GSN_M-Edit: Ein modellgetriebener Editor für modulare GSN-Argumentationen

Patrick Werner[1] Stefan Gerken[2] Michaela Huhn[3]
[1] Institut für Theoretische Informatik, Technische Universität Braunschweig
[2] IC MOL RA R&D, Siemens AG
[3] Institut für Informatik, Technische Universität Clausthal

Abstract: In diesem Paper wird ein Konzept und eine prototypische Implementierung von GSN_M-Edit vorgestellt, einem Editor, der eine Modularisierung von GSN-Strukturen unterstützt. Modularisierung von Argumentationstrukturen in Sicherheitsnachweisen erleichtern die Wiederverwendbarkeit von Modulen in verschiedenen *Goal Structures*. Das vorgestellte Konzept wird anhand einer konkreten Implementierung - sowie anschließender Anwendung auf eine Argumentation evaluiert.

1 Einleitung

In sicherheitskritischen Systemen kommt dem Nachweis der funktionalen Sicherheit eine große Bedeutung zu. Ein wesentliches Kriterium für eine erfolgreiche Sicherheitsnachweisführung ist eine nachvollziehbare, überzeugende und stringente Argumentation, die belegt, dass die Sicherheitsziele von dem technischen System erreicht werden. Hierzu bietet die *Goal Structuring Notation* (GSN) [Kel98] eine anerkannte, graphische Strukturierungsmöglichkeit, die eine Argumentationskette übersichtlich visualisiert.

Größere technische Systeme und Anlagen setzen sich in der Regel aus vielen bereits erprobten Komponenten zusammen. Der Nachweis der Sicherheitsziele nimmt Bezug auf die Systemarchitektur. Daher bietet es sich an, auch die Argumentation modular aufzubauen, bereits exisitente Nachweise für Komponenten einzubeziehen und den Nachweis an der Systemstruktur des technischen Systems auszurichten.

Dies gilt umso mehr, wenn man sich vergegenwärtigt, dass etwa Ein-/Ausgabemodule technischer (Sub-)Systeme in der Regel für unterschiedliche Funktionen eingesetzt werden, und dabei auch in unterschiedlichen sicherheitstechnischen Funktionen mitwirken und deren individuelle Sicherheitsziele erfüllen müssen. Die Sicherheitsargumente für derartige Module werden daher in einer Sicherheitsargumentation an einer Vielzahl von Stellen referenziert.

Um Sicherheitsargumentationen zu modularisieren, werden Werkzeuge zum Editieren benötigt, die die Modularisierung von Zielen und das Einfügen von Referenzen an verschiedenen Stellen einer Argumentationsstruktur unterstützen. Ein solches Werkzeug ist die Grundlage für die effiziente Erstellung von Sicherheitsargumentationen, da bei einer architekturorientierten Strukturierung das verteilte Erstellen und die Wiederverwendung von

Nachweisen für Teilsysteme erheblich vereinfacht wird. Allerdings steigt mit einem modularen Ansatz auch die Notwendigkeit, Änderungen in ihren Auswirkungen nachverfolgen zu können.

Diese Überlegungen führten zu GSN$_M$-Edit, einem Editorprototypen für GSN-Strukturen, der unter Verwendung eines verbreiteten und offenen modellbasierten grafischen Frameworks, genau diese Art der konsistenten Modularisierung über Referenzen auf Teilargumente unterstützt.

Der Beitrag gliedert sich wie folgt: In Kapitel 2 werden die Konzepte der GSN eingeführt. Kapitel 3 betrachtet existente GSN-Editoren bezüglich der Modularisierung. Kapitel 4 stellt die Anforderungen und das Konzept für einen GSN-Editor vor, der Modularisierungsfunktionen unterstützt. In Kapitel 5 wird die Implementierung als Eclipse-Plugin beschrieben. Eine erste Evaluierung, an einem publizierten Sicherheitsnachweis für einen Herzschrittmacher, wird in Kapitel 6 skizziert. Kapitel 7 fasst den Beitrag zusammen und benennt die nächsten Schritte.

2 Die Goal Structuring Notation

Die Goal Structuring Notation (GSN) [Kel98, KW04, GSN11] unterstützt die graphische Darstellung von Argumentationsstrukturen durch die Darstellung der einzelnen Elemente eines Arguments und ihre Beziehungen untereinander. Die GSN wird insbesondere in Europa von einer größer werdenden Anzahl von Firmen verwendet, die sicherheitskritische Systeme herstellen oder betreiben, etwa in den Domänen Luftfahrt, Schienenverkehr, Verteidigung und Medizintechnik. Einige veröffentlichte Ergebnisse über industrielle Erfahrungen mit der Verwendung der GSN in Sicherheitsnachweisen finden sich in [CPK04]. Abbildung 1 zeigt anhand von Beispielinstanzen die Grundelemente der GSN.

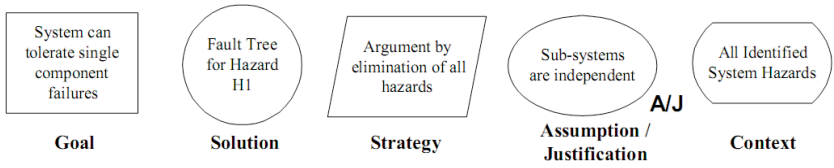

Abbildung 1: Grundsätzliche Elemente der GSN

In der GSN wird eine Argumentationsstruktur als ein gerichteter azyklischer Graph, *directed acyclic graph* (DAG), dargestellt, der auch *'Goal Structure'* genannt wird. Das vorrangige Ziel einer goal structure ist zu zeigen, wie *Goals* (Aussagen zur Systemsicherheit) angemessen und überzeugend argumentativ nachgewiesen werden können, indem sie auf eine Menge von evidenten Belegen (*Solution*) zurückgeführt werden. Dies funktioniert im allgemeinen über eine mehrstufige Argumentation mit weiteren *Sub-goals*. Dabei ist es möglich, die Argumentationsstrategie (*Strategy*) herauszustellen, die bei der Zerlegung in weitere Unterziele der Argumentation verwendet wurde. Die grundlegende GSN Notati-

on sieht darüber hinaus vor, Ziele in einem bestimmten Kontext (*Context*) zu definieren, und bisher nicht explorierte, also noch zu entwickelnde Aussagen zu kennzeichnen. GSN erlaubt grundsätzlich zwei Arten von Relationen zwischen Elementen: Die *SupportedBy* Beziehung (ausgefüllter Pfeil) beschreibt eine *wird-gestützt-von* Beziehung zwischen Strategien und Goals, sowie Solutions und Strategien bzw. Goals. Die *InContextOf* Beziehung (unausgefüllter Pfeil) beschreibt eine kontextuelle, einschränkende Beziehung, bspw. zwischen einem Goal und einem Context. Ein Beispiel für eine vollständige *goal structure* findet sich in [KW04].

Die graphische Darstellung der Argumentationsstruktur mit der GSN hat sich bei der Erstellung von Sicherheitsnachweisen als vorteilhaft erwiesen. Sie erleichtert allen an einem Projekt beteiligten Stakeholdern, die Sicherheitsargumente zu kommunizieren und zu verstehen. Eine Schwäche der GSN in ihrer Grundform ist die fehlende Modularisierung. Alle bisher vorgestellten Elemente zielen darauf ab, Sicherheitsnachweise mit Hilfe von GSN in einer monolithischen *Goal Structure* zu entwickeln.

Bei großen und komplexen Systemen können die Sicherheitsargumentationen aber stark anwachsen, was die Verständlichkeit deutlich mindert. Daher sind einige Erweiterungen der GSN vorgeschlagen worden, um Sicherheitsnachweise durch Modularisierung weiter zu strukturieren und die Wiederverwendbarkeit zu erhöhen [Kel01, GSN11]. Abbildung 2 zeigt die Elemente dieser Erweiterung.

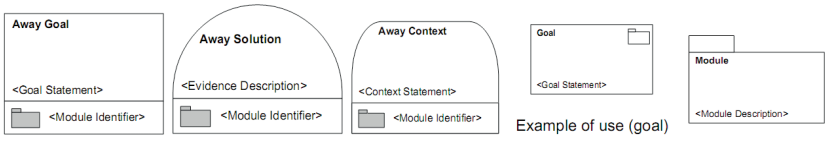

Abbildung 2: Erweiterte Elemente der GSN für Modularisierung

Module werden als zentrales neues Konzept eingeführt. Zusätzlich werden Elemente eingeführt, um Elemente aus (externen) Modulen zu referenzieren: Ein *Away-Goal* stellt dabei ein in einem anderen Modul fortgeführtes Goal dar. Analog werden *Away-Context* und *Away-Solution* Elemente eingeführt, die ebenfalls ihre jeweiligen Grundelemente in anderen Modulen referenzieren. Welche Elemente referenziert werden können, wird explizit durch einen *PublicIndicator* festgelegt, der in der oberen rechten Ecke jedes Goals, Context oder Solution der Basisnotation definiert werden kann. Abbildung 3 zeigt ein Beispiel einer solchen erweiterten *Goal Structure*. Zusätzlich beschreibt der GSN Standard [GSN11] generierte, nicht editierbare Overviewfunktionen. Abbildung 4 zeigt ein Beispiel eines solchen Overviews über alle in einer Goal Structure referenzierten Module.

3 Verwandte Ansätze

Im Zuge der stärkeren Verbreitung der GSN Notation und dem industriellen Einsatz sind mehrere Werkzeuge entstanden, um die Entwicklung von Argumentationsstrukturen in der

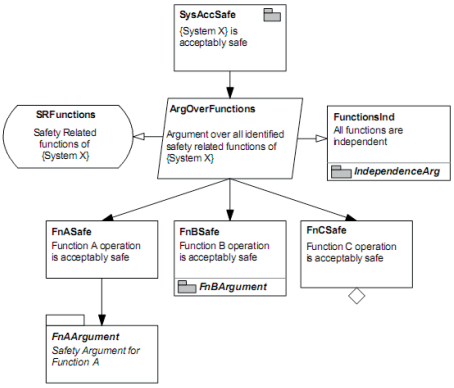

Abbildung 3: Inhalt eines Moduls

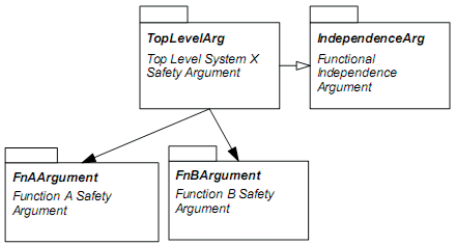

Abbildung 4: Externe Ansicht der Module und Beziehungen

GSN zu unterstützen: Der D-Case Editor [MT11], [D-C] und der ASCE Editor [ASC] stellen Funktionen bereit, um *Goal Structures* zu modellieren und auf GSN konforme Strukturen zu überprüfen. Wie bereits erwähnt, können vor allem größere Sicherheitsnachweise in diesen Tools unübersichtlich werden, da die Konzepte der Modularisierung wie sie in [GSN11] beschrieben sind, nicht umgesetzt sind.

Es ist zwar möglich, Strukturen in sogenannten Templates vorzudefinieren und anschließend in eigene Modelle einzufügen, die Anwendungen von konkreten Referenzierungen ist aber in beiden Implementierungen nicht möglich. Das führt dazu, dass vorhandene Sicherheitsnachweise, etwa von Komponenten, in komplexen Systemen nicht wiederverwendet werden können. Zwar ist es möglich, externe Goal Structures per 'copy-paste' einzufügen, dies führt aber schnell zu unübersichtlichen, nicht navigierbaren Strukturen. Darüber hinaus wird für jede Verwendung eine eigene Kopie benötigt, was erfahrungsgemäß nach wenigen Arbeitsschritten zu Abweichungen und Inkonsistenzen führt.

Der Atego GSN Modeler [Ate] liegt momentan als frühe Beta-Testversion vor. Er erlaubt die Modellierung von modularen *Goal Structures* und stellt darüber hinaus wiederverwendbare Argumentationspattern bereit, ermöglicht aber die Referenzierung von Elementen nur innerhalb eines Projektes. Dies macht eine konsistente, projektübergreifende Datenhaltung - insbesondere bei Mehrfachreferenzierungen - schwierig. Durch die gewählte Darstellungsform werden alle Module eines Projektes gemeinsam angezeigt, was ebenfalls zu unübersichtlichen Strukturen führt.

Unsere eigenen, bisherigen Arbeiten [HZ09, ZH09] zu einem GSN-Profil und Werkzeugunterstützung als Plug-In von Papyrus UML [Pap] zielen auf die enge Integration zwischen Sicherheitsnachweisführung und einem Architekturmodell in UML. Die Modularisierung von GSN-Strukturen wird nicht addressiert.

4 Unterstützung für GSN-Module

In diesem Abschnitt wird ein Konzept für einen GSN-Editor beschrieben, der die Modularisierung von Goal Structures unterstützt. Dafür wird die grundlegende Struktur von Argumentationen auf Projekte und Module erweitert. Ein Projekt stellt die Argumente eines Sicherheitsnachweises dar und besteht aus mehreren Modulen. Um die Modellierung mehrmoduliger Sicherheitsnachweise in GSN auch praktisch zu unterstützen, wird ein erweiterbarer Editor auf der Grundlage modellgetriebener Verfahren entwickelt [SV05].

4.1 Anforderungen

Ausgehend auf den bisher in anderen Werkzeugen nicht umgesetzten Funktionalitäten, dem GSN Standard [GSN11] und der Anwendung in einem praktischen Umfeld ergeben sich die folgenden Anforderungen an die Implementierung des GSN_M-Edit:

- Unterstützung der Erstellung, Bearbeitung und projektübergreifenden Wiederverwendung von GSN-Modulen, inklusive Überblicksfunktionen über die verwendeten Module (Overviews)
- Leichte Erweiterbarkeit des Editors, z.B. um domänen- oder projektspezifische Pattern oder Exportfunktionen
- Automatische Strukturanalyse für GSN-Strukturen mit Fehlerlokalisierung

4.2 Konzepte für die Modularisierung

Grundlage für die modellgetriebene Entwicklung eines GSN-Editors ist ein Metamodell, das die Elemente der Notation und ihre Beziehungen definiert. Zusätzlich zu den Grundelementen der GSN können auch Elemente verwendet werden, die in der modularen Erweiterung des GSN Standards [GSN11] definiert sind. Module dienen der Zusammenfassung kohärenter Teilargumentationen. Wir setzen voraus, dass ein Modul eindeutig einem Projekt zugeordnet wird. Die anderen Elemente für die Modulerweiterung erlauben, Elemente in anderen Modulen zu referenzieren. Dabei kann auch in andere Projekte referenziert werden. Auf diese Weise lassen sich Referenzen zwischen Modulen und auch zwischen Sicherheitsnachweisen erstellen. Diese Referenzen ermöglichen es nicht nur, Sicherheitsnachweise weiter zu strukturieren, sondern auch vorhandene Argumentationsstrukturen wiederzuverwenden. Abbildung 5 zeigt den ersten Teil des Metamodells, das die grundlegendenen Elemente der GSN Notation innerhalb eines Moduls beschreibt.

Alle Elemente innerhalb des Ausschnitts des Datenmodells in Abbildung 5 gehören zu einem Modul. Ein Nachweis besteht aus einer Menge von Modulen, die sich gegenseitig referenzieren können. Innerhalb des Datenmodells finden sich sowohl die grundsätzlichen Elemente der GSN, als auch die Elemente der modularen Erweiterung. Um es dem Be-

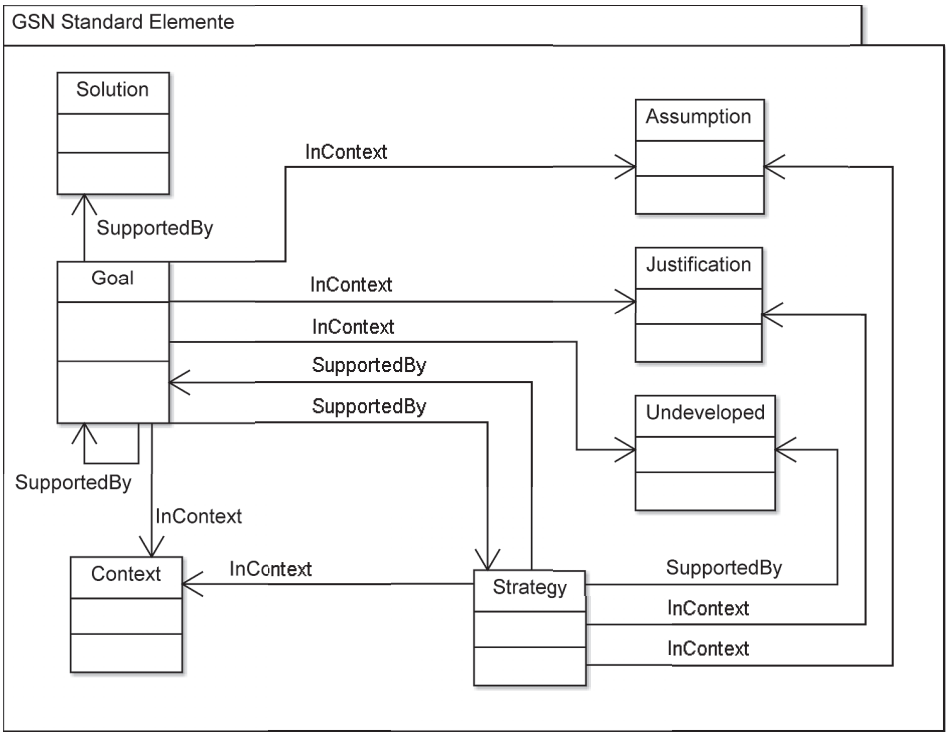

Abbildung 5: Datenmodell GSN - Standardelemente

nutzer einfacher zu machen, diese Elemente zu benennen und mit externen Quellen zu versehen, erben alle Datenelemente von einem Standardelement, mit Ausnahme des *Undeveloped* Elements, das lediglich eine noch nicht beendete Argumentationskette markiert. Dieses Standardelement ordnet jedem Knoten eine Identifikation, einen Inhalt und eine externe Pfadangabe innerhalb des aktuellen Projektes zu. In diesem Pfad kann jeder Knoten eine externe Quelle angeben, auf die er sich bezieht. Um die Übersichtlichkeit zu gewährleisten, kann maximal eine externe Quelle angegeben werden. Mit Hilfe der grundlegenden GSN-Elemente ist es möglich, Goal Structures eines Moduls zu beschreiben: So kann sich ein Goal in mehrere neue Goals aufteilen, bzw. über eine bis mehrere Strategien weiter dekomponiert werden oder durch Solutions belegt werden. Jede Strategie und jedes Goal kann von Assumptions oder Justifications eingeschränkt oder erläutert werden, bzw. in einem bestimmten Context stehen.

Die erweiterten Elemente *AwayGoal*, *AwayContext AwaySolution* und *ModuleReference* stellen in ihrem zugehörigen Modul eigenständige Elemente dar, referenzieren aber genau ein Element aus einem anderen Modul oder, im Fall einer *ModuleReference*, ein komplettes Modul. Dadurch wird eine Referenzierung auf verschiedenen Granularitätsebenen unterstützt. Wie bei den grundlegenden Elementen der GSN werden für referenzierte Elemente SupportedBy- und InContext-Beziehungen angeboten.

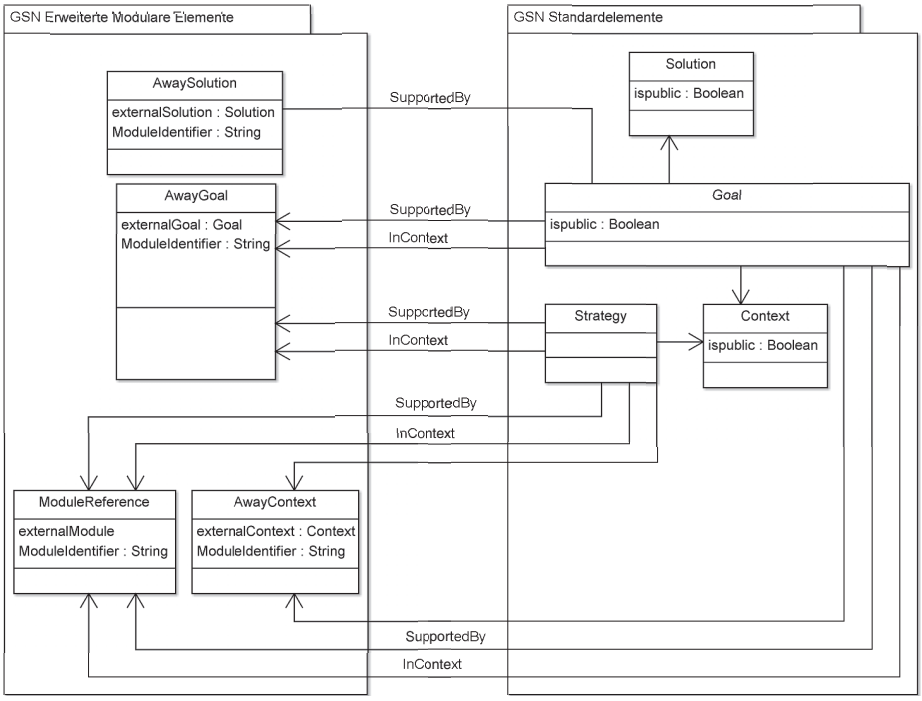

Abbildung 6: Datenmodell GSN - Erweiterung für Modularisierung

Alle Elemente, die in dieser Erweiterung referenziert werden können (Goal, Module, Context, Solution), erhalten in dem erweiterten Datenmodell (siehe Abbildung 6) ein boolesches Attribut, das von Anwender gesetzt werden kann. Dieses *ispublic* Attribut beschreibt, ob das jeweilige Element referenzierbar ist oder nicht.

Die erweiterten Elemente besitzen keine unabhängigen Attribute. Ihre Attribute sind von den von ihnen referenzierten Elementen abhängig. Zusätzlich enthalten sie ein weiteres Attribut. Dieses wird in Abhängigkeit der externen Referenz gesetzt und enthält den Namen des Moduls, in dem sich das referenzierte Objekt befindet. *AwayGoals* und *ModuleReferences* erlauben das Unterteilen des großen Modells in mehrere kleine Module, indem sie die Schnittstelle zwischen den Modulen darstellen. Die Elemente *AwayContext* und *AwaySolution* werden dann verwendet, wenn ein Solution- oder Context-Element in verschiedenen Modulen mehrfach verwendet wird. Diese Struktur hilft mehrfach vorkommende Elemente über Modulgrenzen hinaus einheitlich zu verwalten und Inkonsistenzen zu vermeiden. Die Verbindungen zwischen den Objekten innerhalb eines Moduls werden der Einfachheit halber als unidirektionale Verbindungen umgesetzt. Verbindungen über Modulgrenzen hinaus werden bidirektional modelliert, um die Beziehung in beiden beteiligten Modulen zu speichern. Über diese bidirektionale Modellierung werden im Falle einer Editierung des referenzierten Objektes alle referenzierenden Objekte angepasst.

4.3 Überprüfung von GSN-Strukturen

Erfahrungsgemäß findet die Erstellung eines komplexen Sicherheitsnachweises über mehrere Iterationen statt. Daher können Goal Structures während dieses Entstehungsprozesses von dem GSN Metamodell abweichen. Um dem Benutzer bei der Modellierung von Nachweisen möglichst große Freiheiten zu lassen, überprüft das hier vorgestellte Metamodell nur die Art der Referenzen, bzw. ob nötige Referenzen vorhanden sind oder nicht. Weitere Einschränkungen, die sich nicht kanonisch im Metamodell umsetzen lassen, werden in einer Menge von Regeln formuliert. Diese Regeln werden im Rahmen einer Strukturanalyse überprüft:

- Jedes Modul besitzt ein Top-Goal.
- Jedes Element mit Ausnahme von *solution*-Elementen besitzt *innerhalb seines Moduls* ein eindeutiges Vaterelement.
- Jedes Element ist mit dem aktuellen TopGoal - über eine Kette von Relations - verbunden.
- Jedes Element der modularen Erweiterung referenziert genau ein zugehöriges Grundelement in einem anderen Modul.
- Jedes Goal und jede Strategy sind entweder über SupportedBy-Beziehungen weiter mit Belegen versehen oder als *Undeveloped* markiert.
- Eine Goal Structure ist zyklenfrei bzgl. der SupportedBy- und InContext-Relationen.

Wird eine dieser Regeln innerhalb eines Moduls verletzt, wird eine entsprechende Fehlermeldung ausgegeben und somit Hinweise für nötige Korrekturen gegeben. Zusätzlich zu den hier definierten Regeln überprüft der Syntaxcheck aus dem Metamodell stammende Einschränkungen. Jedes Element der modularen Erweiterung muss genau eine Referenz auf ein zugehöriges Standardelement enthalten. Bei Anwendung der Strukturanalyse wird überprüft, ob diese Referenzen existieren. Alle hier definierten Regeln arbeiten jeweils auf einem bestehenden Modul nicht auf Beziehungen zwischen Modulen. Daher unterliegt es dem Anwender sicherzustellen, dass keine Zyklen über Modulgrenzen hinaus existieren.

4.4 Konzepte für die Wiederverwendbarkeit

Unser Ansatz ermöglicht die Wiederverwendbarkeit von Argumentationsstrukturen. Durch die Verwendung von Mehrfachreferenzierungen können vorhandene *Goal Structures* in neuen Sicherheitsnachweisen verwendet werden, ohne Redundanz zu erzeugen, aber auch referenzierte Module editiert werden, ohne Inkonsistenzen zu verursachen. Abbildung 7 zeigt einen Datenbestand aus zwei Systemen A und B, einem Subsystem C und zwei Komponenten D und E. Der Sicherheitsnachweis des Subsystems C wird hier in zwei Systemen A und B referenziert.

Um den Überblick über alle verwendeten Module zu ermöglichen, wird ein Overview verwendet wie er in Abbildung 4 skizziert ist. Um die Verständlichkeit von bestehenden Sicherheitsnachweisen zu erleichtern, werden zwei verschiedene Overviews angeboten: (1) Die Projektansicht beschreibt den Zusammenhang von Modulen innerhalb eines Projektes, d.h. alle Module innerhalb des Systems A aus Abbildung 7. Diese Ansicht ist hilfreich, um den internen Zusammenhang der Module innerhalb eines Projektes darzustellen. (2) Die Komplettansicht zeigt alle Module, die in der aktuellen Argumentationsstruktur über die Projektgrenzen referenziert werden, und hilft die Referenzen zwischen Projekten zu visualisieren. Die Komplettansicht von System A aus Abbildung 7 zeigt alle Module der Projekte von System A, Subsystem C, den Komponenten D und E und deren Zusammenhang an.

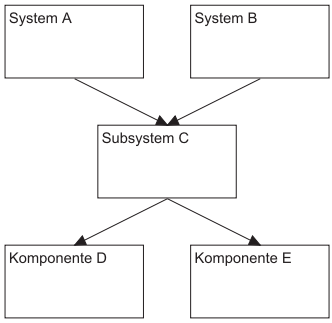

Abbildung 7: Beispiel - Wiederverwendbarkeit

5 Implementierung

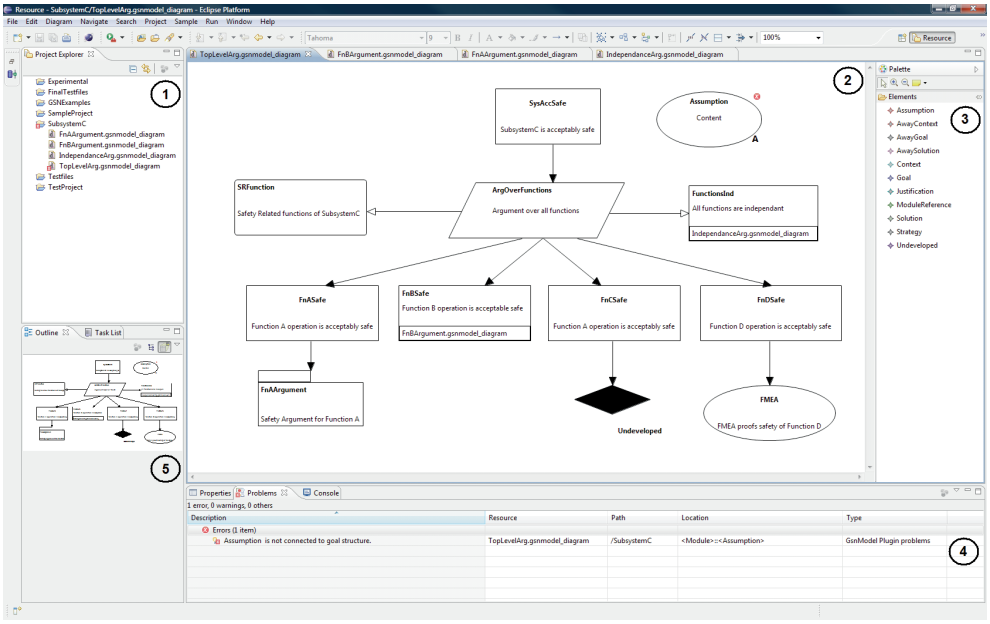

Abbildung 8: Screenshot der Implementierung - Subsystem C - Modul TopLevelArgument

Bei der Implementierung des vorgestellten Konzeptes wird ein modellgetriebener Ansatz

verwendet. Der Editor wird als Eclipse-Plugin implementiert. Dabei liegt die Priorität darauf, möglichst viele Funktionen unter Verwendung generativer Werkzeuge umzusetzen. Um effizient einen lauffähigen Prototypen zu generieren, verwenden wir EuGENia aus der Werzeugfamilie von Epsilon [KRP11]. EuGENia ist ein Front-End für das Graphical Modeling Framework (GMF) [Gro09] und zielt darauf ab, den Entwicklungsprozess von GMF basierten Editoren zu beschleunigen. Dabei wird ein lauffähiger Editor auf Basis eines vorhandenen Eclipse Modeling Framework (EMF) Metamodells erstellt [Gro09]. EuGENia erlaubt einige Annotationen von EMF Modellen, die den weiteren Generationsvorgang beeinflussen und damit weitere manuelle Anpassungen auf Generator- oder Codeebene minimieren. Das innerhalb des Konzepts vorgestellte Metamodell wird als grundlegendes EMF Modell verwendet. EuGENia generiert daraus zuerst die benötigen Dateien, die wiederum benötigt werden, um einen lauffähigen GMF Editor zu generieren. Diese im Zwischenschritt erstellten Dateien werden über Skripte angepasst, um den späteren Anpassungsaufwand auf Codeebene gering zu halten. Die Verwendung von EMF basierten Tools sorgt für Erweiterbarkeit, da alle mit Hilfe des Editors erstellten Instanzen im verbreiteten XML Metadata Interchange (XMI) Format [GDB02] gespeichert werden. Die Overviews werden - aufbauend auf diesen erstellten XMI-Dateien - als separates Eclipse-Plugin implementiert. Zusätzlich zur Codegenerierung bietet Epsilon weitere Funktionen, darunter die Epsilon Validation Language (EVL), eine modellbasierte Validierungssprache zur Konsistenzüberprüfung von Modellen mit Hilfe von definierten Regeln. EVL wird zur Implementierung der automatischen Strukturanalyse verwendet. Abbildung 8 zeigt einen Screenshot des Prototyps. Auf der linken Seite der Abbildung (1) sieht man den Projektexplorer, der die erstellten GSN Projekte im aktuellen Workspace zeigt. Im Zentrum (2) steht das Editorfenster zur Modellierung von *Goal Structures*. Dabei können alle hier beschriebenen GSN-Elemente aus der Palette (3) verwendet werden. Im unteren Teil sieht man zwei weitere Fenster, eines davon zeigt einen Überblick über das aktuelle Modul (5), das andere zeigt alle bei einer Strukturanalyse gefundenen Fehler an (4).

6 Evaluation

Bei der Evaluation wird untersucht, ob sich die GSN Notation [GSN11] im erstellten Editor abbilden und anwenden lässt. Dafür verwenden wir einen komplexeren bereits existierenden Sicherheitsnachweis eines Herzschrittmachers [JLS10]. Anders als in bisher bestehenden Werkzeugen, kann dieser Sicherheitsnachweis in unserer Implementierung entsprechend der in [JLS10] vorgeschlagenen Struktur modularisiert werden. Die Anwendung verlief erfolgreich und erlaubt die Modellierung einer übersichtlichen strukturierten Abbildung dieses Nachweises in dem erstellten Editor. Zur Evaluation wurden einige strukturelle Fehler in die Argumentation eingefügt, die von der anschließenden Strukturanalyse korrekt erkannt wurden. Nach einer Korrektur der gemeldeten Fehler verlief eine erneute automatische Analyse erfolgreich. Auch die Wiederverwendbarkeit wurde adressiert, indem ein Teil des Beispiels in weiteren Projekten referenziert wurde. Dabei zeigte sich der Vorteil der unterschiedlichen Overviews bezüglich des Verständnisses und Überblicks über die erstellen Projekte, aber auch der Nachteil fehlender Versionierungs-

funktionen. Dies ist insbesondere relevant, wenn Argumentationselemente in verschiedenen Versionen eines Sicherheitsnachweises benötigt werden. Daraus resultiert die Anforderung einer anwendungsbezogenen Versionsverwaltung im nächsten Schritt. Im Review mit Domänenexperten erwiesen sich die Modularisierungsmöglichkeiten und die Wiederverwendbarkeitsfunktionen als vielversprechend. Insbesondere die Lesbarkeit und Navigierbarkeit größerer Strukturen, unterstützt durch die Verwendung der Elemente der modularen Erweiterung, unterlag positiven Rückmeldungen. Eine umfassende Evaluation anhand eines großen, praktisch relevanten Projektes ist geplant.

7 Zusammenfassung und Ausblick

Das Eclipse Modeling Framework mit seinen Ergänzungen GMF und Epsilon bietet eine gute Basis, um die GSN modellbasiert in einem grafischen Editor umzusetzen und damit Argumentationen zu modularisieren und zu strukturieren. Das realisierte Plugin GSN$_M$-Edit unterstützt die Wiederverwendung von Argumenten an verschiedenen Stellen eines Sicherheitsnachweises und die zentrale Pflege eines Arguments. Aufgrund der offenen Architektur bietet Eclipse einfache Schnittstellen zu Versionsverwaltungssystemen und anderen Werkzeugen zur Softwareentwicklung und Verifikation. Daher ist die Erweiterung von GSN$_M$-Editors mit weiteren Ansätzen zur Modularisierung nach gegenwärtiger Einschätzung vielversprechend.

Als nächste Schritte sind geplant:

(1) Versionierung von Argumentstrukturen mit einer Änderungsauswirkungsanalyse, mit der sich die von Änderungen betroffenen Argumentfolgen identifizieren lassen. Ggf. soll es möglich sein, verschiedene Versionen eines Arguments gleichzeitig zu verwenden, falls in einem technischen System unterschiedliche Versionen einer Komponente eingesetzt werden.

(2) Verknüpfung der Argumentstruktur mit der Systemarchitektur und Realisierung von Plausibilitätsüberprüfung.

(3) Unterstützung von Argumentationsmustern für dedizierte Eigenschaften, um eine Best-Practice zu bewahren und Konsistenz und Effizienz der Nachweiserstellung zu verbessern.

Literatur

[ASC] ASCE Safety Case Tool. `http://www.adelard.com/web/hnav/ASCE/`.

[Ate] Atego GSN Modeler. `http://http://www.atego.com/download-center/product/atego-gsn-modeler/`.

[CPK04] Paul Chinneck, David Pumfrey und Tim Kelly. Turning Up The HEAT On Safety Case Construction. In *In Practical Elements of Safety: Proceedings of the Twelfth Safety-critical Systems Symposium*, Seiten 223–240. Springer, 2004.

[D-C] D-Case Editor A Typed Assurance Case Editor. http://www.il.is.s.u-tokyo.ac.jp/deos/dcase/.

[GDB02] Timothy J. Grose, Gary C. Doney und Stephen A. Brodsky. *Mastering XMI: Java Programming with XMI, XML and UML.* Wiley, 1. Auflage, April 2002.

[Gro09] Richard C. Gronback. *Eclipse Modeling Project*. Addison-Wesley Longman, Amsterdam, 2009.

[GSN11] GSN Community Standard Version. http://www.goalstructuringnotation.info/documents/GSN_Standard.pdf, November 2011.

[HZ09] Michaela Huhn und Axel Zechner. Analysing Dependability Case Arguments Using Quality Models. In *28th International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, Jgg. 5775 of *LNCS*, Seiten 118–131, 2009.

[JLS10] Eunkyoung Jee, Insup Lee und Oleg Sokolsky. Assurance Cases in Model-Driven Development of the Pacemaker Software. In *Proc. of the 4th Intern. Conf. on Leveraging Applications of Formal Methods, Verification, and Validation (ISoLA, Vol. II)*, Seiten 343–356, 2010.

[Kel98] Tim Kelly. *Arguing Safety – A Systemic Approach to Managing Safety Cases*. Dissertation, University of York, September 1998.

[Kel01] Tim Kelly. Concepts and Principles of Compositional Safety Cases. Bericht COMSA/2001/1/1, May 2001.

[KRP11] Dimitrios Kolovos, Louis Rose und Richard Paige. The Epsilon Book. http://www.eclipse.org/gmt/epsilon/doc/book/, 2011.

[KW04] Tim Kelly und Rob Weaver. The Goal Structuring Notation - A Safety Argument Notation. In *Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.

[MT11] Yutaka Matsuno und Kenji Taguchi. Parameterised Argument Structure for GSN Patterns. In *11th Intern. Conf. on Quality Software (QSIC)*, Seiten 96–101, 2011.

[Pap] Papyrus for UML. http://www.eclipse.org/modeling/mdt/papyrus/.

[SV05] Thomas Stahl und Markus Völter. *Modellgetriebene Softwareentwicklung. Techniken, Engineering, Management*. Dpunkt Verlag, 1. Auflage, 2005.

[ZH09] Axel Zechner und Michaela Huhn. Structural Analysis of Safety Case Arguments in a Model-based Development Environment. In *Dagstuhl-Workshops Modellbasierte Entwicklung eingebetteter Systeme (MBEES V)*, 2009.