

Design for Trust: Neue Dimensionen der Sicherheit

W. Reif, A. Thums, G. Schellhorn

Universität Augsburg

Lehrstuhl für Softwaretechnik und Programmiersprachen

Universitätsstr. 14, D-86135 Augsburg

Tel.: +49 (0)821 598 2174

Fax: +49 (0)821 598 2175

E-Mail: {reif,thums,schellhorn}@informatik.uni-augsburg.de

Zusammenfassung

Trotz der enormen Fortschritte der Softwaretechnik in den letzten 30 Jahren ist die Entwicklung sicherer und zuverlässiger eingebetteter Systeme noch immer eine der großen Herausforderungen des Gebiets.

Besondere Methoden sind vor allem für sicherheitskritische Anwendungen mit höchsten Qualitätsanforderungen interessant. Beispiele hierfür sind Medizintechnik, Luft- und Raumfahrt, Eisenbahn, Automobilelektronik, Datensicherheit sowie die Kontrolle technischer Anlagen. Weitere Einsatzgebiete sind Software in Massenprodukten sowie zentrale Softwaredienste, über die sehr viele Anwendungen abgewickelt werden. Aber auch im Bereich E-Commerce wird die Sicherheit zu einem der erfolgsbestimmenden Faktoren.

Der Vortrag gibt einen Überblick, was moderne Methoden heute in puncto Sicherheit und Zuverlässigkeit leisten können und wo deren Grenzen sind. Neben klassischen Methoden der Sicherheitsanalyse (FTA) wird auch das Potenzial formaler Methoden beleuchtet, wie sie z.B. bei Zertifizierungen nach den hohen Stufen von ITSEC und CC gefordert sind.

1 Einleitung

Die zunehmende Verwendung von Software in sicherheitskritischen Anwendungen verlangt nach Methoden, um die Zuverlässigkeit und Sicherheit des gesamten, eingebetteten Systems nachzuweisen.

Sicherheitsanalysemethoden wie FMEA [Rei79], FTA [VGRH81] oder HAZOP [Kno89] werden in den Ingenieurwissenschaften schon seit Jahrzehnten für technische Systeme eingesetzt. Durch die Tendenz, Hardware durch Software zu ersetzen, ist es notwendig, dass softwarebasierte Systeme die gleichen Sicherheitskriterien wie rein technische Systeme erfüllen. Deshalb wurde schon in [Lev95, Sto96] die Anwendung von Sicherheitsanalysemethoden auf Software vorgeschlagen. Ein Problem der dieser Methoden ist jedoch, dass

sie auf informelle Systembeschreibungen angewendet werden. Dadurch kann es bei komplexen Systemen leicht geschehen, dass Fehlerursachen übersehen werden und man ungerechtfertigtes Vertrauen in das System bekommt. An dieser Stelle können formale Methoden helfen.

Formale Methoden basieren auf Sprachen für die Systembeschreibung und zur Formulierung von (Sicherheits-) Eigenschaften mit einer präzisen, mathematischen Semantik. Die Systeme und die Sicherheits- und Konsistenz-eigenschaften, die das System erfüllen soll, werden in einer formalen Sprache beschrieben. Die Eigenschaften werden dann mit mathematischen Beweisen über dem formalen Modell nachgewiesen.

Ursprünglich wurde Sicherheitsanalyse für Hardware entwickelt und die formalen Methoden hauptsächlich für Softwareverifikation eingesetzt. Für softwarebasierte System, wie der Medizintechnik, Luft- und Raumfahrt, Eisenbahn und Automobilelektronik ist jedoch eine durchgängige Methodik unabdingbar, die durch die Integration von Sicherheitsanalyse mit formalen Methoden geschaffen wird. Sicherheitseigenschaften werden mit Sicherheitsanalyse abgeleitet und über dem formalen Modell nachgewiesen. Eine formale Semantik der Sicherheitsanalysetechnik erlaubt die Validation der Analyse. Wir werden im Folgenden diesen Ansatz am Beispiel einer dezentralisierten Bahnübergangsteuerung [JS99] erläutern.

In dieser Arbeit präsentieren wir kurz unseren Ansatz zur formalen Sicherheitsanalyse, genauer der FTA, konzentrieren uns aber auf deren Anwendungsmethodik. Um von dem kreativen Ansatz der Fehlerbaumanalyse zu profitieren, starten wir auf Basis eines informellen Modells. Darauf aufbauend wird in getrennten Schritten ein formales Modell und die Fehlerbäume aller relevanten Gefahren erstellt. Dann werden die Sicherheitseigenschaften, die aus FTA abgeleitet werden, in das formale Modell übernommen und nachgewiesen. Dies kann auf zwei verschiedenen Stufen geschehen. Entweder werden die Sicherheitseigenschaften formalisiert und nachgewiesen, oder wir gehen einen formaleren Weg und formalisieren, zusätzlich zu den Sicherheitseigenschaften, die Ereignisse des Fehlerbaums über dem formalen Modell. Aus dem Fehlerbaum werden dann Beweisverpflichtungen über diesen Ereignissen generiert, deren Nachweis die Vollständigkeit und Korrektheit des Fehlerbaums garantieren. Für diese Validierung der FTA benutzen wir die Fehlerbaumsemantik aus [STR02]. Diese beiden Stufen der Integration nennen wir lose bzw. eng gekoppelte formale FTA.

Zur Erläuterung unseres Ansatzes wird in Abschnitt 2 das Beispiel des funkbasierten Bahnübergangs beschreiben. Basis der formalen FTA sind formale Methoden, für die in Abschnitt 4, und die FTA, die in Abschnitt 3 beschrieben werden. Die Formalisierung der FTA geschieht in Abschnitt 5. Der Fokus dieser Arbeit liegt auf der Anwendungsmethodik der formalen

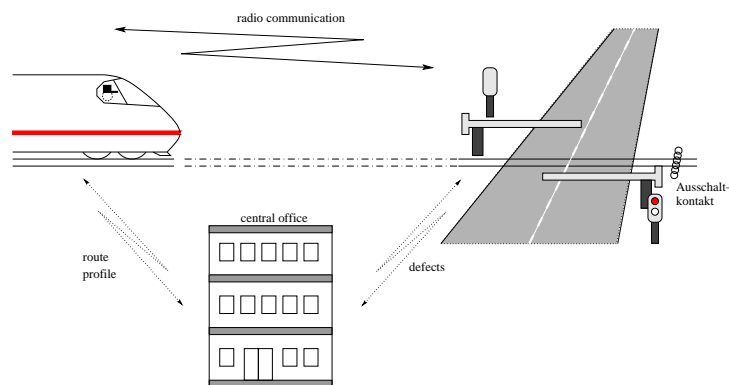


Abbildung 1: Fallbeispiel: funkbasierte Bahnübergangsteuerung

FTA, die in Abschnitt 6 erläutert wird. Abschließende Bemerkungen in Abschnitt 7 runden die Arbeit ab.

2 Die funkbasierte Bahnübergangsteuerung

Die funkbasierte Bahnübergangsteuerung [JS99] ist an eine neuartige Entwicklung der Deutschen Bahn, dem funkbasierten Fahrbetrieb (FFB) [Bet], angelehnt. Der FFB kann für Bahnstrecken mit einer Höchstgeschwindigkeit von 160 km/Std eingesetzt werden. Der Unterschied dieser neuen Technologie gegenüber der herkömmlichen Sicherung besteht darin, dass Sensoren und Signale an der Stecke durch Berechnungen in der Zug- und Bahnübergangsteuerung und durch Kommunikation über Funk ersetzt werden. Dies bietet einen flexibleren und kostengünstigeren Einsatz. So kann der Schließzeitpunkt des Bahnübergangs vom Zug, in Abhängigkeit von seiner momentanen Geschwindigkeit, berechnet und die Wartezeiten am Bahnübergang minimiert werden.

Der Schließvorgang eines Bahnübergangs wird von dem sich nähernden Zug angestoßen (see Fig. 1). Er kennt die Position des Bahnübergangs, den sogenannten Gefahrenpunkt, und berechnet aus der Differenz zu seiner eigenen Position und seiner Geschwindigkeit den Einschaltzeitpunkt des Bahnübergangs. Erhält der Bahnübergang ein Schließsignal, schaltet er die Lichtzeichenanlage an, d. h. zuerst das Gelblicht und etwas später das rote Licht. Dann werden die Schranken geschlossen. Wenn diese vollständig geschlossen sind, ist der Bahnübergang nur für eine bestimmte Zeitspanne – damit die Wartezeiten für andere Verkehrsteilnehmer nicht zu lang werden – gesichert. Kann der Bahnübergang aufgrund von Defekten nicht gesichert werden, verbleibt er ungesichert. Durch eine Statusabfrage erhält die Zugsteuerung Auskunft

über den Zustand des Bahnübergangs. Ist er gesichert, bekommt der Zug eine Freigabe und kann den Bahnübergang passieren. Ansonsten muss die Zugsteuerung dafür sorgen, dass der Zug rechtzeitig vor dem Bahnübergang zum Stehen kommt. Der Bahnübergang kann dann nur noch durch ein manuelles Sicherungsverfahren passiert werden.

In den nachfolgenden Abschnitten verwenden wir dieses Beispiel zur Veranschaulichung unserer Techniken. Für die formale FTA wurde dieses Beispiel in STATEMATE modelliert und wir verweisen auf [KT02] für eine ausführliche Beschreibung der Modellierung und auf [RST00b] für eine Sicherheitsanalyse dieses Systems.

3 Fehlerbaumanalyse (FTA)

Die Fehlerbaumanalyse (FTA) ist eine bekannte Technik aus den Ingenieurwissenschaften und wurde für technische Systeme entwickelt, um von ihr ausgehende Gefahren zu analysieren (Top Ereignis). Diese *Ereignis* bildet die Wurzel des Fehlerbaums. Ereignisse, die diese Gefährdung verursachen (Zwischenereignisse) werden in Unterknoten notiert und (rekursiv) analysiert. Jedes untersuchte Ereignis ist mit seinen Ursachen durch sogenannte *Gatter* verknüpft (siehe Abbildung 2). Dadurch ergibt sich ein Baum, in dem sich Ereignisse und Gatterknoten abwechseln. Ein *UND-Gatter* kenn-

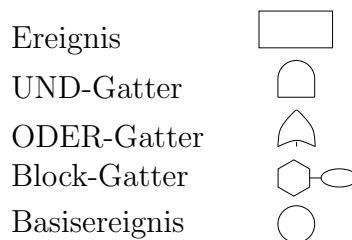


Abbildung 2: Fehlerbaumsymbole

zeichnet, dass alle Ursachen notwendig sind, um ein Ereignis zu verursachen, bei einem *ODER-Gatter* genügt eine einzelne Ursache. Ein *BLOCK-Gatter* verlangt zusätzlich zu den Ursachen eine Bedingung (im Oval annotiert) die erfüllt sein muss, um das Ereignis hervorzurufen. Damit ist es mehr oder weniger ein *UND-Gatter* mit dem Unterschied, dass die Nebenbedingung kein Fehler oder Defekt sein muss.

Blätter im Fehlerbaum kennzeichnen *Basisereignisse* für die Gefährdung, die entsprechen der Gatter im Fehlerbaum auftreten müssen, um die Gefährdung zu verursachen. Eine Kombination von Basisereignissen, die die Gefährdung verursachen, wird *Cut Set* genannt. Ein *Cut Set* ist *minimale*, wenn das

verhindern eines einzigen Basisereignisses genügt, um die Gefährdung (durch dieses Cut Set) auszuschließen. Die minimalen Cut Sets können aus einem Fehlerbaum berechnet werden, indem die Gatter durch die entsprechenden booleschen Operatoren ersetzt werden. Dazu ersetzt man Bottom up die Ereignisse im Fehlerbaum durch die Konjunktion bzw. Disjunktion der Untereignisse. Die minimalen Cut Sets ergeben sich dann aus der Disjunktiven Normalform der Formel für den Wurzelknoten.

Die Cut Sets helfen kritische Ereignisse zu identifizieren. Wenn ein Ereignis in mehreren Cut Sets vorkommt, hängt die Sicherheit des Gesamtsystems stark vom Eintreten dieses Ereignisses ab. Die Sicherheit steigt stark, wenn dieses Ereignis verhindert werden kann. Neben einer solchen *qualitativen Analyse* kann auch eine *quantitative Analyse* durchgeführt werden. Dazu müssen die Eintrittswahrscheinlichkeiten der Basisereignisse bekannt (Basisereignisse sind häufig Defekte von Standardkomponenten, für die statistische Daten vorliegen) und statistisch unabhängig sein. Die Eintrittswahrscheinlichkeit eines minimalen Cut Sets ist das Produkt der Eintrittswahrscheinlichkeiten für die Basisereignisse. Die Gesamtwahrscheinlichkeit die Summe der minimalen Cut Set Wahrscheinlichkeiten. Für differenzierte Verfahren verweisen wir auf [VGRH81].

Als Beispiel eines (partiellen) Fehlerbaums betrachten wir die Gefährdung *Kollision* bei der funkbasierten Bahnübergangsteuerung (siehe Abbildung 3). Die Kollision ist als *Zug auf Bahnübergang, Schranken nicht geschlossen* de-

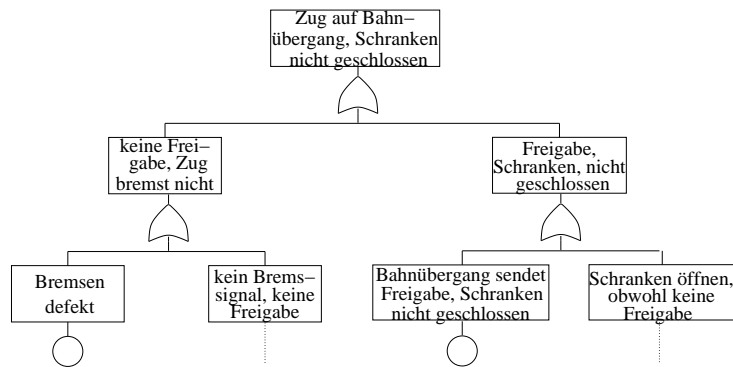


Abbildung 3: FTA für *Kollision*

finiert. Diese Ereignis tritt ein, wenn entweder der Zug nicht brems, obwohl er keine Freigabe erhalten hat, oder die Schrankensteuerung eine Freigabe sendet, obwohl die Schranken nicht geschlossen sind. Das erste (linke) Untereignis hat wiederum zwei Ursachen. Entweder sind die Bremsen defekt oder sie haben kein Brems-signal von der Zugsteuerung erhalten. Auf diese

Art und Weise werden alle Ereignisse auf Ausfälle/Defekte der Basiskomponenten des Modells heruntergebrochen (der gesamte Fehlerbaum findet sich in [RST00b], weitere Sicherheitsbetrachtungen in [RST00a]).

4 Formale Methoden

Formale Methoden bieten die Möglichkeit, die Mehrdeutigkeit und fehlerhafte Interpretation natürlicher Sprache oder informeller Spezifikationsmethoden zu überwinden. Dazu werden mathematische Techniken mit einer präzisen Semantik eingesetzt. Der Vorteil der formalen Methoden besteht insbesondere darin, dass sie schon in den frühen Phasen des Softwareentwicklungsprozesses (Analyse und Design) eingesetzt werden können. Dadurch werden die Fehler und Schwächen der Modelle frühzeitig und nicht erst in der Implementierungs- oder Integrationsphase erkannt und können kostengünstiger beseitigt werden.

Formale Spezifikations Sprachen ermöglichen durch (automatischen) Analysen und Validation eindeutige, vollständige, konsistente und korrekte Spezifikationen zu erstellen. Die Grundlage dafür ist die präzise Semantik formaler Sprachen. Bei der Validation werden erwarteten Eigenschaften des Systems formuliert und mittels mathematischer Beweiser deren Gültigkeit über dem Modell nachgewiesen. Die Verifikation weist dann Sicherheitseigenschaften des Systems nach und stellt damit sicher, dass das System korrekt und sicher erstellt wird. Verfeinerungen der Spezifikation erlauben es dann, Programmcode aus der Spezifikation abzuleiten und als korrekt (gegenüber dem Modell) nachzuweisen. Damit schaffen die formalen Methoden eine Beweisunterstützung von der Analyse über den Design bis hin zur Implementierung.

Neben der Anwendungsprogrammierung ist zunehmend auch der Entwurf und die Analyse moderner Programmiersprachen ein lohnendes Einsatzgebiet formaler Methoden. Das Spektrum reicht von der klärenden, formalen Fundierung der Semantik [BR95], [Coh97], [BS99], [vON99], über den Nachweis von Spracheigenschaften wie z.B. der Typsicherheit, [NN97], [NvO98], bis hin zur Verifikation von Compilern für unterschiedliche Zielmaschinen [SA98], [Moo88], [GDG⁺96], [BHL⁺96]. Dieser Zweig der formalen Methoden hat nicht zuletzt durch das Aufkommen von JAVA und der damit verbundenen Sicherheitproblematik einen starken Aufschwung erfahren.

Formale Methoden haben mittlerweile auch Eingang in die europäischen bzw. weltweit harmonisierten Kriterienkataloge ITSEC [ITS91], bzw. Common Criteria [CCI98] zur Evaluierung und Zertifizierung informationstechnischer Produkte gefunden. Für eine Prüfung nach den hohen Vertrauensklassen in ITSEC und Common Criteria, die z.B. für online-banking oder digitale Unterschriften in Frage kommen, wird u.a. ein formales Sicherheitsmodell

gefordert. Solche Prüfungen werden von Herstellern initiiert und von unabhängigen, behördlichen Stellen (z.B. dem Bundesamt für Sicherheit in der Informationstechnik, BSI) oder zugelassenen, privaten Einrichtungen durchgeführt. Insbesondere die jüngsten Gesetzesregelungen zur digitalen Signatur haben in Deutschland eine Reihe von Evaluierungen und Zertifizierungen angestoßen.

5 Formale FTA

Die Kombination von FTA und formalen Methoden erlaubt die Korrektheit und Vollständigkeit von Fehlerbäumen über einem Modell nachzuweisen und die Güte formaler Modelle qualitativ und quantitativ zu bewerten. Dazu müssen die Ereignisse des Fehlerbaums formalisiert, d. h. in Formeln der formalen Modells übersetzt, und Bedingungen, die Gatter zwischen Ereignis und Unterereignisse beschreiben, nachgewiesen werden. Diese Beweise garantieren dann die Korrektheit und Vollständigkeit der FTA. Der validierte Fehlerbaum wird dann zur Bewertung des formalen Modells genutzt.

In diesem Abschnitt geben wir die formale Bedeutung der Gatter an. Jedes Gatter wird durch eine Formel beschrieben die die Gatterbedingung beschreibt. Da wir dynamische Systeme betrachten, bei denen die Konsequenzen eines Ereignisses nicht immer sofort sichtbar werden, führen wir neue, sogenannte *Ursache/Wirkung-Gatter* ein. Die Formalisierung der FTA findet dann in Intervalltemporallogik (ITL) statt. Eine kurze Einführung in die logischen Grundlagen der ITL beschreiben wir in Abschnitt 5.1 und die Formalisierung der Fehlerbäume in Abschnitt 5.2.

5.1 Grundlagen der ITL

Die formale Semantik der FTA basiert auf einer Intervalltemporallogik (ITL, [Mos85]) mit kontinuierlichen Semantik ähnlich zum Duration Calculus (DC, [HZ92, CHR91]). Die Syntax und Semantik werden hier informell eingeführt und wir verweisen auf [STR02] für Details.

Die Menge der ITL-Formeln wird als einer Erweiterung der Formeln erster Stufe definiert. Temporale Formeln φ werden mit Formeln erster Stufe, aussagelogischen Verknüpfungsoperatoren und (neben weiteren) den folgenden Temporaloperatoren gebildet.

- $\boxplus \varphi$: in allen Anfangsstücken des Intervalls gilt φ
- $\boxtimes \varphi$: es gibt ein Anfangsstück des Intervalls, in dem φ gilt
- $\boxminus \varphi$: in allen Subintervallen gilt φ

- $\diamond \varphi$: es gibt Subintervalle, in denen φ gilt
- $\varphi ; \psi$: das Intervall kann so geteilt werden, dass φ im ersten und ψ im zweiten Teil gilt (sprich: ‘ φ chop ψ ’)

Die Semantik einer temporalen Spezifikation sind alle Intervalle \mathcal{I} die zum Zeitpunkt $a = 0$ beginnen und die Axiome der Spezifikation erfüllen. Wenn wir Korrektheits- und Vollständigkeitsbedingungen von Fehlerbäumen nachweisen, müssen diese über der Spezifikation gelten, d. h. die Bedingungen müssen für jedes Intervall \mathcal{I} der Spezifikation gelten.

5.2 FTA Semantik

Die Definition der Gatterbedingungen als Konjunktion bzw. Disjunktion der Unterereignisse ist aus verschiedenen Gründen nicht immer adäquat. Das Problem kann an einem der Knoten im Fehlerbaum aus Abbildung 3 veranschaulicht werden, das (mit entsprechenden Prädikaten) wie folgt beschrieben werden kann:

$$\text{release} = \text{true} \wedge \neg \text{closed}(\text{barriers}) \quad (1)$$

Wenn die Unterereignisse so definiert werden, dass jedes ein Konjunktionsglied enthält, dann beschreibt jedes Konjunktionsglied für sich genommen keinen Fehler mehr. Die Tatsache, dass die Schranken nicht geschlossen sind, ist für sich genommen kein Fehler. Ansonsten müssten die Schranken für immer geschlossen bleiben, um einen sicheren Betrieb zu garantieren. Analog ist die Tatsache, dass der Zug eine Freigabe erhalten hat und deshalb annimmt, dass der Bahnübergang gesichert ist, kein Fehler, ansonsten dürfte nie ein Zug über den Bahnübergang fahren. Deshalb ist in unserem Fehlerbaum dieses Ereignis durch ein ODER-Gatter zerlegt worden $(2) \vee (3)$. Entweder hat der Bahnübergang eine Freigabe gesendet, obwohl die Schranken nicht geschlossen sind, oder die Schranken öffnen, nachdem der Zug die Freigabe erhalten hat.

$$\text{crossing_sends_release} = \text{true} \wedge \neg \text{closed}(\text{barriers}) \quad (2)$$

$$\text{release} = \text{true} \wedge \text{crossing_open_signal} = \text{true} \quad (3)$$

Diese Zerlegung zeigt ein weiteres Problem für den Fall, dass die Semantik einfach als Disjunktion definiert wird: Es wird nicht betrachtet, dass Unterereignisse (Ursachen) zeitlich *vor* der Wirkung eintreten. Deshalb führen wir die Unterscheidung zwischen *Zerlegungsgatter* (kurz Z-Gatter) mit der klassischen booleschen Semantik und *Ursache/Wirkung-Gatter* (kurz UW-Gatter), bei denen die Ursachen vor der Wirkung eintreten müssen.

Die Tatsache, dass bei einem UW-Gatter die Ursache vor der Wirkung auftreten muss, impliziert, dass die Semantik nicht einfach als Äquivalenz formuliert werden kann. Stattdessen benötigen wir zwei Bedingungen. Die Korrektheitsbedingung garantiert, dass wenn die Ursache eintritt, auch die Wirkung folgt. Die Vollständigkeitsbedingung hingegen, dass alle Ursachen im Fehlerbaum aufgezählt wurden: die Wirkung darf nicht eintreten, ohne dass zuvor die Ursachen eintraten. Aus Symmetriegründen teilen wir auch die Bedingungen der Z-Gatter in Korrektheits- und Vollständigkeitsbedingungen auf. Die Implikation von den Ursachen zur Wirkung ist die Korrektheit, die Implikation von der Wirkung zur den Ursachen die Vollständigkeit. Schließlich müssen für das UW-AND-Gatter noch zwei Fälle unterschieden werden, je nachdem ob die Ursachen gleichzeitig eintreten müssen, oder nicht. Wir nennen den ersten Fall *synchron*, den zweiten *asynchron*.

Zusammenfassend erhalten wir 5 verschiedene Gattertypen: Z-ODER- und Z-UND-Gatter ($\hat{\wedge}_Z$, $\hat{\wedge}_Z$), UW-ODER ($\hat{\wedge}_{UW}$) und synchrones und asynchrones UW-UND-Gatter ($\hat{\wedge}_{UW}$, $\hat{\wedge}_A$). Die beiden BLOCK-Gatter sind analog zum UND-Gatter definiert und werden im Folgenden nicht weiter betrachten. Die Korrektheits- und Vollständigkeitsbedingungen dieser Gattertypen sind in Abbildung 4 für jeweils zwei Ursachen gezeigt. Die Verallgemeinerung auf $n > 2$ Ursachen sollte offensichtlich sein.

Die Bedingungen der Z-Gatter entspricht der klassischen booleschen Semantik, die in jedem Subintervall des Systemablauf gelten muss. Die Korrektheitsbedingung des UW-ODER-Gatters sagt aus, dass wenn in einem Systemablauf eine der Ursachen auftritt, auch die Wirkung in diesem Ablauf eintreten muss. Die Vollständigkeitsbedingung fordert, dass es nicht möglich ist, dass der Ablauf an einem Zeitpunkt t unterteilt werden kann, so dass keine der Ursachen vor t , die Wirkung aber sofort nach t eintritt.

Die Bedingungen für das synchrone und asynchrone UW-AND-Gatter sind analog zum UW-ODER-Gatter mit dem Unterschied, dass statt eine der Ursachen beide Ursachen (synchron oder asynchron) eintreten müssen.

Wir gehen davon aus, dass diese Formalisierung adäquat ist, da sie die Ursache/Wirkung-Beziehung berücksichtigt und die Bedeutung der minimalen Cut Sets erhält. D. h., wenn alle Vollständigkeitsbedingungen (nach Abbildung 4) eines Fehlerbaums nachgewiesen werden können und für jedes minimale Cut Set wenigstens ein Basisereignis auf jedem Ausführungspfad ausgeschlossen werden kann, dann tritt die Systemgefährdung niemals auf. Dieses Theorem wurde formal im Spezifikations- und Verifikationswerkzeug KIV [BRS⁺00] bewiesen. Eine detailliertere Beschreibung des Theorems, der FTA Semantik und Verweise auf verwandte Arbeiten findet sich in [STR02].

Gatter g	Korrektheit CORR(g)	Vollständigkeit COMPL(g)
	$\boxtimes (\varphi_1 \wedge \varphi_2 \rightarrow \psi)$	$\boxtimes (\psi \rightarrow \varphi_1 \wedge \varphi_2)$
	$\boxtimes (\varphi_1 \vee \varphi_2 \rightarrow \psi)$	$\boxtimes (\psi \rightarrow \varphi_1 \vee \varphi_2)$
	$\diamond (\varphi_1 \wedge \varphi_2) \rightarrow \diamond \psi$	$\neg (\neg \diamond (\varphi_1 \wedge \varphi_2) ; \diamond \psi)$
	$\diamond \varphi_1 \wedge \diamond \varphi_2 \rightarrow \diamond \psi$	$\neg (\neg \diamond \varphi_1 ; \diamond \psi)$
	$\diamond \varphi_1 \vee \diamond \varphi_2 \rightarrow \diamond \psi$	$\neg (\neg \diamond (\varphi_1 \vee \varphi_2) ; \diamond \psi)$

Abbildung 4: Semantik von Fehlerbäumen

6 Methodik

Ein wesentlicher Punkt bei der Entwicklung sicherheitsrelevanter Software ist, dass die Sicherheitsaspekte „von Anfang an“ in das System integriert und nicht nachträglich hinzugefügt werden („Safety must be designed into a System“ [Lev95]). Nachträgliches Hinzufügen von (Sicherheits-) Funktionalität macht ein System komplizierter und damit anfälliger für Fehlverhalten. Deshalb schlagen wir vor, die Sicherheitsanalyse während des Designs durchzuführen.

Wir unterteilen die Entwicklung eines Modells für sicherheitskritische Anwendungen in vier Phasen. Die erste Phase dient dazu, ein detailliertes Verständnis der Anwendungsdomäne zu erhalten. Die zweite Phase besteht aus der Analyse des Systems mit formalen Methoden und der FTA. Diese beiden Techniken werden in dieser Phase noch getrennt angewendet, um die verschiedenen Betrachtungsweisen der Techniken – die funktionale Sicht und die Betrachtung der Sicherheit – zu erhalten. In der dritten Phase werden die Ergebnisse und Erkenntnisse der beiden Techniken ausgetauscht. Dies führt zu Verbesserungen im Formalen Modell und in den Fehlerbäumen. Diese Pha-

se enthält auch die Validation der Fehlerbäume, die auf zwei verschiedenen Stufen, lose bzw. eng gekoppelte formale FTA genannt, durchgeführt werden kann. Schließlich fasst die vierte Phase die Analyseergebnisse zusammen.

6.1 Anforderungsanalyse (I)

Das Ziel der ersten Phase ist eine präzise Definition der *Systemanforderungen* und der *Systemgrenzen* zu erhalten. Dies erfordert ein gemeinsames Verständnis der Systementwickler und des Kunden von der Systemfunktionalität und -umfang. *Semi-formale* Spezifikationen (z. B. UML Diagramme) und ausführbare Prototypen zur Simulation sind sehr hilfreich, um dieses Ziel zu erreichen. Diese dienen auch als Ausgangspunkt für die Sicherheitsanalyse, wie in Abbildung 5 gezeigt. In [ORS⁺02] präsentieren wir eine Fallstudie, in der

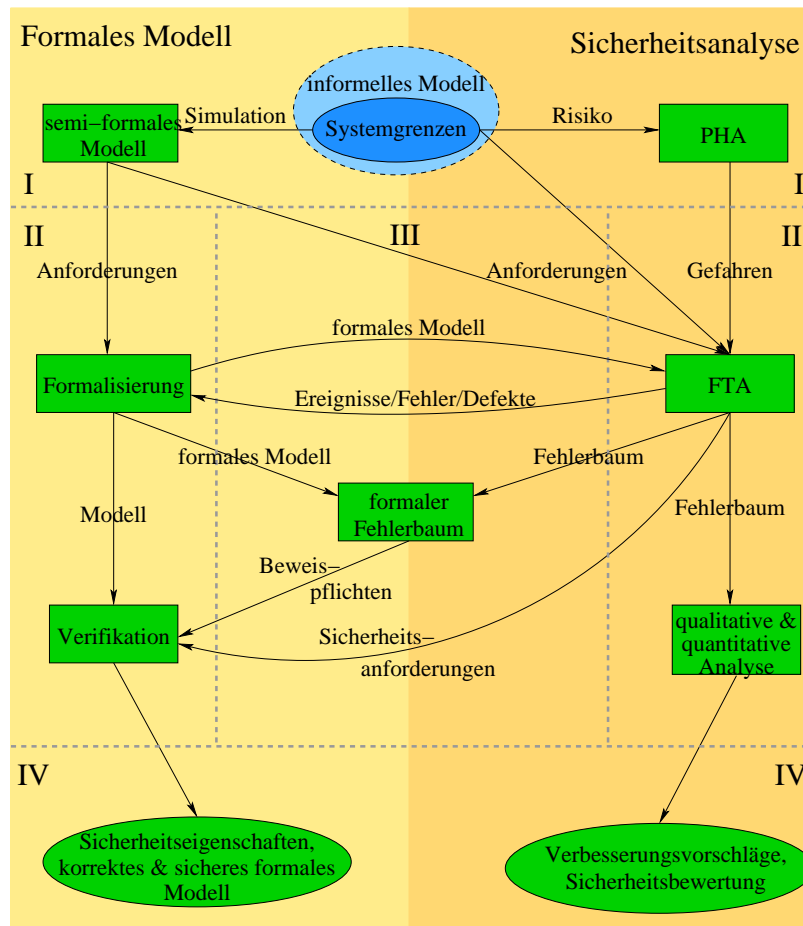


Abbildung 5: Entwicklungsprozess

die Simulation von Use-Cases durch ein ausführbares Modell gute Dienste erwies, um zu einem gemeinsamen Verständnis des Systems zu gelangen.

Zweitens schlagen wir eine *preliminary hazard analysis* (PHA) [Sto96] vor, um eine (vollständige) Liste der Systemgefährdungen aufzustellen, die als Ausgangspunkt für die FTA in der nächsten Phase dient. Diese Liste sollte alle kritischen Gefährdungen enthalten.

6.2 Sicherheitsanalyse (II)

Die zweite Phase startet mit der getrennten Erstellung der FTA und des formalen Modells. Unsere Erfahrung zeigt, dass es sehr wichtig ist, das formale Modell und die Sicherheitsanalyse unabhängig voneinander durchzuführen. Ansonsten ist die FTA zu sehr auf das formale Modell fixiert und verliert die Sicht auf mögliche Fehler, Defekte oder unspezifiziertes Verhalten. Jedoch ist eine unabdingbare Vorbedingung für diesen getrennten Ansatz, dass das selbe Verständnis des Systems und der Systemgrenzen bei den Entwicklern vorliegt. Deshalb ist die erste Phase des Entwicklungsprozesses sehr wichtig.

Für die formale Behandlung des Systems muss zuerst eine Spezifikationsprache zur *Formalisierung* ausgewählt werden. Diese muss von einer Verifikationsumgebung unterstützt werden, damit die Sicherheitsbedingungen, wie im nächsten Abschnitt beschrieben, nachgewiesen werden können. Die Formalisierung beschreibt die informellen Anforderungen mit einer formalen Sprache und ergibt das Systemmodell. Eine Validation gegen die informellen Anforderungen und dem semi-formalen Modell und der Nachweis gewünschter Systemeigenschaften ist notwendig, um ein korrektes formales Modell zu erhalten.

Die *FTA* muss alle Gefährdungen aus der PHA untersuchen. Daraus ergibt sich eine Menge von Fehlerbäumen, deren minimale Cut Sets der Ausgangspunkt für eine *qualitative Analyse* ist. Sind für die Basisereignisse die Ausfallwahrscheinlichkeiten bekannt, kann auch eine *quantitative* Bewertung der Systemsicherheit durchgeführt werden.

6.3 Integration (III)

Die dritte Phase bringt dann die Ergebnisse der formalen und der Sicherheitsanalyse zusammen. Der Fehlerbaum muss angepasst werden, um die Bedingungen des formalen Modells widerzuspiegeln. Wir beobachteten, dass in Fehlerbäumen häufig Ursache/Wirkung-Beziehungen gefunden werden, die im formalen Modell nicht gelten und deshalb geändert werden müssen. Auf der anderen Seite sind häufig Ereignisse die Komponentenfehler beschreiben und während der FTA entdeckt wurden, nicht im formalen Modell zu finden. Jedoch müssen alle Ereignisse der FTA im Modell ausgedrückt werden

können. Ist dies nicht der Fall, muss entweder das formale Modell um diese Ereignisse erweitert oder der Fehlerbaum muss so abgeändert werden, dass diese Ereignisse nicht mehr auftreten. Zusätzlich entscheidet ein Sicherheitsingenieur, welche Fehler im formalen Modell betrachtet werden müssen und welche vernachlässigt werden können. An dieser Stelle unterscheiden wir zwischen Hard- und Software Fehler.

Hardware- vs. Softwareausfälle. Die FTA eines softwarebasierten Systems deckt sowohl relevante Hardware- als auch Softwareausfälle auf. Hardwarefehler kann man nicht ausschließen, da sie nicht konstruktionsbedingt sind, sondern durch betriebsbedingte Defekte hervorgerufen werden. Deshalb kann man solche Ursachen nicht ausschließen, sondern nur die Ausfallwahrscheinlichkeit durch Wahl zuverlässiger Komponenten bzw. Einbau redundanter Komponenten verringern. An dieser Stelle hilft die FTA verschiedene Systemmodelle miteinander zu vergleichen und Vorschläge für Verbesserungen des Modells zu liefern. Um ein adäquates Systemmodell zu erhalten, müssen diese Fehler im formalen Modell berücksichtigt werden.

Für softwarebedingte Fehler sehen wir eine andere Vorgehensweise vor. Hier folgen wir Leveson: „... if design errors are found in the tree through this process, they should be fixed rather than left in the code and assigned a probability“ [Lev95]. Für Softwarekomponenten fordern wir deshalb die formale Verifikation.

Lose vs. eng gekoppelte formale FTA Nachdem das formale Modell und die FTA aneinander angepasst wurden, gibt es zwei Stufen der formalen FTA. Wenn die FTA und das formale Modell durch Austausch von Ergebnissen gekoppelt werden, nennen wir dies *lose gekoppelte formale FTA*. Die FTA wird dann hauptsächlich dazu genutzt um Sicherheitseigenschaften für das System abzuleiten, die dann über dem formalen Modell nachgewiesen werden. Obwohl dieser Austausch nicht formalisiert ist, zeigen verschiedene Fallstudien [ORS⁺02, RST00b] den gegenseitigen Nutzen. Die formalen Modelle erhalten eine Systematik um Sicherheitseigenschaften abzuleiten und die FTA eine formale Basis.

Zusätzlich zu den Sicherheitseigenschaften formalisiert die *eng gekoppelte formale FTA* die Ereignisse des Fehlerbaums über dem formalen Modells. Der Nachweis der Beweispflichten für die Fehlerbaumgatter aus Abschnitt 5 garantiert, dass der Fehlerbaum korrekt und vollständig ist. Wenn zusätzlich das formale Modell garantiert, dass mindestens ein Basisereignis jedes minimalen Cut Sets ausgeschlossen werden kann, kann die analysierte Gefährdung nicht eintreten. Dies wird durch das minimale Cut Set Theorem garantiert.

Model-Checking vs. Interaktive Verifikation Sowohl die lose als auch die eng gekoppelte formale FTA erfordert Verifikation. Wenn das System als endliches, diskretes Modell dargestellt werden kann, können die Beweispflichten für eingeschränkte Fehlerbäume automatisch mittels Modellprüfung nachgewiesen werden. In [TSR02] werden die angesprochenen Einschränkungen besprochen und die Beweispflichten für CTL Modellprüfer beschrieben.

Wenn die Modellprüfung nicht anwendbar ist, wird die interaktive Verifikation benötigt. Die interaktive Verifikationsumgebung KIV unterstützt den Nachweis von ITL Formeln, die für Nachweis der FTA Bedingungen erzeugt werden. Zur Zeit wird in KIV eine Spezifikationsunterstützung für Statecharts [HN96], wie in [BT02] beschrieben, erstellt, um dynamische, nicht zustandsendliche Systeme zu modellieren.

6.4 Ergebnisse (IV)

Schließlich fasst die vierte Phase die Ergebnisse der beiden Techniken zusammen. Der gesamte Entwicklungsprozess resultiert in einem korrekten und sicheren formalen Modell mit verifizierten Sicherheitseigenschaften. Zusätzlich zu diesen Sicherheitsgarantien für ein funktionierendes System (ohne Ausfälle), bewertet die FTA die Systemsicherheit bezüglich Komponentenausfällen und unterstützt durch qualitative und quantitative Analyse Verbesserungsvorschläge für das System zu erarbeiten. Z. B. wurde mit der formalen FTA verschiedene Verbesserungsvorschläge in einer Industriefallstudie [ORS⁺02] gefunden, die sehr wahrscheinlich im realen Projekt aufgegriffen werden.

7 Fazit

Für sicherheitskritische, eingebettete Systeme wurde eine Methode zur Erstellung hochsicherer Systemspezifikationen vorgestellt. Dies wird durch die Kombination zweier typischer Sicherheitsanalysetechniken erreicht: Fehlerbaumanalyse aus den Ingenieurwissenschaften und die formalen Methoden aus dem Softwareengineering. Die Kombination gibt der FTA eine formale Semantik die es erlaubt, Korrektheit und Vollständigkeit von Fehlerbäumen nachzuweisen und formale Modelle sicherheitstechnisch zu bewerten.

Zur Formalisierung der FTA führten wir die formalen Methoden, die herkömmliche FTA und die notwendigen formalen Grundlagen für die Formalisierung ein. Die Formalisierung ergibt Beweispflichten, deren erfolgreicher Nachweis die Korrektheit und Vollständigkeit des Fehlerbaum garantiert.

Der Fokus dieser Arbeit liegt auf der Präsentation der Methodik für die Anwendung der formalen FTA. Ein wesentlicher Punkt ist dabei, dass die

FTA und die formalen Methoden erst einmal unabhängig voneinander angewendet werden und sie dann schrittweise aneinander angepasst werden. Dieses Vorgehen erhält die verschiedenen Sichtweisen der beide Techniken auf das System und resultiert trotzdem in einem gemeinsamen Modell. Die Kopplung der FTA und der formalen Methoden kann auf zwei Stufen, die lose bzw. eng gekoppelte formale FTA, durchgeführt werden. Die erste Stufe tauscht Sicherheitseigenschaften aus, die über dem formalen Modell nachgewiesen werden. Die zweite formalisiert zusätzlich die Ereignisse des Fehlerbaums und weist ihn als korrekt und vollständig nach.

Für sicherheitskritische Systeme, wie in der Medizintechnik, Luft- und Raumfahrt, Eisenbahn oder Automobilelektronik schlagen wir die eng gekoppelte formale FTA vor. Sind die Systeme zustandsendlich, können die Beweispflichten automatisch mit Modellprüfung gezeigt werden. Für nicht zustandsendliche Systeme entwickeln wir Beweisunterstützung für die interaktive Verifikation der FTA-Bedingungen in KIV. Statechart-Spezifikationen [HN96] und ein entsprechender Beweiskalkül [BT02, BDRS02] wird integriert um dynamisch Systeme spezifizieren und verifizieren zu können.

Literatur

- [BDRS02] M. Balsler, C. Duelli, W. Reif, and G. Schellhorn. Verifying concurrent systems with symbolic execution. *Journal of Logic and Computation*, 12(4), 2002.
- [Bet] Betriebliches Lastenheft für FunkFahrBetrieb. Stand 1.10.1996.
- [BHL⁺96] J. Bowen, C. A. R. Hoare, H. Langmaack, E.-R. Olderog, and A. P. Ravn. A ProCoS II Project Final Report: ESPRIT Basic Reserach Project 7071. *EATCS-Bulletin*, 1996.
- [BR95] E. Börger and D. Rosenzweig. The WAM—definition and compiler correctness. In Christoph Beierle and Lutz Plümer, editors, *Logic Programming: Formal Methods and Practical Applications*, volume 11 of *Studies in Computer Science and Artificial Intelligence*. North-Holland, Amsterdam, 1995.
- [BRS⁺00] M. Balsler, W. Reif, G. Schellhorn, K. Stenzel, and A. Thums. Formal system development with KIV. In T. Maibaum, editor, *Fundamental Approaches to Software Engineering*, number 1783 in LNCS. Springer, 2000.

- [BS99] E. Börger and W. Schulte. A Programmer Friendly Modular Definition of the Semantics of Java. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java*. Springer LNCS 1523, 1999.
- [BT02] M. Balser and A. Thums. Interactive verification of statecharts. In *Integration of Software Specification Techniques (INT'02)*, 2002.
- [CCI98] CCIB-98-026. *Common Criteria for Information Technology Security Evaluation, Version 2.0*. ISO/IEC JTC 1, May 1998. available at <http://csrc.nist.gov/cc>.
- [CHR91] Zhou Chaochen, C. A. R. Hoare, and Anders P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, December 1991.
- [Coh97] R. Cohen. An ACL2 Formalization of the Java Virtual Machine. Technical report, Computational Logic Inc., 1997.
- [GDG⁺96] W. Goerigk, A. Dold, Th. Gaul, G. Goos, A. Heberle, F.W. von Henke, U. Hoffmann, H. Langmaack, H. Pfeifer, H. Rueß, and W. Zimmermann. Compiler correctness and implementation verification: The verifix approach. Technical Report LiTH-IDA-R-96-12, Linköping University, 1996.
- [HN96] D. Harel and A. Naamad. The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, October 1996.
- [HZ92] M. R. Hansen and Zhou Chaochen. Semantics and completeness of the Duration Calculus. In J. W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 209–225. Springer-Verlag, 1992.
- [ITS91] ITSEC. *Information Technology Security Evaluation Criteria, Version 1.2*. Office for Official Publications of the European Communities, June 1991.
- [JS99] L. Jansen and E. Schnieder. Referenzfallstudie Bahnübergang – Referenzfallstudie im Bereich Verkehrsleittechnik des DFG-SPP Softwarespezifikation. Technical report, Institut für Regelungs- und Automatisierungstechnik, <http://www.ifra.ing.tu-bs.de/m33/spezi/>, 1999. in German.

- [Kno89] R. E. Knowlton. Hazard and operability studies: The guide word approach. Technical report, Chematics International Company, Vancouver, 1989.
- [KT02] J. Klose and A. Thums. The STATEMATE reference model of the reference case study ‘Verkehrsleittechnik’. Technical Report 2002-01, Universität Augsburg, 2002.
- [Lev95] N. Leveson. *Safeware: System Safety and Computers*. Addison Wesley, 1995.
- [Moo88] J Moore. PITON: A Verified Assembly Level Language. Technical report 22, Computational Logic Inc., 1988. available at the URL: <http://www.cli.com>.
- [Mos85] B. Moszkowski. A temporal logic for multilevel reasoning about hardware. *IEEE Computer*, 18(2):10–19, 1985.
- [NN97] W. Naraschewski and T. Nipkow. Type inference verified: Algorithm w in isabelle/hol. In C. Paulin-Mohring, editor, *Proceedings of the International Workshop TYPES’96*, LNCS 1512, Berlin, 1997. Springer. To appear.
- [NvO98] T. Nipkow and D. von Oheimb. Java light is Type-Safe – Definitely. In *25th ACM Symposium on Principles of Programming Languages*. ACM, 1998.
- [ORS⁺02] F. Ortmeier, W. Reif, G. Schellhorn, A. Thums, B. Hering, and H. Trappschuh. Safety analysis of the height control system for the Elbtunnel. In *Proceedings SAFECOMP 2002*, pages 296 – 308, Catania, Italy, 2002. Springer LNCS 2434.
- [Rei79] D. Reifer. Software failure modes and effects analysis. *IEEE Transactions on Reliability*, 28(3):147 – 249, August 1979.
- [RST00a] W. Reif, G. Schellhorn, and A. Thums. Formale Sicherheitsanalyse einer funkbasierten Bahnübergangsteuerung. In E. Schnieder, editor, *Forms2000 – Formale Techniken für die Eisenbahnsicherung*, volume Reihe 12, Nr. 441 of *Fortschritt-Bericht VDI*, 2000.
- [RST00b] W. Reif, G. Schellhorn, and A. Thums. Safety analysis of a radio-based crossing control system using formal methods. In E. Schnieder and U. Becker, editors, *9th IFAC Symposium Control in Transportation Systems 2000*, June 2000.

- [SA98] G. Schellhorn and W. Ahrendt. The WAM Case Study: Verifying Compiler Correctness for Prolog with KIV. In W. Bibel and P. Schmitt, editors, *Automated Deduction — A Basis for Applications*, volume III: Applications, chapter 3: Automated Theorem Proving in Software Engineering. Kluwer Academic Publishers, 1998.
- [Sto96] N. Storey. *Safety-Critical Computer Systems*. Addison-Wesley, 1996.
- [STR02] G. Schellhorn, A. Thums, and W. Reif. Formal fault tree semantics. In *Proceedings of The Sixth World Conference on Integrated Design & Process Technology*, Pasadena, CA, 2002.
- [TSR02] A. Thums, G. Schellhorn, and W. Reif. Comparing fault tree semantics. In D. Haneberg, G. Schellhorn, and W. Reif, editors, *FM-TOOLS 2002*, Technical Report 2002-11, pages 25 – 32. Universität Augsburg, 2002.
- [VGRH81] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *Fault Tree Handbook*. Washington, D.C., 1981. NUREG-0492.
- [vON99] D. von Oheimb and T. Nipkow. Machine-checking the Java Specification: Proving Type-Safety. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java*. Springer LNCS 1523, 1999.