Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterseminar SS 2014

Vitalij Stepanov

Modellbasierte Zuverlässigkeitsanalyse

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Vitalij Stepanov

Modellbasierte Zuverlässigkeitsanalyse

Masterseminar SS 2014 eingereicht im Rahmen der Master of Science Informatik

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Buth

Eingereicht am: 31. August 2014

Inhaltsverzeichnis

1 Einführung

Modellbasierte Entwicklung gewinnt immer mehr an Bedeutung, weil sie viele Vorteile bietet. Dazu zählt zum Beispiel die Entdeckung von Fehlern bereits in der Designphase. In der Luftfahrtindustrie wird das V-Modell für den Prozess der Systementwicklung verwendet [Lan08] mit der Konsequenz, dass später entdeckte Fehler höhere Kosten verursachen. Die Entwicklung von komplexen sicherheitskritischen Produkten und die damit verbundene Komplexität der Integration und Verifikation ist eine Herausforderung für die Entwicklung insbesondere bezüglich der Verwaltung der Artefakte und der Verifikation. Daher strebt die Industrie nach transparenteren Vorgängen bei der Entwicklung.

1.1 Motivation

Oftmals werden einzelne Komponenten des Gesamtsystems von unterschiedlichen Abteilungen mit den bevorzugten Werkzeugen entwickelt, was die Kollaboration bzw. die Validierung deutlich erschwert. Auch die anschließende Zertifizierung erfordert Nachweise von Traceability während der Entwicklung. Die dokumentbasierte Spezifikation wird in menschlicher Sprache erfasst, ist im Vergleich zu den formalen Sprachen nicht eindeutig. Daher ist die Verifikation sehr fehleranfällig und zeitintensiv [XZw12]. Spezifikationen, die beispielsweise mittels Unified Modeling Language (UML) oder Systems Modeling Language (SysML) beschrieben wurden, sind dagegen eindeutig und lassen sich einfach maschinell verarbeiten. Besonders bei der Entwicklung von sehr komplexen und ausfallkritischen Produkten, wie z.B. die Flugzeugkomponenten, spielen die Interoperabilität und die Traceability eine entscheidende Rolle. Um die Verifizierbarkeit von Produktqualität gegen Anforderungen übersichtlich und transparent darzustellen, sind domänenübergreifende Informationssysteme erforderlich, die Ergebnisse von einzelnen Entwicklungszyklen miteinander verknüpfen. Besonders bei der plattformunabhängigen, verteilten Integration bzw. Verlinkung von Modellen oder deren Komponenten zu den weiteren Werkzeugen werden offene Standards, wie beispielsweise Open Life Cycle Management (OSLC), benötigt [OS14].

1.2 Einführung in die Zuverlässigkeitsanalyse

Mittels der Zuverlässigkeitsanalyse werden potentielle Schwachstellen des Produktes identifiziert. Die Zuverlässigkeit kann entweder empirisch, durch die Ermittlung der Ausfallhäufigkeit, oder analytisch, aus der Ableitung der Zuverlässigkeitswerte der Teile des Produkts, ermittelt werden. Der von *SAE International* definierte Standard beschreibt in der ARP4761 die Notwendigkeit der Verwendung von Fehlerbaumanalyse zur Bewertung der Zuverlässigkeit[SAE96].

Fehlerbaumanalyse Die Fehlerbaumanalyse (eng. Fault Tree Analysis (FTA)) ist ein *top-down* basierter deduktiver Ansatz, mit dem Ziel die Ausfallursachen im Sinne des Beitrags einzelner Komponenten und Umstände zu ermitteln [SAE96]. Unter bestimmten Randbedingungen kann dann eine Wahrscheinlichkeit eines Ausfalls ermittelt werden. Der Fehlerbaum besteht aus den Elementen, deren Bedeutung nachfolgend beschrieben wird. Ein so genanntes *Top-Ereignis* bildet den Wurzelknoten des Fehlerbaums und steht für die negierte funktionale Anforderung, die im normalen Fall nicht eintreten darf. Aus den *Gates*, die in klassischen Fault Trees für Ausdrucke AND OR oder NOT stehen können, werden logische Aussagen formuliert. Jeder Eingang des Gates wird mit einem weiteren Gate, einem *Basisereignis* oder einem *House-Ereignis* abgeschlossen. Das *Basisereignis* beschreibt den Fehler, der zum Ausfall der Funktion beiträgt. Das *House-Ereignis* ist keine Fehlerquelle und kann im gewöhnlichen Betrieb auftreten. Kann aber in Verbindung mit einem Ereignis zu der Fehlerursache führen. Als *Cut Set* wird die Menge der Basisereignisse und House-Ereignisse bezeichnet, die auftreten und zum Ausfall des Systems bzw. einer bestimmten Funktion führen. Bei einem *Minimal Cut Set* handelt es nach [Kec02] um einen *Cut Set*, bei dem jeder Basisereignis eine notwendige Bedingung zum Ausfall des Systems darstellt. Das heißt, dass bei der Entfernung von einem beliebigen Element aus dem *Minimal Cut Set* das Wurzel-Fehler-Ereignis nicht mehr auftritt.

Fault Injection Als *Fault Injection* wird ein Verfahren bezeichnet, das das künstliche Einbringen von Fehlern zur Verbesserung der Robustheit von Produkten verwendet [JH05]. Zu den Zielen der Fehlerinjektion zählen die Identifizierung von Flaschenhälsen, Untersuchung von Verhalten bei Anwesenheit der Fehler oder die Bewertung der Effektivität von Mechanismen zur Fehlertoleranz [HTI97]. Die Fehlerinjektion kann sowohl in Software als auch in Hardware realisiert werden. Bei der hardwarebasierten Fehlerinjektion wird zwischen Injektion mit Kontakt und kontaktlosen unterschieden. Die hardwarebasierte Fehlerinjektion kann erst an einem Prototyp durchgeführt werden. In diesem Projekt wird die Fehlerinjektion auf Modellebene verwendet, was die Entdeckung und das Eliminieren von Designfehler bereits in der Konzeption bzw. Entwurfsphase ermöglicht.

2 Rückblick: Projekt 2

Während dieses Projektes wurde der Ansatz der automatischen Fehlerinjektion für Matlab Simulink erarbeitet und in einer prototypischen Implementierung auf die Machbarkeit untersucht [Ste14]. Dabei ist eine flexible, einfach erweiterbare Toolbox zur automatisierten Injektion von Fehlverhalten entstanden, die durch sein generischen Ansatz die Konfigurierbarkeit bei der Fehlerinjektion vereinfacht. Die Fehlerinjektion wird mit Hilfe des *Design Verifier* [The11] auf die Verletzung von Anforderungen überprüft. Aus den Ergebnissen der Verifikation werden Mengen von Basisereignissen ermittelt, die einen Fehler auslösen können. Anschließend werden diese zu den Fehlerbäumen zusammengefasst.

Zur Modellierung des unerwünschten Verhaltens wurde die Simulink Bibliothek um die für die Injektion erforderlichen Fehlerblöcken erweitert. Der von [PM01] vorgeschlagene Ansatz wurde zum größten Teil übernommen. Die *Omit*-Klasse steht dabei für zu spät erhaltene Nachrichten. Bei einem unendlichen *Delay* kommt die Nachricht nie an. Die Klasse *Commit* steht für zu früh erhaltene Nachrichten. Mittels der *Value Range* Klasse können die Werte der Nachricht verfälscht werden. Die Matlab Simulink Bibliothek wurde entsprechend der Anleitung [Mat14] um die vorgestellten Fehlerklassen erweitert. Der zusätzliche Block Fault Injection Point (FIP) dient der Annotation, um die fehlerbehafteten Stellen zu markieren. Er enthält einen Eingang, einen Ausgang und reicht das Signal weiter, ohne es zu verändern. Der Benutzer kann durch die Verwendung dieser Blöcke die fehlerbehafteten Stellen markieren.

Die einfache, flexible Konfigurierbarkeit wird dadurch erreicht, dass ein zu dem Simulink Modell zugehörige Fehlermodell extern abgespeichert wird. Der Anwender erstellt das Fehlermodell, indem er eine sogenannte *White List* festlegt. Die *White List* enthält für jeden FIP Block die Liste mit den Namen der Fehlerarten aus der Simulink Bibliothek, die für diesen Block für Injektion verwendet werden dürfen. Durch das Setzen des Attribute *enabled* können Fehler aktiviert oder deaktiviert werden. Mit dem Attribute *failure_rate* kann die Fehlerrate definiert werden, die zur Berechnung der Ausfallwahrscheinlichkeit verwendet wird. Einerseits kann der Validierer die *White List* mittels der dafür erstellten GUI direkt im Modell editieren. Anderseits

kann die *White List* aus der Ergebnissen von Particular Risk Analysis (PRA) generiert werden, bei der die Beschädigungen z.B. durch die Explosion eines Triebwerks berechnet werden.

Der Algorithmus der Fehlerinjektion ist ähnlich zu dem in [XZw12] erwähnten realisiert. Im ersten Schritt wird aus der *White List* kombinatorisch die Liste aus Basisereignissen erzeugt. Diese Liste entspricht allen möglichen Szenarien für die Einzelfehler. Im zweiten Durchgang werden Einzelfehler wieder mit den Basisereignissen kombiniert, um Doppelfehler zu erzeugen. Dabei darf in einem Szenario an einem FIP nur ein Fehler auftreten. So werden die Szenarien der Größe S_i aus S_{i-1} erzeugt, bis die maximale bzw. vorgegebene Größe erreicht wurde. Bei erschöpfender Verifikation ist die Anzahl der Injektionsszenarien von der Anzahl n de Fehlerarten und der Anzahl m der FIP's abhängig und besteht aus maximal n^m Szenarien. Durch die Begrenzung der Anzahl der auftretenden Fehler kann der Algorithmus vorzeitig abgebrochen werden, ohne an die Limits zu kommen.

Jeder dieser Injektionsszenarios wird anschließend mit dem Design Verifier auf die Verletzungen gegen Anforderungen überprüft. Hat er eine Verletzung gefunden, dann ist das Szenario ein Cut Set. Zur Berechnung der *minimalen Cut Sets* wird die Besonderheit des Injektionsalgorithmus verwendet, indem zuerst die Einzelfehler injiziert werden. Jeder injizierte Einzelfehler, der zur Verletzung der Eigenschaft führt, ist ein Ereignis, welches nach der Definition ein minimaler Cut Set ist. Er wird dann in die *ViolationList* eingetragen, die Cut Sets dem Namen der Eigenschaft zuordnet. Wird die gleiche Eigenschaft bei einem Doppelfehler verletzt und ein echte Teilmenge des Cut Sets in der Liste vorhanden ist, dann ist es kein *minimaler Cut Set* und kann verworfen werden. So entsteht für jede Eigenschaft ein Fehlerbaum.

Die Erkenntnisse aus dem Projekt zeigen trotzt des Performanceproblems das Potenzial des Verfahrens in Bezug auf die Automatisierung und die Konfigurierbarkeit bei der Verifikation. Während des Projektes wurden zusätzliche Anforderungen identifiziert, die in der Masterarbeit umgesetzt werden. Eine erforderliche Eigenschaft ist die Möglichkeit der Parametrierung von Fehler durch den Design Verifier. So kann beispielsweise ein Fehler mit dem *Enable*-Signal durch den Design Verifier aktiviert bzw. deaktiviert werden. Dadurch können kausale Abhängigkeiten bei der Verifikation entdeckt werden. Das Modell, an dem das Verfahren validiert werden soll, verwendet sehr stark das Konzept der Zusammenfassung von Signalen zu einem Bus und muss daher auch implementiert werden, weil die Beschädigung des Buses alle Nachrichten die dort verschickt werden betrifft.

3 Masterarbeit

Die Masterarbeit wird im Rahmen des Projektes CRitical sYSTem engineering AcceLeration (CRYSTAL) durchgeführt [MVS+14], bei dem die Interoperabilität und die Traceability bei der Entwicklung von sicherheitskritischen Systemen verbessert werden soll. Dabei werden domänenspezifische Werkzeuge, wie z.B. Matlab Simulink und FaultTree+, zu der so genannten Reference Technology Plattform (RTP) verbunden, um damit deren Möglichkeiten zu erweitern [KEOS14; VKE12].

3.1 Zielsetzung

Die Masterarbeit kann thematisch in zwei Bereiche aufgeteilt werden. Im Bereich Zuverlässigkeitsanalyse soll ein generisches Verfahren zur Fehlerinjektion für die Matlab-Simulink Modelle entwickelt werden. Dabei wird das Modell mit dem nominalen Verhalten um ein dynamisches Fehlermodell ergänzt und gegen Erfüllung von Anforderungen verifiziert. Die Ereignisse, die zu den Verletzungen der Anforderungen führen, werden zu einem Fehlerbaum zusammengefasst. Im Bereich Interoperabilität soll ein prototypischer, OSLC basierter Adapter für Matlab entwickelt werden, der die Ergebnisse an FaultTree+ oder ein weiteres Werkzeug zur Zuverlässigkeitsanalyse zur Verfügung stellt. Weiterhin soll der Adapter mit der IBM Jazz Plattform kompatibel sein, um die Kollaboration mit den bereits etablierten LifeCycle-Werkzeugen zu gewährleisten [IBM14]. Anschließend wird das entwickelte Verfahren anhand eines Anwendungsfalls validiert. Im Anwendungsfall wird ein von [Tun11] entwickelte Modell verwendet, das die Beschreibung vom Nominalverhalten und die Anforderungen eines Environmeantal Control System (ECS) enthält. Das Modell wird mit dem Fehlverhalten erweitert und analysiert.

3.2 Vorgehen

Zuerst werden die Rollen identifiziert und Anwendungsszenarien festgelegt. Im ersten Szenario verwendet der Benutzer die Fehlerinjektion lokal in Matlab Simulink direkt. Im zweiten Szenario wird die Fehlerinjektion aus einer anderen Anwendung über OSLC Schnittstelle

durchgeführt. Mit den Erfahrungen aus dem Projekt 2 und dabei getätigten Recherche wird dann eine Systemarchitektur Entwickelt, die den definierten Anforderungen entspricht. Für jede der zu integrierenden Komponente wird eine Softwarearchitektur ausgearbeitet. Nach der Realisierung wird das entwickelte Informationssystem an einem realistischen Modell des ECS validiert.

3.2.1 Konzept des Informationssystems

In diesem Abschnitt der Masterarbeit werden die grundlegenden Konzepte des RTP erarbeitet. Weiterhin wird die Problematik der Parallelität und Datenkonsistenz bei der Integration von *single user*-Werkzeugen untersucht.

Systemarchitektur Die Abbildung 3.1 zeigt die schematische Darstellung der Systemarchitektur. Sie besteht aus einem Rechner mit der Matlab-Anwendung und einem Rechner mit „FaultTree+“. Die Modelle mit dem nominalen Verhalten sind auf dem Matlab-Rechner enthalten. Durch die Ausführung der Fault Injection werden Fehlerbäume generiert und lokal abgespeichert. Um die Fehlerbäume zu der Analyse nach FT+ zu transportieren, wird ein OSLC basierte Webanwendung entwickelt, die alle Artefakte zur Verfügung stellt. Die Webanwendung stellt mittels der Component Object Model (COM)-Schnittstelle die Verbindung zu Matlab her. Eine in FT+ integrierte Client-Anwendung ruft die verfügbaren Artefakten ab bzw. startet die Fehlerinjektion. Anschließend können die Fehlerbäume importiert werden, um weitere Analysen durchzuführen.

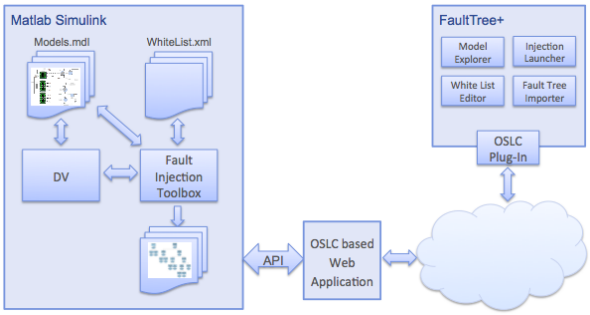

Abbildung 3.1: Systemarchitektur des Informationssystems

Softwarearchitektur Dieser Abschnitt wird sich der Erarbeitung und detaillierten Beschreibung des inneren Aufbau von Komponenten widmen. Für jede der drei Komponenten wird eine eigen Softwarearchitektur erarbeitet, die die Systemarchitektur verfeinert. Die Fehlerinjektion ist Matlab-spezifisch, daher wird sie zum größten Teil innerhalb Matlab realisiert. Die Synchronisierung und Problemme der Parallelität sind Bestandteil des OSLC Adapters. Der Plug-In für FaulTree+ stellt eine OSLC Client-Anwendung dar. Anschließend wird die das Zusammenspiel von Komponenten auf das Erfüllen von Anforderungen und die Eignung für die Anwendungsszenarien bewertet.

3.2.2 Realisierung

In diesem Abschnitt der Arbeit findet die praktische Umsetzung statt. Dafür werden die bereits im vorangegangen Abschnitt vorgestellten Softwarearchitekturen implementiert und anschließend dem Integrationstest unterzogen.

Matlab Toolbox Im Projekt 2 wurden bereits die ersten Implementierungsversuche durchgeführt und das Verfahren der Fehlerinjektion auf Machbarkeit der technischen Umsetzung überprüft. Der Algorithmus der Fehlerinjektion ist nicht sehr effektiv (siehe Abschnitt 2), die Architektur wird aber so entworfen, dass der Algorithmus zu einem späteren Zeitpunkt mit einem effizienteren ausgetauscht werden kann. Der bisherige Fokus lag auf dem Fehlerinjektionsalgorithmus und der Anwendung der Fehlerinjektion innerhalb von Matlab/Simulink. Die Toolbox wird daher um die Schnittstelle für OSLC Adapter erweitert, so dass die Verwendung der Funktionalität auch außerhalb möglich ist.

OSLC Adapter Der OSLC Standard basiert auf dem HTTP Protokoll, das die Ressourcen mittels eines Web-Servers zur Verfügung stellt [OS14]. Die direkte Integration eines Webservers in Simulink erwies sich als sehr umständlich, daher wird die *COM Automation Server*-Schnittstelle verwendet. Es sind bereits OSLC Bibliotheken für Java und C# vorhanden, die die Implementierung deutlich vereinfachen. Da die Bibliotheken für FaultTree+ und für Matlab der Bestandteil des DotNet-Frameworks sind, wird für die Entwicklung die Bibliothek *OSLC4Net* [Cod14] in Verbindung mit ASP.NET verwendet. Die Anbindung einer *single-user* Anwendung an das Netzwerk erfordert besondere Aufmerksamkeit in Bezug auf Parallelität, Synchronisierung und Datenkonsistenz. In diesem Arbeitsabschnitt wird der Fokus auf diese Problematik gerichtet.

Plug-In für FaultTree+ Dieses Werkzeug wird verwendet, um die Ergebnisse der Fehlerinjektion als Fehlerbaum grafisch darzustellen und zusätzlich die Zuverlässigkeitsanalysen durchzuführen. Um die Daten der Fehlerinjektion zu importieren, wird ein Plug-In entwickelt, der die Verbindung mit dem OSLC-Adapter herstellt. Mit dem Plug-In wird der Anwender in der Lage sein, eine neue *White List* zu erstellen oder vorhandene zu editieren und anschließend die Fehlerinjektion auszuführen. Zusätzlich hat er die Möglichkeit, die Artefakte aufzulisten und als Fehlerbaum in die Anwendung zu importieren. Oftmals ist die Überprüfung einer Fehlfunktion im ursprünglichen Designmodell erforderlich. Dafür wird jeder Knoten des Fehlerbaums einen Link enthalten, durch den direkt zu dem Matlab-Designmodel gelangen kann, der genau dieses Verhalten aufweist. Dadurch kann die Traceability demonstriert werden.

3.2.3 Validierung

Dieser Abschnitt beschreibt welche Methodik wird verwendet um sicherzustellen, ob das entwickelte Produkt den Anforderungen entspricht und nützlich ist. Weiterhin wird untersucht ob die Performanceproblemmatik bewältigt wurde. Um die Arbeit bei den realistischen Bedingungen zu testen, wird das von [Tun11] erstellte Modell verwendet.

Validierungsmodell Dieses Modell bildet die in der Luftfahrtindustrie übliche Integrated Modular Avionics (IMA)-Architektur ab. Diese Architektur besteht aus den Central Processor Input/Output Module (CPIOM), die als Steuergeräte agieren. Die Remote Data Concentrator (RDC) empfangen von den angeschlossenen Aktoren und Sensoren die Daten und übermitteln sie über ein echtzeitfähiges Netzwerk an das CPIOM. Das CPIOM hat unter anderem die Funktion des Monitors. Er überwacht die Komponenten des Systems und soll die festgestellten Ausnahmezustände melden. Weiterhin enthält das Modell funktionale und nichtfunktionale Anforderungen, deren Gültigkeit mittels *Design Verifiers* überprüft werden kann. Für die Validierung wird dieses Modell um einen Fehlermodell ergänzt und die Fehlerinjektion durchgeführt.

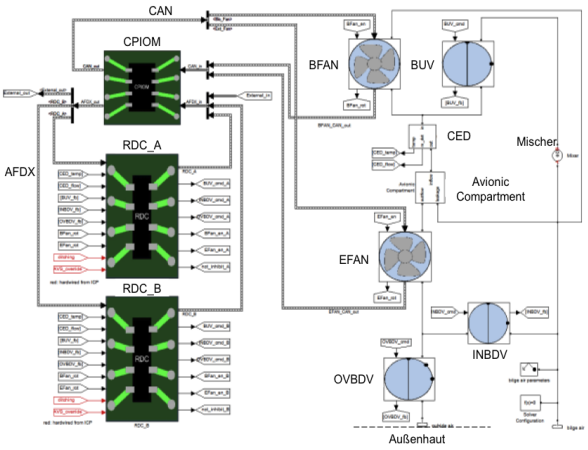

Abbildung 3.2: Simulink Modell eines Environmental Control Systems (ECS)

Validierungskriterien Anschließend erfolgt eine kritische Betrachtung der Ergebnisse. Dabei werden unterschiedlichen Kriterien untersucht. Zuerst wird der resultierende Fehlerbaum mit einem durch manuelle Analyse erstellten Fehlerbaum verglichen, um die Aussagekraft von Ergebnissen zu ermitteln. Aufgrund des kombinatorischen Problems soll auch die Laufzeit der Analyse bewertet werden. Die Laufzeit kann durch die Erhöhung der Ressourcen bzw. die Reduzierung der Modellgröße beinflusst werden. Durch die Validierung sollen die Grenzen des Verfahrens ermittelt werden. In einem weiteren schritt wird die Traceability und Interoperabilität untersucht. Anschließend wird auch die Usability bewertet, bei der die Intuitive Bedienung bewertet wird.

4 Ausblick

Während der Recherche und der Befragung von Stakeholder wurden mehrere Lücken entdeckt, die ich mit meiner Masterarbeit schließen möchte. Dieser Abschnitt beschreibt die daraus resultierenden Chance und Risiken.

4.1 Chancen

Derzeit existiert keine automatisierte Zuverlässigkeitsanalyse für Matlab/Simulink Modelle. Die Fault Injectin Toolbox wird die Zuverlässigkeitsanalyse für Simulink Modelle ermöglichen. Durch die Vereinfachung der Vorgängen wird die Fehleranfälligkeit reduziert, was bisher durch sehr pragmatische und konzentrierte Arbeitsweise erreichbar war.

Im Bereich der Interoperabilität existiert noch keine OSLC Spezifikation für Safety Analyse (SA). Eine hohe Granularität der Datendarstellung bietet viele Möglichkeiten zur Wiederverwendung und der Verlinkung von Daten. Die Datenkonsistenz wird durch die Vermeidung von Kopien erreicht. Zusätzlich werden auch Plattformübergreifende Analysen möglich. So kann beispielsweise die *Impact Analyse* durchgeführt werden, die die Auswirkungen der Veränderungen von Anforderungen sichtbar macht.

4.2 Risiken

Zur Zeit existiert kein effektiveres Verfahren zur Lösung der vollständigen Verifikation mittels Design Verifier. Das kombinatorische Problem stellt daher ein Risiko, was möglicherweise die Einschränkungen auf die Modellgröße mit sich bringt.

In der Masterarbeit sollen viele unterschiedliche Werkzeuge integriert werden. Dies erfordert viel Zeit für die Erforschung der Dokumentation. Aufgrund der fehlenden Erfahrungen kann der Umfang nicht genau abgeschätzt werden. Auch die Komplexität des Gesamtsystems lässt sich am Anfang sehr schwer einschätzen.

5 Zusammenfassung

Ziel der Masterarbeit ist die Integration von entwickelten und bereits vorhandenen Werkzeugen zu einem Informationssystem. Die Automatisierung der Arbeitsabläufe bei der Zuverlässigkeitsanalyse von sicherheitskritischen Komponenten soll die Interoperabilität und die Traceability verbessern. Während des Projekts wurde bereits die *Fault Injection Toolbox* für Matlab Simulink prototypisch umgesetzt. Die Implementierung ist in der Lage, aus der *White List* einen *Injektionsplan* zu erstellen. Jede der darin enthaltenen Injektionsszenarien wird nacheinander ausgeführt und das Modell auf die Verletzung der Anforderungen verifiziert. Aus den Ergebnissen der Verifikation werden *minimale Cut Sets* generiert, die zu den Fehlerbäumen zusammengefasst werden. In der Masterarbeit wird diese Implementierung weiterentwickelt. Während des Projekts entdeckte zusätzliche Anforderungen werden auch umgesetzt. Anschließend wird das System zu einem Workflow aus mehreren Werkzeugen verbunden. Die Werkzeugkette wird generisch implementiert, sodass sie ohne großen zusätzlichen Aufwand erweitert werden kann. Anschließend wird das Produkt anhand eines realistischen Modells validiert. Dabei wird die Aussagekraft von Ergebnissen der automatischen Zuverlässigkeitsanalyse ermittelt. Aufgrund des kombinatorischen Problems soll auch die Laufzeit der Analyse bewertet werden. Das Verfahren hat Potential den LifeCycle der Entwicklung deutlich zu verbessern und transparenter zu gestalten.

Literaturverzeichnis

[Cod14] CodePlex. OSLC4Net project, 2014. [Online] http://oslc4net.codeplex.com/.

[HTI97] Mei-Chen Hsueh, Timothy K. Tsai, and Ravishankar K. Iyer. Fault injection Techniques and Tools. In *Computer (Volume:30 , Issue: 4)*, 1997.

[IBM14] IBM. IBM Jazz Products, 2014. [Online] http://jazz.net/products.

[JH05] Anjali Joshi and Mats P.E. Heimdahl. Model-Based Safety Analysis of Simulink Models Using SCADE Design Verifier. In *SAFECOMP*, volume 3688 of *LNCS*, page 122, 2005.

[Kec02] Dimitri B. Kececioglu. *Reliability Engineering Handbook* -. DEStech Publications, Inc, Lancaster, revised edition edition, 2002.

[KEOS14] Omar Kacimi, Christian Ellen, Markus Oertel, and Daniel Sojka. Creating a reference technology platform. In *Proceedings of MODELSWARD 2014*, pages 645–652. SCITEPRESS, 1 2014.

[Lan08] Rene Langermann. *Beitrag zur durchgängigen Simulationsunterstützung im Entwicklungsprozess von Flugzeugsystemen.* PhD thesis, Institut für Luft- und Raumfahrtsysteme, 2008.

[Mat14] Mathworks. Add libraries to the library browser, 2014. http://www.mathworks.de/de/help/simulink/ug/adding-libraries-to-the-library-browser.html.

[MVS+14] Andreas Mitschke, Luciana Lo Verde, Wladimir Schamai, Stefan Richter, Claudio Pessa, Ivo Viglietti, and Francesco Brunetti. CRYSTAL aerospace use case description. Technical report, CRYSTAL, 2014.

[OS14] Open-Services. OSLC Specifications, 2014. [Online] http://open-services.net.

[PM01] Y. Papadopoulos and M. Maruhn. Model-based synthesis of fault trees from matlab-simulink models. In *Dependable Systems and Networks, 2001. DSN 2001. International Conference on*, pages 77–82, July 2001.

[SAE96] SAE International. *ARP4761 Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, 1996.

[Ste14] Vitalij Stepanov. Entwicklung einer Matlab Fault Injection Toolbox zur automatisierten Generierung von Fehlerbäumen. Master's thesis, Hochschule für Agewandte Wissenschaft, 2014.

[The11] The MathWorks, Inc. *Design Verifier User Manual*, 2011.

[Tun11] Marvin Tunnat. Integration modellbasierter Methoden in den Entwicklungsprozess hybrider Flugzeugregelungssysteme am Beispiel des Ventilation-Control-System. Master's thesis, Techische Universität Hamburg Harburg, 2011.

[VKE12] Parham Vasaiely, Andreas Keis, and Rainer Ersch. The CESAR and MBAT Interoperability Specification. Technical report, EADS UK Ltd, 2012.

[XZw12] Wang Xi and Xu Zhong-wei. A Novel Fault Injection Algorithm for Safety Analysis. In *Engineering and Technology (S-CET), 2012 Spring Congress on*, pages 1–4, May 2012.

Glossar

ARP4761
: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment. 2

COM
: Component Object Model. 6

CPIOM
: Central Processor Input/Output Module. 8

CRYSTAL
: CRitical sYSTem engineering AcceLeration. 5

ECS
: Environmeantal Control System. 5, 6

FIP
: Fault Injection Point. 3, 4

FTA
: Fault Tree Analysis. 2

IMA
: Integrated Modular Avionics. 8

Interoperabilität
: ist die Fähigkeit unabhängiger, heterogener Systeme, möglichst nahtlos zusammenzuarbeiten, um Informationen auf effiziente und verwertbare Art und Weise auszutauschen bzw. dem Benutzer zur Verfügung zu stellen, ohne dass dazu gesonderte Absprachen zwischen den Systemen notwendig sind. 1, 5

LifeCycle
: beschreibt die Phasen die das Produkt während seiner Existenz erlebt. 5

OSLC
: Open Life Cycle Management. 1, 5, 6, 9

PRA
: Particular Risk Analysis. 4

RDC
: Remote Data Concentrator. 8

RTP
: Reference Technology Plattform. 5, 6

SysML
: Systems Modeling Language. 1

Traceability
: die Rückverfolgbarkeit der Vorgänge Platformübergreifend zur Analyse von Erfüllung von Qualitätskriterien. 1, 5

UML
: Unified Modeling Language. 1