Consistency Checking of Safety Arguments in the Goal Structuring Notation Standard

Adrian Groza* and Nicoleta Marc*
*Intelligent Systems Group,
Department of Computer Science,
Technical University of Cluj-Napoca, Romania
Adrian.Groza@cs.utcluj.ro,nicoleta@isg.cs.utcluj.ro

Abstract—Justification of software conformance against specifications and standards is a strong requirement for safety-critical applications. Certification bodies require the construction of assurance cases. In an assurance case, the evidence supporting the claims is collected throughout the entire development cycle of a the safety application. The challenge is to build well-structured and coherent safety cases, given that the available technological instrumentation does not focus on automatic reasoning and verification of the safety case. In this paper, we propose a tool that facilitates the construction and automatic assessment of safety cases. The tool supports the Goal Structuring Notation (GSN) 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. A running scenario is illustrated in the domain of vehicular networks.

Index Terms—Goal Structuring Notation, assurance cases, description logic, vehicular networks.

I. INTRODUCTION

Software safety assurance is often demonstrated by compliance with national or international safety standards. Structured and evidence-based arguments are more and more used to describe assurance of systems [9]. In this line, our goal is to develop a system that supports automated construction and assessment of safety cases. The focus is on justifying the correctness of critical systems and conformance to international standards.

Autonomous systems raise problems for safety analysis and safety assurance [11], and therefore for certification [2]. For example, the ISO 26262 standard for vehicles [16], [15], [17] requires building a safety case for electrical/electronic products that presents a certain level of risks in order to prove that the system requirements are complete and satisfied by evidence.

This paper introduces a tool for writing safety arguments using the Goal Structuring Notation (GSN) standard. The main feature is that the GSN graphical notation is translated in 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 remaining of the paper is organised as follows: Section II introduces the technical instrumentation used: the Goal Structuring Notation and description logic. Section III describes the formalisation of the GSN standard in description logic. Section IV details the architecture of the system. Section V illustrates the formal checks performed in an autonomous vehicle system. Section VI discusses related work, while section VII points out the advantages of the system.

II. TECHNICAL INSTRUMENTATION

A. Description Logic

In the description logic $\mathcal{ALC}$, concepts are built using the set of constructors formed by negation, conjunction, disjunction, value restriction, and existential restriction [3], as shown in table I. 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 $\Delta^I$ 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 I shows the extension of $\cdot^I$ for non-atomic concepts.

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

Definition 1. A terminology TBox is a finite set of terminological axioms of the forms $C \equiv D$ or $C \subseteq D$.

Example 1 (Terminological box). In the box $\text{Vanet}$ in fig. 1, the first line specifies the domain of the role $\text{belongsTo}$ and its range is defined in the second line. Vehicles are partitioned into private and public (lines 3 and 4). The concept $\text{PublicVehicle}$ is further refined in $\text{Bus}$ and $\text{Police}$ (lines 5 and 6). The axiom 7 specifies that a $\text{PublicVehicle}$ should belong only to public agencies. The road side unit used for car-2-infrastructure communication can be deployed both by public or private service providers (line 8).
The Goal Structuring Notation (GSN) is an argumentation notation used to structure and represent graphically a safety argument.

According to GSN, the main elements of a 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 the level is reached where the sub-goal can be supported by evidence. The Strategy element is used to describe the method approached by the system to prove the claim. A Solution node provides the evidence or references to the evidence supporting the goal. The Context element provides information relevant for the corresponding goal, while the assumption element will provide statements that are already true. The Justification element 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.

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

### III. 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 A1 specifies the range for the role supportedBy:

\[(A_1) \quad T \subseteq \forall{\text{supportedBy}.(Goal \sqcap \text{Strategy} \sqcap \text{Solution})}\]

Axiom A2 specifies the domain of the role supportedBy, axiom A3 introduces the inverse role supports, and A4 constraints the role supportedBy to be transitive:

\[(A_2) \quad \exists{\text{supportedBy}.T} \subseteq \text{Goal} \sqcap \text{Strategy}
\]

\[(A_3) \quad \text{supportedBy}^{-} \equiv \text{supports}
\]

\[(A_4) \quad \text{supportedBy} \subseteq \text{supportedBy}
\]

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 A5 and A8 specify the range of the roles hasInference, respectively hasEvidence, while A6 and A9 the domain of the same roles. Definitions A7 and A10 say that the supportedBy is the parent role of both hasInference and hasEvidence, thus inheriting its constraints:

\[(A_5) \quad T \subseteq \forall{\text{hasInference}.\text{Goal}}
\]

\[(A_6) \quad T \subseteq \forall{\text{hasEvidence}.\text{Evidence}}
\]

\[(A_7) \quad \text{hasInference} \subseteq \text{supportedBy}
\]

\[(A_8) \quad \text{hasEvidence} \subseteq \text{supportedBy}
\]

\[(A_9) \quad \text{Goal} \subseteq \text{supportedBy}
\]

\[(A_{10}) \quad \text{supportedBy} \subseteq \text{supportedBy}
\]
Goals and sub-goals are propositions that we wish to be true that can be quantified as quantified or qualitative, provable or uncertainty.

\begin{align*}
(A_{11}) & \text{QuantitativeGoal} \equiv \text{Goal} \\
(A_{12}) & \text{ProvableGoal} \equiv \text{Goal} \\
(A_{13}) & \text{QualitativeGoal} \equiv \text{Goal} \\
(A_{14}) & \text{UncertaintyGoal} \equiv \text{Goal}
\end{align*}

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

\begin{align*}
(A_{15}) & \text{SupportGoal} \equiv \text{Goal} \land \exists \text{supports}.T \\
(A_{16}) & \text{TopLevelGoal} \equiv \text{Goal} \land \neg \text{SupportGoal}
\end{align*}

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

\begin{align*}
(A_{17}) & \exists \text{hasText}.\text{String} \\
(A_{18}) & \text{topLevelGoal} \equiv \text{Goal} \land \exists \text{supports}.T
\end{align*}

Three individuals gt, gp, and gu of type goal and their textual descriptions are instantiated by assertions f1 to f6:

\begin{align*}
(f_{1}) & \text{gt} : \text{TopLevelGoal} \\
(f_{2}) & \text{gt}, \text{“The system meets its requirements”} : \text{hasText} \\
(f_{3}) & \text{gp} : \text{ProvableGoal} \\
(f_{4}) & \text{gp}, \text{“Quick release are used”} : \text{hasText} \\
(f_{5}) & \text{gu} : \text{UncertaintyGoal} \\
(f_{6}) & \text{gu}, \text{“The item has a reliability of 95%”} : \text{hasText}
\end{align*}

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

\begin{align*}
(A_{20}) & \text{Explanation} \equiv \text{Statement} \sqcup \text{Reference} \sqcup \text{Justification} \sqcup \text{Assumption}
\end{align*}

where these top level concepts are disjoint:

\begin{align*}
(A_{21}) & \text{Statement} \equiv \neg \text{Reference} \\
(A_{22}) & \text{Statement} \equiv \neg \text{Justification} \\
(A_{23}) & \text{Statement} \equiv \neg \text{Assumption} \\
(A_{24}) & \text{Reference} \equiv \neg \text{Justification} \\
(A_{25}) & \text{Reference} \equiv \neg \text{Assumption} \\
(A_{26}) & \text{Justification} \equiv \neg \text{Assumption}
\end{align*}

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.

\begin{align*}
(A_{27}) & \text{Evidence} \equiv \exists \text{hasFormula}.\text{Formula} \sqcup \exists \text{hasSpecification}.\text{Statement} \sqcup \exists \text{hasModel}.\text{KripkeModel} \sqcup \exists \text{hasTestResult}.\top
\end{align*}

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

\begin{align*}
(A_{28}) & \text{NotVerifiedGoal} \equiv \text{Goal} \land \exists \text{hasEvidence}. \neg \text{VerifiedEvidence} \\
(A_{29}) & \text{NotVerifiedEvidence} \equiv \text{Evidence} \sqcup \exists \text{hasTestResult}. \neg \text{VerifiedEvidence}
\end{align*}

IV. SYSTEM ARCHITECTURE

Our tool consists of a set of Eclipse plugins. The tool is structured on layers (Fig. 4). At the bottom, there is the layer consisting of the core framework of the tool.

The second layer consists of several eclipse plug-ins used to implement the tool. The Eclipse Modelling Framework (EMF) was used for developing the model part. The Graphical Modelling Framework (GMF) and the Graphical Editing Framework (GEF) were used to implement the graphical user interface of the tool. Epsilon was used to construct the plug-ins for model management tasks.

The third layer contains the GSN and ARM metamodels, plus tool plug-ins through which all tool functionality is provided. This layer consists of: GSN plug-ins, which implements the GSN editor functionality, ARM plug-ins, which implements the ARM editor functionality, ONTOLOGY plug-ins, provides semantic reasoning facility.

The user interface layer consists of the GSN editor, the ARM editor and the model management tools: i) GSN to ARM transformation, ii) GSN validation using ontology-based reasoning, iii) safety case transformation in ABOX, iv) querying the safety case, v) various GSN editing wizards.

The main aim of the tool is to enhance end-user capabilities to build reliable assurance cases. The system supports management and assessment of the safety case, the Ontology plugin being responsible for this.

Based on the description logic, we developed an ontology that formalises the Goal Structuring Notation. The resulted GSN ontology is loaded from the Ontology plugin onto RacerPro engine using the jRacer library. The Ontology plugin provides also the engine used for translating the safety case diagram into an Abox. Furthermore, a connection to the RacerPro [12] reasoning engine is established in order to load the Abox in the GSN tbox so that the users could validate the abox against GSN tbox and query the safety case from the console.

An advantage is the possibility for the user to load many diagrams into the ontology and set the current safety case to be analyzed and query it from the console. Having the abox and tbox loaded in RacerPro, the user can select from the editor to
create the OWL ontology of the safety case used to generate a
document containing description of the safety case in natural
language or other reports.

The workspace of the system is presented in Fig. 5. A safety
project (top-left) consists of several assurance cases, developed
either as a graphical diagram (files with gsn extension) or
as an axon 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 axon, validate the diagram and generate
reports.

The main window (top-center) depicts the active gsn dia-
gram. 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 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, the four queries exemplified are: i) retrieving all
the goals in the diagram, ii) identifying the top level goal,
iii) listing all pieces of evidence supporting the goal g2, and
iv) 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.

V. FORMAL VERIFICATION OF SAFETY CASES

A. Vehicle Overtaking Scenario

A GSN diagram built in our SafeEd tool is represented
in Fig. 6. The considered safety scenario is taken from the
autonomous driving domain. The top level goal g1 states
that any autonomous vehicle should ensure safety when op-
erating in the environment. The goal holds in two contexts:
the existence of an environment formalisation (context c1),
respectively the existence of a mechanism providing situation
awareness. One solution for ensuring safety is dynamic risk
assessment approach [18]. The corresponding strategy s1 used
to support the goal g1 is to dynamically assess the risk.
The sub-goals g2, g3, g4, and g5 are used to fulfill the strategy
s1. For instance, the sub-goal g2 claims the correctness of
the model, statement that is supported by various pieces of
evidence, including formal verification e2.

The diagram in Fig. 6 is translated into the Abox represented
in Fig. 7. Here, the facts f31 to f34 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 f35 to f62. The
natural language text describing claims, solutions, contexts or
evidences are encapsulated as concrete attributes [12] in Racer
syntax (assertions f63 to f70).

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 [4]. Fig. 8 shows how such an analysis is performed
in order to comply to the ISO26262 requirements, according
to [6]. 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 8, 9, 10 only the hazard analysis and risk
assessment claim of the product is developed and shown the
corresponding process-based (Goal 2) and product-based(Goal
6) arguments.

The process-based goal Goal2 in refined in Fig. 9. 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: 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 10. 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 effects and causes of hazardous events have been analyzed. The goals have as solution techniques the same nodes Evidence2, Evidence3, Evidence4.

B. Validating the Safety Case

The RacerPro [12] reasoning engine is used by a tool to query and validate the Abox against the GSN box. 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.

In our running scenario, after analysing the diagram, the engineer observers that $g_3, g_4, g_5$ are undeveloped goals that needs evidences or have to be divided into sub-goals. If the engineer provides evidence for $g_3$ then the goal will no longer be part of undeveloped goals.

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.

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
Retrieving Information about the Safety Case

<table>
<thead>
<tr>
<th>Query</th>
<th>RacerPro query</th>
<th>RacerPro answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top level goal</td>
<td>(concept – instances TopLevelGoal)</td>
<td>g₁</td>
</tr>
<tr>
<td>Support goals</td>
<td>(concept – instances SupportGoal)</td>
<td>g₂, g₃, g₄, g₅</td>
</tr>
<tr>
<td>Evidence supporting goal g₂</td>
<td>(retrieve – individual – fillers g₂ hasEvidence)</td>
<td>e₁, e₂, e₃, e₄</td>
</tr>
<tr>
<td>Undeveloped Goals</td>
<td>(concept – instances UndevelopedGoals)</td>
<td>g₂, g₃, g₅</td>
</tr>
<tr>
<td>Generate OWL</td>
<td>(save – kb&quot;PATH/kb.owl&quot; : syntax : owl)</td>
<td></td>
</tr>
<tr>
<td>Check if Abox is consistent</td>
<td>(abox – consistent?)</td>
<td></td>
</tr>
<tr>
<td>Get all contexts of a specific goal</td>
<td>(individual – fillers g₁ inContextOf)</td>
<td>c₁, c₂</td>
</tr>
</tbody>
</table>

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 of individuals of type NotVerifiedGoal decreases.

Generating Natural Language Reports on the Safety Case

The tool supports the generation of documentation and reports for the safety case. As technical instrumentation, we use the RacerPro engine to further translate the ontology from description logic syntax into Web Ontology Language...
(OWL). The OWL file is fed to NaturalOWL [5] engine to transform the owl ontology into natural language and save the files as pdf. The generated files will contain texts describing individuals or classes of individuals from owl ontology.

Also reports with what still needs to be done or a report containing the assessment and validation of the safety case can be generated. An example can be found in figure 11 representing a validation report generated by our tool for the car overtaking safety case represented in Fig. 6. The 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.

The report provides also quantitative information of the diagram, in terms of number of nodes and their types. 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. Having the diagram and diagram documentation facilitate the work of the safety engineer or certification auditors.

VI. DISCUSSION AND RELATED WORK

There are several tools used for building safety cases, among them ACedit [8] and AdvoCATE [7]. Both tools support the creating of safety cases and they use the GSN notation for structuring the safety arguments.

ACedit [8] 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. Our 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.

AdvoCATE [7] is an Assurance Case Automation Tools, 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. 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. signal

The resulted formal case allows 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.
A tool for integrating informal and formal reasoning has been proposed in [14]. 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.

The GSN standard has been used to model autonomous vehicle scenarios [18] or unmanned aerial vehicles [13] overtaking [10] Various aspects of the ISO26026 standard have been modelled in GSN in [6].

VII. CONCLUSION

This paper described a tool that can be used to build safety arguments according to the Goal Structuring Notation standard. The novelty of our approach is that the assurance case is automatically translated into 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 during developing of a safety critical application for autonomous vehicles.

Our tool is extensible and can be integrated with other corporate applications developed based on the Eclipse platform. In this line, ongoing work regards enhancing the tool with other Eclipse plugins related to handling requirements in natural language (ProR plugin) or formal validation of properties (various model checking plugins in Eclipse) [14].

ACKNOWLEDGMENTS

We thank the reviewers for valuable comments. This research was supported by the Technical University of Cluj-Napoca, Romania, through the internal research project “GREEN-VANETS: Improving transportation using Car-2-X communication and multi-agent systems”.

REFERENCES