.

Table 6.4: Tools Comparison Table

Functionality	Acedit	AdvoCATE	Our Tool
Creating diagram	GSN models must be differentiated better in the interface	GSN elements are better represented (color is a plus)	GSN models must be differentiated better in the interface
Edit diagram	can edit only diagrams build with the tool	can edit other diagrams type	can edit only diagrams build with the tool
Import/Export other diagram types	can't import other diagrams types	can import other diagrams types from: D-case, Asce	can't import other diagrams types
Metrics	not available	Limited computation of metric	available using Racer lisp library
Diagram Assesment and evaluation	not available	all diagram nodes are saved on columns in CSV based on their type their type	Reasoning services used. Better,dynamic and more flexible solution than limitation of displaying on columns
Diagram Validation	based only on constraints	based only on constraints	based only on constraints plus digram consistency and missing elements
Reporting	not available available	CSV with nodes structured on columns	Three pdf reports: validation, to-do and documentation

Chapter 7

User's manual

7.1 System Installation

For using the tool, the user needs to have installed on the PC the following:

- Eclipse Distribution with Epsilon, this can be downloaded from here: http://www.eclipse.org/epsilon/download/
- RacerPro knowledge base reasoner on 32 bits
- NaturalOwl engine

To install the tool through eclipse updates:

1. Open the wizard and select from Eclipse top menu Install/Update->Availbale Software Sites
2. Click the Add.. button
3. If the sofware site is in your local file system click Local.. to specify tyhe directory location of the file.
4. If the software site is in your local file but packaged as a jar or zip, click Archive... to specify the name of the file.
5. Select Add Site...
6. Select all checkboxes
7. Click next
8. Accept Sofware license
9. Click finish

7.2 User's Manual

Creating a project and add diagram to it - steps:

1. select File->New->Projectselect General-> Project
2. select next and give a name to your project
3. select finish
4. press right click in the created project
5. select New->Other
6. in the pop up wizard type GSN
7. select GSN1 diagram
8. select Next-> Finish

This action will add a new project to the workspace and the diagram in it.

Transform diagram in A-box(after diagram file is created):

1. select diagram file
2. press right click on files
3. select Ontology
4. next select Transform A-box

The output of this action will be a new file having the same name as the diagram and the extension .racer located in the same project folder as the diagram. The file is not loaded in the Racer engine!

Load diagram (only if the racer file has been generated):

1. select diagram file
2. press right click on files
3. select Ontology
4. next select Load A-box

This action adds the A-box to the T-box in the Racer and will set it as main A-box.

Validate diagram:

1. select diagram file

2. press right click on files
3. select Ontology
4. next select Validate A-box

The action will write the results of the validation in validate.log file, this file will also contain the results of the previous validations of this or other files. An example of this report is shown in figure 7.1

```
==================06/04/14 08:48:20====================================
All the nodes have claims!

Evidence or solution must be provided for the following goals:
G3, G4, G5
G3 goal parents: G1
G4 goal parents: G1
G5 goal parents: G1

G1 goal has undeveloped childs

All goals are linked to the top-level goal!
========================================================================
```

Figure 7.1: Autonomous vehicle scenario.

Generate Validation report:

1. select diagram file
2. press right click on files
3. select Reports
4. next select Validation report

The action is similar to validation action, the difference is that the report contains info only about this file.

The generation of documentation and todo reports is the same as this. The generated file are of pdf type.

Generate ToDo report:

1. select diagram file
2. press right click on files
3. select Reports

4. next select ToDo report

Generate Documentation:

1. select diagram file
2. press right click on files
3. select Reports
4. next select Documentation

The user can query a diagram by following the next steps:

1. if the command-console is not visible then add it in the view
   - select window->perspective->other
   - select console from the view
2. Write *start* and press enter to activate the console
3. select Ontology->Load Abox (to be sure that the diagram si loaded)
   or write $(set - current - abox\ \ diagramName)$
4. write the query and press enter

The console will look like this:

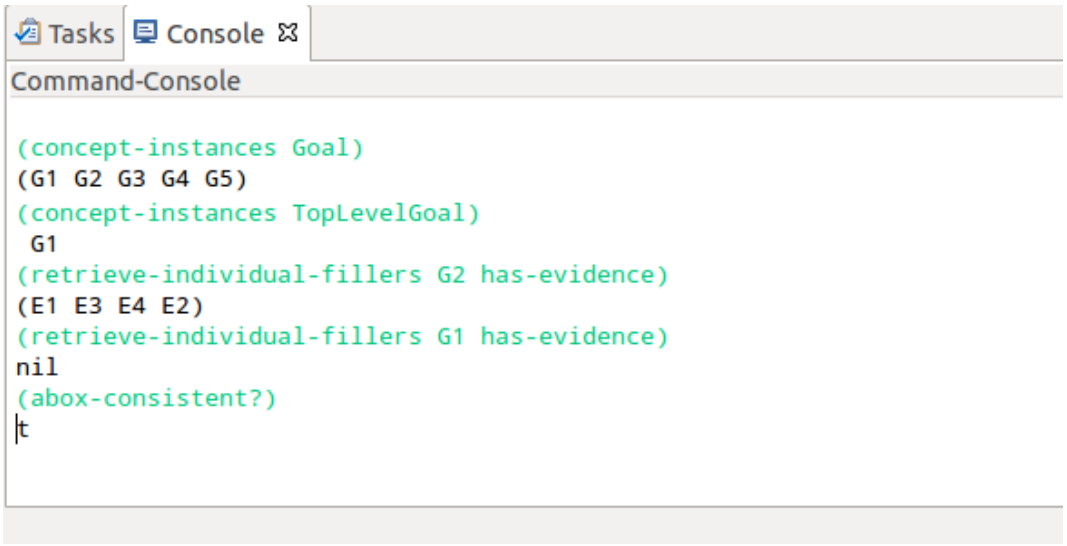

Figure 7.2: Command Console.

Chapter 8

Conclusions

8.1 Contributions and Achievements

From this paper we can deduce that it is very necessary to be sure that a system has a correct behavior and can be operately successfully so it is very important to build a safety case that describes the real level of safety that the system assures.

Assurance cases have evolved from the concept of safety cases being a requirement in many international standards. It is very important to build well-structured and coherent safety cases because wrong construction and reasoning in safety arguments can undermine a systems safety claims and lead to failures of the system. Like in case of Therac25, not building a simple assurance case (that would have lead to discover several faults) resulted in death of people.

All the elements of a safety case are part of the causal chain of hazard. The safety case is based on claims, any claim is broken into sub-claims until is reached a point when the claims can be proved by the development or assessment of an artefact as evidence. Every claim and strategy adopted to support the claim shoulf be very clearly formulated and state the context in which the argument is made so that the graphical representation of the safety case will be well structured.

It's a good practice to develop the safety cases prior to development because in this way you find the risks and faults before starting the development so you'll have the solution to mitigate them.

This paper described our tool that can not only be used to build safety arguments according to the Goal Structuring Notation standard but it also provides a better management and assesment of the safety case. The novelty of our approach is that the assurance case is automatically translated in description logic. The advantage is that the specific reasoning services of description logic are enacted to verify the compliance of the case with the GSN standard and also to signal possible argumentation flaws. The tool was demonstrated when developing safety all the safety cases presented in Chapter Testing and Validation. The main advantage of our the tool is that it can reduce the risks of not building well-structured safety cases or not provide evidence for all the statements from

the safety case.

Another big advantage is that our tool is extensible and can integrate with other corporate applications developed based on the Eclipse platform. In this line, ongoing work regards enhancing the tool with other.

8.2 Further Work

There are some improvements that can be done in the future to improve the tool and add new functionality. Being build as a set of eclipse plug-ins that need the eclipse platform to run on, the bigest improvement would be to get rid of this and run the tool as a standalone application by creating a Rich Client Platform(RCP) application for the tool. We choose this because it will improve the protability of the tool. RCP can run as standalone applications by including a minimal set of eclipse plugin Other future improvements:

- reverse the process of translation, from an A-box build accordingly to the T-box the user should be able to transform it into diagram. This will ease a lot more the work of the safety engineer.
- import and export diagrams from other editors
- improve the user interface: the metamodels should be colored distinctly from the start
- the generated reports should provide more information
- add the possibility to extend the metamodels

Bibliography

[1] G. Despotou, A. Apostolakis, and D. Kolovos, "Assuring dependable and critical systems: Implementing the standards for assurance cases with acedit," *Department of computer Science University of York, UK*, 2012.

[2] E. Denney, G. Pai, and J. Pohl, "Advocate: An assurance case automation toolset," in *Computer Safety, Reliability, and Security*. Springer, 2012, pp. 8–21.

[3] S. Hallerstede, M. Jastram, and L. Ladenberger, "A method and tool for tracing requirements into specifications," *Science of Computer Programming*, 2013.

[4] F. Baader, *The description logic handbook: theory, implementation, and applications*. Cambridge university press, 2003.

[5] V. Haarslev, K. Hidde, R. Möller, and M. Wessel, "The racerpro knowledge representation and reasoning system," *Semantic Web*, vol. 3, no. 3, pp. 267–277, 2012.

[6] J. K. Richard Hawkins, Tim Kelly and P. Graydon, "A new approach to creating clear safety arguments," 2011.

[7] T. P. Kelly, "Arguing safety – a systematic approach to managing safety cases," 1998.

[8] T. P. K. Patrick J.Graydon, "Using argumentation to evaluate software assurance standards," *Information and Software Technology*, vol. 55, no. 9, pp. 1551–1562, 2013.

[9] A. Wardziński, "Safety argument strategies for autonomous vehicles."

[10] R. Palin, D. Ward, I. Habli, and R. Rivett, "Iso 26262 safety cases: compliance and assurance," 2011.

[11] J. Birch, R. Rivett, I. Habli, B. Bradshaw, J. Botham, D. Higham, P. Jesty, H. Monkhouse, and R. Palin, "Safety cases and their role in iso 26262 functional safety assessment," in *SAFECOMP*, 2013, pp. 154–165.

[12] E. M. B. Melkild, "Goal and evidence based dependability assessment," p. 53, 2013.

[13] M. Conrad, P. Munier, and F. Rauch, "Qualifying software tools according to iso 26262." in *MBEES*, 2010, pp. 117–128.

[14] R. Dardar, B. Gallina, A. Johnsen, K. Lundqvist, and M. Nyberg, "Industrial experiences of building a safety case in compliance with iso 26262," in *Software Reliability Engineering Workshops (ISSREW), 2012 IEEE 23rd International Symposium on*. IEEE, 2012, pp. 349–354.

Appendix A

Published papers

1.Nicoleta Marc, Adrian Groza,"A formal model of Goal Structuring Notation", Computer Science Student Conference, Cluj-Napoca, 2014 Won first prize at the Conference.

2. Adrian Groza, Nicoleta Marc "Consistency Checking of Safety Arguments in the Goal Structuring Notation Standard", 2014 Paper submitted at ICCP2014