

**MBEES 2018**  
**MBEES 2018**  
**MBEES 2018**  
**MBEES 2018**  
**MBEES 2018**

## **Tagungsband des Dagstuhl-Workshops**

**Modellbasierte Entwicklung  
eingebetteter Systeme XIV**

**Matthias Riebisch  
Michaela Huhn  
Hardi Hungar  
Sebastian Voss**



# **Tagungsband**

## **Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme XIV**

**Model-Based Development of Embedded Systems  
16.04.2018 – 18.04.2018**

fortiss GmbH  
Guerickestr. 25  
80805 München

## **Organisationskomitee**

Prof. Dr. Michaela Huhn, Ostfalia Hochschule für angewandte Wissenschaften

Prof. Dr.-Ing. habil. Matthias Riebisch, Universität Hamburg

PD Dr. Hardi Hungar, Deutsches Zentrum für Luft- und Raumfahrt

Dr. Sebastian Voss fortiss GmbH

## **Programmkomitee**

Sibylle Froeschle, OFFIS & Universität Oldenburg

Michaela Huhn, Ostfalia Hochschule für angewandte Wissenschaften

Hardi Hungar, DLR

Matthias Riebisch, Universität Hamburg

Bernhard Rumpe, RWTH Aachen

Andy Schürr, TU Darmstadt

Andreas Vogelsang, TU Berlin

Sebastian Voss, fortiss GmbH

# Inhaltsverzeichnis

Zum fehlenden Architekturverständnis über Implementierungsmodelle multifunktionaler eingebetteter Systeme in der industriellen Praxis <i>Timo Kehrer, Andreas Vogelsang, Thomas Vogel and Heiko Dörr</i>	1
Qualification of Model-Based Development Tools - A Case Study <i>Mirko Conrad, Sophia Kohle and Hartmut Pohlheim</i>	7
Feature-based Recommendation for Product Configuration in the Software Product Lines <i>Yibo Wang, Lothar Hotz and Matthias Riebisch</i>	19
Feature-oriented Domain-specific Languages <i>Philipp Ulsamer, Tobias Fertig and Peter Braun</i>	31
Using PLC Programming Languages for Test-Case Specification of Hardware-in-the-loop Tests <i>David Thönnessen and Stefan Kowalewski</i>	41
Finding Inconsistencies in Design Models and Requirements by Applying the SMARTD Process <i>Stefan Kriebel, Evgeny Kusmenko, Bernhard Rumpe and Michael von Wenckstern</i>	51
Applying DSE for Solving the Deployment Problem in Industry 4.0 <i>Tarik Terzimehic, Sebastian Voss, Monika Wenger and Vincent Aravantinos</i>	61
Exploration of hardware topologies and complexity reduction <i>Johannes Eder</i>	71

Ein Mittel zur Wiederverwendung -- Komponentenbasierte Architekturen  
in der Automatisierungstechnik

*Constantin Wagner, Julian Grothoff and Ulrich Epple*

81

Integrating a Signaling Component Model into a Railway Simulation

*Daniel Schwencke*

87



Innerhalb der Gesellschaft für Informatik e.V. (GI) befasst sich eine große Anzahl von Fachgruppen explizit mit der Modellierung von Software- bzw. Informationssystemen. Der erst neu gegründete Querschnittsfachausschuss Modellierung der GI bietet den Mitgliedern dieser Fachgruppen der GI - wie auch nicht organisierten Wissenschaftlern und Praktikern - ein Forum, um gemeinsam aktuelle und zukünftige Themen der Modellierungsforschung zu erörtern und den gegenseitigen Erfahrungsaustausch zu stimulieren.



Das Institut für Software Engineering ist eine wissenschaftliche Einrichtung der Fakultät Informatik der Ostfalia, Hochschule für angewandte Wissenschaften. Die Forschungsschwerpunkte sind

- Entwicklung komplexer Systeme auf Basis von Java und Java EE
- Entwicklung webbasierter Oberflächen und mobiler Systeme
- Theoretische Grundlagen der Software-Entwicklung und formale Methoden
- Entwurfs- und Implementierungskonzepte für Software-Systeme
- Qualitätssicherung von Entwicklungsprozessen
- Modellgetriebene Software-Entwicklung

Die Anwendbarkeit der Lösungen wird immer wieder in industrienahen Projekten überprüft.



Schloss Dagstuhl wurde 1760 von dem damals regierenden Fürsten Graf Anton von Öttingen-Soetern-Hohenbaldern erbaut. 1989 erwarb das Saarland das Schloss zur Errichtung des Internationalen Begegnungs- und Forschungszentrums für Informatik. Das erste Seminar fand im August 1990 statt. Jährlich kommen ca. 2600 Wissenschaftler aus aller Welt zu 40-45 Seminaren und viele sonstigen Veranstaltungen.



fortiss ist das Forschungsinstitut des Freistaats Bayern für softwareintensive Systeme und Services mit Sitz in München. Das Institut beschäftigt derzeit rund 130 Mitarbeiter, die in Forschungs-, Entwicklungs- und Transferprojekten mit Universitäten und Technologie-Firmen in Bayern, Deutschland und Europa zusammenarbeiten.

Schwerpunkte sind die Erforschung modernster Methoden, Techniken und Werkzeuge der Softwareentwicklung, des Systems- & Service-Engineering und deren Anwendung auf verlässliche, sichere cyber-physische Systeme wie das Internet of Things (IoT). fortiss ist in der Rechtsform einer gemeinnützigen GmbH organisiert. Gesellschafter sind der Freistaat Bayern (als Mehrheitsgesellschafter) und die Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V..



Der Arbeitsbereich Softwareentwicklungs- und -konstruktionsmethoden SWK im Fachbereich Informatik der Universität Hamburg forscht auf dem Gebiet der Evolution von Softwaresystemen. Dazu gehören Arbeiten zu modellbasierter Softwareentwicklung, Softwarearchitekturen, Software-Reengineering sowie deren Einbettung in Entwicklungsprozesse. Das Ziel der Arbeiten sind ingenieurgemäße Vorgehensweisen und Methoden und deren Anwendbarkeit in der industriellen Praxis.



Das Deutsche Zentrum für Luft- und Raumfahrt (DLR) ist das Forschungszentrum der Bundesrepublik Deutschland für Luft- und Raumfahrt. Seine Forschungs- und Entwicklungsarbeiten in Luftfahrt, Raumfahrt, Energie, Verkehr, Digitalisierung und Sicherheit sind in nationale und internationale Kooperationen eingebunden.

Das Institut für Verkehrssystemtechnik ist eines der 36 Institute des DLR. Es betreibt Forschung und Entwicklung für Automobil- und Bahnsysteme und das Verkehrs- und Mobilitätsmanagement. Die Leitziele der Arbeit sind: Sicherheit, Effizienz, Nachhaltigkeit, Wirtschaftlichkeit und Qualität.

**Dagstuhl-Workshop MBEES:**  
**Modellbasierte Entwicklung eingebetteter Systeme**  
**(Model-Based Development of Embedded Systems)**

Die Bandbreite der Beiträge der diesjährigen Auflage der Dagstuhl-Workshopreihe „Modellbasierte Entwicklung eingebetteter Systeme“ zeigt, dass Modelle in vielen Bereichen eine wichtige Rolle spielen, von der Beherrschung der Komplexität von Entwurfsaufgaben über die Validierung und Verifikation von Systemen bis zur Generierung von Konfigurationen oder Software. Bei eingebetteten Systemen besteht eine besondere Herausforderung in der Modellierung über Plattform-, Technologie- und Hardware-Software-Grenzen hinweg. Die Spezifik solcher Systeme erfordert eine starke Anpassung von Modellierungssprachen an die Anforderungen der Domänen. Neue Entwicklungsrichtungen wie Industrie 4.0 und die Beherrschung der daraus resultierenden Datenmengen erfordern die Weiterentwicklung bisher bekannter Modellierungsansätze, beispielsweise bezüglich anwendungsorientierter Modelle mit domänenspezifischen Konzepten. Beiträge zur Weiterentwicklung des Stands der Forschung und zum Transfer in die industrielle Anwendung sind daher ein wesentliches Anliegen über die elf Ausgaben des Workshops hinweg. Der Fokus auf den – für eingebettete Systeme besonders wesentlichen – Bereich der Regelungs- und Steuerungstechnik stellt ein bedeutsames Ziel der industriellen modellbasierten Entwicklung dar.

Wie in den vorangehenden Jahren stellen die in diesem Tagungsband zusammengefassten Papiere sowohl gesicherte Ergebnisse, als auch Work-In-Progress, industrielle Erfahrungen und innovative Ideen aus diesem Bereich zusammen und erreichen damit eine interessante Mischung theoretischer Grundlagen und praxisbezogener Anwendung. Die breiter angelegten Diskussionsmöglichkeiten zielen auf eine Verstärkung des Austauschs zwischen solchen Ausrichtungen ab. Genau wie bei den vorhergehenden erfolgreich durchgeführten Workshops 2005 bis 2017 sind damit wesentliche Ziele dieses Workshops erreicht:

- Austausch über Probleme und existierende Ansätze zwischen den unterschiedlichen Disziplinen (insbesondere Elektro- und Informationstechnik, Maschinenwesen/Mechatronik, Automatisierungstechnik und Informatik)
- Austausch über relevante Probleme in der industriellen Anwendung und existierende Ansätze in der Forschung
- Verbindung zu nationalen und internationalen Aktivitäten (z.B. Initiative des IEEE zum Thema Model-Based Systems Engineering, GI-AK Modellbasierte Entwicklung eingebetteter Systeme, GI-FG Echtzeitprogrammierung, MDA Initiative der OMG)

Die Themengebiete, für die dieser Workshop gedacht ist, sind fachlich sehr gut abgedeckt. Die Beiträge adressieren verschiedenste Aspekte modellbasierter Entwicklung eingebetteter Softwaresysteme, unter anderem:

- Modelle in der architekturzentrierten Entwicklung und bei der Produktlinienentwicklung für Hardware-Software-Systeme und Software-intensive Systeme



- Domänenspezifische Ansätze zur Modellierung von Systemen
- Modellbasierte Validierung, Verifikation und Diagnose
- Modellierung zwecks Simulation von Systemverhalten zur Laufzeit
- Bewertung der Qualität von Modellen
- Funktionale Sicherheit und modellbasierte Entwicklung
- Evolution von Modellen
- Einbindung von Modellbasierter Entwicklung in Entwicklungsprozesse

Das Organisationskomitee ist der Meinung, dass mit den Teilnehmern aus Industrie, Werkzeugherstellern und der Wissenschaft die bereits seit 2005 erfolgte Community-Bildung erfolgreich weitergeführt wurde. Der nunmehr zwölfte MBEES Workshop belegt, dass eine solide Basis zur Weiterentwicklung des Themas modellbasierter Entwicklung eingebetteter Systeme existiert. Der hohe Anteil von deutschen Forschern und Entwicklern an den einschlägigen internationalen Konferenzreihen zu Modellierung und Cyberphysical Systems zeigt, dass die deutsche Zusammenarbeit in diesem Themenfeld Früchte getragen hat.

Die Durchführung eines erfolgreichen Workshops ist ohne vielfache Unterstützung nicht möglich. Wir danken daher den Mitarbeitern von Schloss Dagstuhl.

Schloss Dagstuhl im März 2018,

**Das Organisationskomitee:**

Michaela Huhn, Ostfalia Hochschule für angewandte Wissenschaften

Hardi Hungar, DLR

Matthias Riebisch, Uni Hamburg

Sebastian Voss, fortiss GmbH

Mit Unterstützung von

Tarik Terzimehic, fortiss GmbH

## Zum fehlenden Architekturverständnis über Implementierungsmodelle multifunktionaler eingebetteter Systeme in der industriellen Praxis

Timo Kehrer<sup>1</sup>, Andreas Vogelsang<sup>2</sup>, Thomas Vogel<sup>3</sup>, Heiko Dörr<sup>4</sup>

**Abstract:** Klassische eingebettete Systeme werden zunehmend zu autonomen und offenen Systemen, die auf ihre Umwelt reagieren und im Verbund mit anderen Systemen übergeordnete Ziele verfolgen. Modellbildung ist ein vielversprechender Ansatz, die Komplexität solcher multifunktionalen Systeme zu beherrschen. Auf der Ebene der Teilfunktionen hat sich Matlab-Simulink in vielen Unternehmen, bspw. in der Automobilbranche, als de-facto Standard für die modellbasierte Entwicklung von eingebetteten Systemen etabliert. Die Implementierungsmodelle der Teilfunktionen werden jedoch arbeitsteilig entwickelt und erst spät im Entwicklungsprozess zu einem Gesamtsystem integriert. Ein Gesamtsystem wird somit durch eine Menge lose gekoppelter Simulink-Modelle beschrieben, das Wissen über deren Kommunikationsbeziehungen ist lediglich implizit vorhanden und geht im Zuge der Softwareevolution zunehmend verloren. Bei der Integration kommt es daher häufig zu unerwünschtem und oftmals nicht vorhergesehenem Verhalten. Eine Analyse der entsprechenden Interaktionen einzelner Teilfunktionen ist derzeit lediglich auf Basis des generierten Quellcodes möglich. Dies steht jedoch in eklatantem Widerspruch zum Paradigma der modellbasierten Softwareentwicklung und führt zu hohen Integrationskosten. In diesem Papier analysieren wir diese Problematik des fehlenden „architektonischen“ Verständnis über eine Menge von Implementierungsmodellen und stellen unser Forschungsvorhaben zur Anhebung von funktionsübergreifenden Analysen auf die Modellebene vor.

### 1 Einführung

Klassische *eingebettete Systeme* entwickeln sich zunehmend zu autonomen und offenen Systemen, die auf ihre Umwelt reagieren und im Verbund mit anderen Systemen übergeordnete Ziele verfolgen. Dabei übernehmen einzelne Systeme immer mehr und vielfältigere Aufgaben und werden somit zu *multifunktionalen Systemen* [1]. Abb. 1 zeigt eine exemplarische Übersicht, wie die Entwicklung von multifunktionalen Systemen heute typischerweise aus Sicht eines Herstellers (*engl.* Original Equipment Manufacturer (OEM)) strukturiert ist. Ein komplexes *System* wird zunächst zerlegt in *Subsysteme*, welche die einzelnen Funktionen thematisch gruppieren. Im Fahrzeugbau wird oft von Domänen gesprochen

---

<sup>1</sup> Institut für Informatik, Humboldt-Universität zu Berlin. timo.kehrer@informatik.hu-berlin.de

<sup>2</sup> Fachgebiet IT-basierte Fahrzeuginnovationen, Technische Universität Berlin. andreas.vogelsang@tu-berlin.de

<sup>3</sup> Institut für Informatik, Humboldt-Universität zu Berlin. thomas.vogelinformatik.hu-berlin.de

<sup>4</sup> Model Engineering Solutions GmbH. doerr@model-engineers.com

(Karosserie, Antriebsstrang, Fahrerassistenz, Infotainment, etc.). Im Rahmen eines Teilsystems werden dann eine Reihe von *Kundenfunktionen* spezifiziert. Eine Kundenfunktion beschreibt ein gewünschtes Verhalten an der Systemgrenze, z.B. durch Kunden wahrnehmbar.

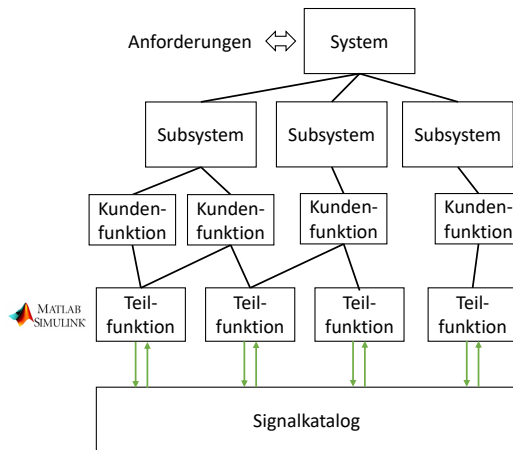


Abb. 1: Zerlegung eines multifunktionalen Systems.

eine partielle Sicht auf die Architektur der Teilfunktionen und deren Abhängigkeiten dar.

Modellbildung ist ein vielversprechender Ansatz, die Komplexität von multifunktionalen Systemen zu beherrschen. Auf der Ebene der Teilfunktionen setzen Unternehmen verstärkt auf *modellbasierte Entwicklung*, wobei sich Matlab Simulink<sup>5</sup> als de-facto Standard für die modellbasierte Entwicklung von eingebetteten Systemen etabliert hat. In Simulink wird das Verhalten einer Teilfunktion mit Hilfe eines datenflussorientierten Modells beschrieben. Aus diesem *Implementierungsmodell* kann später automatisch Code erzeugt werden, der auf einem Steuergerät ausgeführt wird und das gewünschte Verhalten erbringt. Die Implementierungsmodelle der Teilfunktionen eines multifunktionalen Systems werden jedoch arbeitsteilig und meist in unterschiedlichen Abteilungen eines Unternehmens entwickelt und erst sehr spät im Entwicklungsprozess zu einem Gesamtsystem integriert. Ein Gesamtsystem wird somit durch eine Menge lose gekoppelter und autark entwickelter Simulink-Modelle beschrieben, das Wissen über deren Kommunikationsbeziehungen ist lediglich implizit vorhanden. Bei der Integration kommt es daher häufig zu ungewünschtem, nicht vorhergesehenem Verhalten, so dass Integrations- und Testaktivitäten heute einen Großteil der Kosten bei der Entwicklung von eingebetteten Systemen verursachen [2].

In Abschnitt 2 analysieren wir die Problematik des fehlenden „architektonischen“ Verständnis über eine Menge von Implementierungsmodellen, welches wir insbesondere auf das Fehlen von frühen und funktionsübergreifenden Analysen auf Modellebene zurückführen. Daraus leiten wir in Abschnitt 3 unser Forschungsvorhaben zur konsequenten Anhebung aller

<sup>5</sup> <https://de.mathworks.com/products/simulink.html>

Der Begriff *Kunde* ist hier breit gefasst und enthält alle externen Systeme und Aktoren, die mit dem zu entwickelnden System interagieren. Kundenfunktionen werden weiter heruntergebrochen in einzelne *Teilfunktionen*, welche die Kundenfunktionen dann realisieren, indem sie Signale verarbeiten und daraus neue Signale erzeugen. Die im System verfügbaren Signale sind in einem *Signalkatalog* erfasst. Wenn zu den Teilfunktionen, die an der Realisierung einer Kundenfunktion beteiligt sind auch noch deren Abhängigkeiten dokumentiert werden, dann spricht man auch von einer *Wirkkette*. Wirkketten stellen also

integrativen Analysetätigkeiten auf die Modellebene ab. Eine Auswahl an alternativen Ansätzen beleuchten wir in Abschnitt 4 und resümieren unser Positionspapier in Abschnitt 5.

## **2 Problemanalyse**

Die Kosten für die Integration und den Test multifunktionaler eingebetteter Systeme sind heute trotz modellbasierter Entwicklungsansätze sehr hoch. Kernhypothese unseres Forschungsvorhabens ist, dass dies i.W. auf das fehlende architektonische Verständnis über eine Menge lose gekoppelter Implementierungsmodelle zurückzuführen ist.

Eine Integration der Teilfunktionen findet erst auf Code- bzw. Geräteebene und damit sehr spät im Entwicklungsprozess statt, so dass Abhängigkeiten zwischen diesen Teilfunktionen in den früheren Phasen der Entwicklung nicht berücksichtigt werden. Es gibt keine explizite Verwaltung und Kontrolle über die Abhängigkeiten. In Vorarbeiten haben wir festgestellt, dass vermeintlich unabhängige Kundenfunktionen im Automobilbereich auf der Ebene der Teilfunktionen hochgradig vernetzt sind [3], ca. die Hälfte aller Abhängigkeiten zwischen Kundenfunktionen waren den Entwicklern unbekannt. In einem untersuchten Projekt im Bereich Automotive Infotainment war das fehlende Verständnis über die Interaktionen der Teilfunktionen die Ursache für mehr als 40% aller Fehler [4]. Ferner werden die Schnittstellendefinitionen der Teilfunktionen in vielen Fällen in den Implementierungsmodellen nicht eingehalten und es existieren keine Ansätze, um diese Abweichungen kontinuierlich zu prüfen. Feilkas et al. [5] berichten über drei industrielle Fallstudien in denen bis zu 19% aller Abhängigkeiten in der Implementierung von der ursprünglich spezifizierten Architektur abweichen. Dabei war in vielen Fällen den Entwicklern nicht klar, ob es sich bei diesen Abweichungen um Fehler in der Spezifikation oder in der Implementierung handelt. Auch Wirkketten werden in der Praxis häufig nur zur Spezifikation von Kundenfunktionen genutzt. Eine Verifikation der implementierten Architektur gegenüber spezifizierten Wirkketten bleibt häufig aus. Hinzu kommt, dass die Spezifikation mit Hilfe von Wirkketten den Blick für Abhängigkeiten zwischen Wirkketten trübt. Abhängigkeiten zwischen Teilfunktionen unterschiedlicher Wirkketten sind in der implementierten Architektur vorhanden, spiegeln sich aber nicht in den spezifizierten Wirkketten wieder.

Zusammenfassend lässt sich sagen, dass der Einsatz von Modellbasierung zur frühen Fehlervermeidung und Reduzierung von Integrations- und Testkosten zwar vielversprechend ist, das volle Potential aber erst ausgeschöpft werden kann, wenn die erstellten Implementierungsmodelle auch modellübergreifend analysiert werden und kontinuierlich auf Abweichungen gegenüber abstrakteren Spezifikationen geprüft werden.

## **3 Forschungsvorhaben**

Übergeordnetes Ziel unseres Forschungsvorhabens ist die Entwicklung von strukturbasierten Verfahren zur Analyse der Architektur aller kommunizierenden Implementierungsmodelle,

mit denen sich mögliche Integrationsrisiken frühzeitig erkennen lassen. Dies umfasst die Identifikation und Erkennung von Architektur-Smells und Anti-Patterns, sowie die Verifikation gegenüber funktionsübergreifenden, strukturellen Spezifikationen wie bspw. Wirkketten. Grundlage aller modellbasierten Analysen bildet die tatsächlich vorliegende Architektur über einer Menge autark entwickelter Implementierungsmodelle, welche in einem ersten Reverse Engineering Schritt anhand der impliziten Abhängigkeiten zwischen den kommunizierenden Implementierungsmodellen gewonnen werden soll. Die Art und Granularität der extrahierten Architekturmodelle richten sich nach den jeweiligen Analysen. Da Simulink-Modelle oft als Basis für die Codegenerierung dienen, werden diese in der Regel sehr gut gepflegt und aktuell gehalten. Wir planen daher mit unseren Analysen auf einer Menge von bestehenden Simulink-Modellen aufzusetzen, die jeweils eine Teilfunktion beschreiben. Es ist geplant, die Analysen in das Qualitätssicherungswerkzeug MES M-XRAY<sup>6</sup> einzubetten. Die entwickelten Verfahren sind jedoch auch auf andere blockorientierte Modelltypen wie bspw. Komponenten- und Konnektormodelle [6] übertragbar.

Ein Beispielszenario für die Verifikation einer Implementierungsarchitektur ist deren Konformanzprüfung gegenüber der spezifizierten Soll-Architektur, welche sich bspw. aus den Schnittstellendefinitionen der Software-Komponenten zur Realisierung von Teilfunktionen ableiten lässt. In Matlab-Simulink werden zur Schnittstellenspezifikation einer Teilfunktion oftmals die Ein- und Ausgaben (d.h. die Input- und Outputsignale) des Top-Level Subsystems des entsprechenden Simulink-Modells verwendet. Eine Soll-Architektur resultiert somit implizit aus den dadurch festgelegten Abhängigkeiten zwischen den verschiedenen Simulink-Modellen. Im Beispiel in Abb. 2 enthält die Soll-Architektur eine Abhängigkeit zwischen Teilfunktion 1 und Teilfunktion 2 über das Signal *Sig\_B*. Zur Extraktion des tatsächlichen Architekturmodells steigen wir tiefer in den hierarchischen Simulink-Modellen hinab und interpretieren diese als Implementierung einer Teilfunktion. Wir analysieren, in wie weit hier zusätzliche Abhängigkeiten vorhanden sind, die auf dem Top-Level Subsystem nicht berücksichtigt wurden. Diese impliziten Abhängigkeiten stellen potentielle Verletzungen der Ist-Architektur gegenüber der Soll-Architektur dar (im Beispiel die implizite Abhängigkeit über das Signal *Sig\_X*).

Eine offene Forschungsfrage ist, inwiefern Integrationsfehler tatsächlich mit architektonischen Schwachstellen wie der oben skizzierten Abweichung von Soll- und Ist-Architekturen korrelieren. Ähnliche Fragen ergeben sich für Architektur-Smells, Anti-Patterns sowie für negative Ergebnisse bei der Verifikation von Wirkketten. Dies soll in empirischen Studien in Kooperation mit unserem assoziierten Partner aus dem Bereich Automotive untersucht werden. Auf unseren Forschungsergebnissen aufbauend wollen wir ferner Verfahren zur Bewertung von Integrationsrisiken und schließlich auch zu deren geeigneter Behandlung entwickeln.

---

<sup>6</sup> <https://www.model-engineers.com/de/m-xray.html>

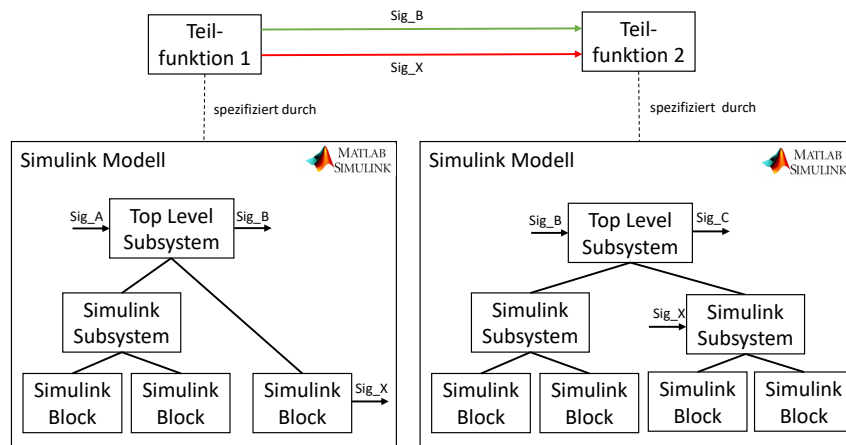


Abb. 2: Extraktion einer Soll- und Ist-Architektur durch Analyse von Simulink-Modellen.

#### 4 Alternative Ansätze und verwandte Arbeiten

In der Praxis sind Verfahren zum Reverse Engineering von Architekturen aus Implementierungsmodellen sowie darauf aufbauende, modellbasierte Analysen nicht etabliert. Vielmehr werden aktuell Ansätze zum Forward-Engineering von Software-Architekturen unter Einsatz klassischer UML-basierter Notationen erprobt. Jedoch haben diese Modelle häufig nur einen Dokumentationscharakter und dienen nicht als technische Grundlage für die Implementierung. Weiterhin werden spezielle Architekturdatenbanken eingesetzt, aus denen Architekturspezifikationen für einzelne Simulink-Modelle gewonnen werden können. Alle Ansätze haben gemein, dass ein nachträglicher Architekturabgleich nicht konstruktiv unterstützt wird. Analysewerkzeuge für Implementierungsmodelle sind zwar bereits verfügbar, unterstützen jedoch lediglich die Analyse einzelner, autonomer Modelle. Ein Beispiel für ein ausgereiftes Analysewerkzeug für einzelne Simulink-Modelle ist MES M-XRAY, modellübergreifende Analysen werden bislang jedoch nicht unterstützt.

Aus Forschungsperspektive mit unserem Ansatz vergleichbar ist die in [7] skizzierte Idee des „kontinuierlichen Reverse Engineering“ und der stetigen Konformanzprüfung der extrahierten Modelle zu dem entsprechenden Quellcode. Extrahiert werden hier jedoch klassische, aus der Objektorientierung bekannte Modelltypen. In [8] werden Abhängigkeitsgraphen aus Simulink-Modellen erzeugt, allerdings auf Basis eines einzelnen Simulink-Modells und mit dem Ziel der automatisierten Testfallgenerierung. Modelle über Soll- und Ist-Architekturen lassen sich als zwar als co-evolvierende Modelle auffassen, existierende Analysen von co-evolvierenden Modellen konzentrieren sich meist auf Änderungen in klassischen Multi-View-Modellen [9]. Ansätze zur Verifikation von Implementierungsmodellen gegenüber abstrakten Spezifikationen fokussieren sich wie die in [6] vorgestellte Methode auf die Formalisierung und Verifikation von Anforderungsspezifikationen, und wurden bislang ebenfalls nur auf einzelne Implementierungsmodelle angewendet.

## 5 Resümee

Trotz modellbasierter Methoden ist die Entwicklung multifunktionaler eingebetteter Systeme eine überaus aktuelle Herausforderung. Ein Gesamtsystem wird in der Praxis meist durch eine Menge lose gekoppelter Simulink-Modelle beschrieben, ein architektonisches Verständnis über deren Kommunikationsbeziehungen ist lediglich implizit vorhanden, was zu hohen Integrationskosten führt. Ziel unseres Forschungsvorhabens ist die Entwicklung von strukturbasierten Verfahren zur Analyse der Architektur aller kommunizierenden Implementierungsmodelle, mit denen sich mögliche Integrationsrisiken frühzeitig erkennen, bewerten und letzten Endes auch behandeln lassen.

## Literatur

- [1] Manfred Broy. „Multifunctional software systems: Structured modeling and specification of functional requirements“. In: *Science of Computer Programming* 75.12 (2010).
- [2] Claudiu Farcas u. a. „Addressing the Integration Challenge for Avionics and Automotive Systems—From Components to Rich Services“. In: *Proc. of the IEEE* 98.4 (2010), S. 562–583.
- [3] Andreas Vogelsang und Steffen Fuhrmann. „Why Feature Dependencies Challenge the Requirements Engineering of Automotive Systems: An Empirical Study“. In: *Proc. IEEE International Requirements Engineering Conference (RE’13)*. 2013.
- [4] Sebastian Benz. „Generating Tests for Feature Interaction“. Diss. Technische Universität München, 2010.
- [5] Martin Feilkas, Daniel Ratiu und Elmar Jurgens. „The loss of architectural knowledge during system evolution: An industrial case study“. In: *International Conference on Program Comprehension (ICPC)*. IEEE. 2009, S. 188–197.
- [6] Vincent Bertram u. a. „Component and Connector Views in Practice: An Experience Report“. In: *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE. 2017, S. 167–177.
- [7] Gerardo Canfora, Massimiliano Di Penta und Luigi Cerulo. „Achievements and Challenges in Software Reverse Engineering“. In: *Commun. ACM* 54.4 (2011).
- [8] Adepu Sridhar, D. Srinivasulu und Durga P. Mohapatra. „Model-based test-case generation for Simulink/Stateflow using dependency graph approach“. In: *IEEE International Advance Computing Conference (IACC)*. 2013.
- [9] Sinem Getir, Michaela Rindt und Timo Kehrler. „A Generic Framework for Analyzing Model Co-Evolution.“ In: *ME @ MoDELS*. 2014.

# Qualification of Model-Based Development Tools

## A Case Study

Mirko Conrad<sup>1</sup>  
samoconsult GmbH  
mirko.conrad@samoconsult.de

Sophia Kohle, Hartmut Pohlheim  
Model Engineering Solutions GmbH  
sophia.kohle@model-engineers.com  
pohlheim@model-engineers.com

**Abstract:** Modern functional safety standards typically provide objectives or requirements on how to gain confidence in development tools used for electric and/or electronic systems. The corresponding approaches are referred to as tool qualification, tool validation, or tool certification.

To gain confidence in the tools used to develop automotive E/E systems, the ISO 26262 standard outlines a two-step approach consisting of tool classification potentially followed by tool qualification.

Although some research about ISO 26262 tool classification has been published, information about the tool qualification part is rather limited. This paper intends to reduce this gap by reporting on the qualification of MXAM, a static analysis tool for Simulink/TargetLink models.


## 1 Introduction

Software tools are widely used in multiple domains to assist in developing or verifying electric and/or electronic systems (E/E systems). In E/E system development, such tools can assist with analysis and potentially improve system safety by automating the activities performed and by predictably performing tasks that may be prone to human error. On the contrary, an error in a tool may have a negative impact on safety if the tool inadequately performs its intended functions (cf. [DO 330]).

To reduce the potential risks associated with tool usage and to ensure the integrity of the tool functionality, recent (functional) safety standards call for dedicated activities to gain confidence in the tools used in the development of E/E systems. Depending on the domain or standard, the corresponding approaches are referred to as tool validation, tool qualification, or tool certification.

For the development of automotive E/E systems, the pertinent functional safety standard is ISO 26262 [ISO 26262]. Part 8 of this standard calls for a two-step process to gain

---

<sup>1</sup>  <https://orcid.org/0000-0003-3221-6503>



confidence in software tools. It starts with (I) tool classification to determine the required level of confidence in each software tool. Depending on the outcome of the first step, (II) a subsequent tool qualification process to establish the required confidence might be necessary.

When ISO 26262 came into effect in 2011, tool classification and qualification changed from a niche topic to a mainstream requirement for automotive OEMs, their suppliers, and software tool vendors. In the meantime, practitioners gained quite a bit of experience with tool classification (see e.g., [Mai09], [CMR10], [Con10], [KKG10], [CF11], [HK11], [Spä14], and [CF15]). Tool support for the ISO 26262 tool classification activities is available by means of dedicated Excel templates [Con14], [Con16], or the Tool Chain Analyzer [SWP+12].

Actionable advice on tool qualification is – despite some general observations regarding this topic (see e.g. [SWP+12]), work-share considerations (see e.g., [CMR10], [CSM11]), or tool type-specific advice mostly for code generators and compilers (see e.g., [SS07], [SCD07], [Glö08], [SML08], [KKG10]) – still lacking.

By outlining the approach taken to qualify the MES Model Examiner® (MXAM), a static analysis tool/guideline checker for executable models used in the development of automotive E/E systems, the authors aim to widen the accessible body of information on this topic.

## 2 Gaining Confidence in the Use of Software Tools – The ISO 26262 Approach

The two-step approach to gaining confidence in the use of software tools required as per ISO 26262 is illustrated in Figure 1

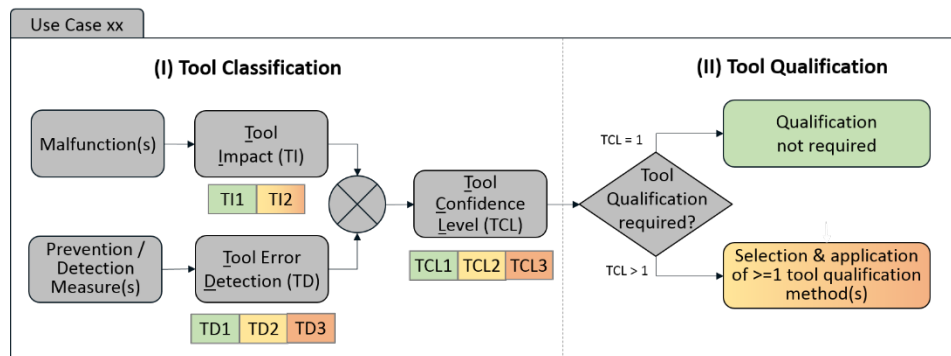


Figure 1: Two-step approach to gaining confidence in the software tools used.

## 2.1 Tool Classification

Tool classification is based on the actual/intended usage of the tool. Therefore, the tool usage needs to be documented by means of tool use cases. Each of the use cases is subjected to further analysis.

First, potential malfunctions of the tool that could occur in the context of the use case at hand need to be identified and documented. For each malfunction, it needs to be determined whether the tool could introduce errors into the E/E system under development or fail to detect such errors. If it can be argued that there is no such possibility, the malfunction has a tool impact of 1 (TI1), otherwise the tool impact is 2 (TI2). If all malfunctions in the context of a given use case are rated TI1, the entire use case can be considered TI1.

Second, the measures applied to prevent or detect these malfunctions or their resulting erroneous output need to be documented and the confidence in these measures needs to be rated. Depending on whether there is high, medium, or even low confidence, the tool error detection is 1 (TD1), 2 (TD2), or 3 (TD3) respectively. In addition to the error detection of the individual measures, the tool error detection of a combination of measures might be rated [Con16]<sup>2</sup>.

Third, a tool confidence level (TCL) is assigned to each combination of a tool use case and a corresponding tool malfunction. Given a tool impact class (i.e., TI1 or TI2) and a tool error detection class (i.e., TD1, TD2, or TD3), the corresponding TCL can be derived according to Table 1 [ISO26262].

Table 1 Determination of the Tool Confidence Level (TCL).

Tool Error Detection Tool Impact	TD1	TD2	TD3
TI1	TCL1	TCL1	TCL1
TI2	TCL1	TCL2	TCL3

Those combinations of use cases and malfunctions rated as TCL1 do not require further action. For all other combinations, i.e. those rated TCL2 or TCL3, the tool qualification process needs to be initiated. The tool classification step is documented in a tool criteria evaluation report (a.k.a. tool classification report).

## 2.2 Tool Qualification

As per ISO 26262, tool qualification shall be carried out using a suitable combination of the following four tool qualification methods:

---

<sup>2</sup> A combination of orthogonal detection measures might have better error detection capability than each of the individual measures. Rating such measure combinations requires engineering judgement and cannot be automated.

- a) Increased confidence from use.
- b) Evaluation of the tool development process.
- c) Validation of the software tool.
- d) Development in compliance with a safety standard.

The selection of appropriate tool qualification methods depends on the TCL and on the Automotive Safety Integrity Level (ASIL) of the E/E system to be developed.

However, the practical significance of tool qualification methods a)<sup>3</sup> and d) is rather limited<sup>4</sup>. The vast majority of all tool qualification approaches known to the authors use method b) or c) or a combination thereof.

If the method b) ‘Evaluation of the tool development process’ is applied to qualify a software tool, its tool development process shall comply with an appropriate standard. The tool development process shall be assessed based on an appropriate national or international standard (e.g., Automotive SPICE, CMMI, or ISO 15504) and the proper application of the assessed development process shall be demonstrated.

If the method c) ‘Validation of the software tool’ is utilized, the validation of the software tool shall meet three criteria:

- a) It shall be demonstrated that the software tool complies with its specified requirements, e.g., by using validation tests or reviews designed to evaluate functional and non-functional quality aspects of the tool.
- b) If malfunctions occur during the validation, these malfunctions and the resulting erroneous outputs shall be analyzed. Also, information on their possible consequences and measures to avoid or detect them shall be provided.
- c) The reaction of the software tool to anomalous operating conditions (e.g., foreseeable misuse, incomplete input data, and incompatible combinations of configuration settings) shall be examined.

The tool qualification step is documented in a tool qualification report.

### 3 Case Study: MXAM Qualification Kit

When utilizing the model-based development paradigm to develop automotive E/E systems, software units can be designed by means of executable Simulink or TargetLink models. Prior to code generation, the unit design needs to be statically analyzed to verify compliance with the applicable modeling guidelines. Modeling guideline checking can be carried out using MXAM.

---

<sup>3</sup> Although a) sounds promising, it is rarely applicable due to frequent changes/updates to the tools being used.

<sup>4</sup> Given the number and variety of development tools for E/E systems, proof of formal correctness could replace or extend the listed qualification methods for only a small subset of these tools.

### 3.1 The MES Model Examiner® DRIVE

Using data and control flow analysis, MXAM checks Simulink or TargetLink models for aspects of functional safety, such as strong data typing, appropriate scaling and ranges of data, naming conventions, and layout.

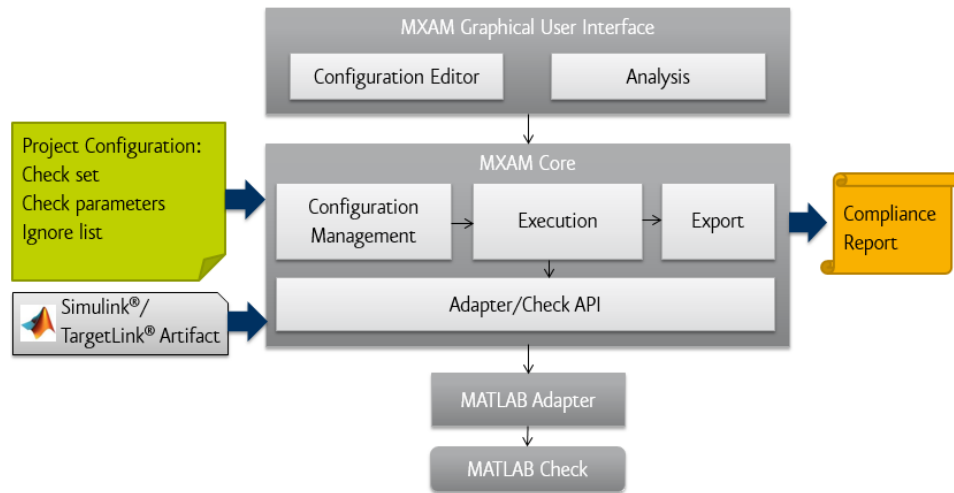


Figure 2: Architectural components of MXAM.

MXAM consists of a framework providing general functionalities such as GUI, management of projects and guidelines and checks, repair actions, and reporting. Further, dependent on the platform, guideline checks are provided or can be implemented by the tool user (cf. Figure 2). These checks comprise a description and an implementation.

### 3.2 Tool Usage, Malfunctions, and Prevention/Detection Measures

The tool criteria evaluation report for MXAM that ships with the MXAM ISO 26262 Qualification Kit (cf. [Ko18]) analyzes four main uses cases, including:

- [UC\_MXAM01] Design Model – Modeling Guidelines Compliance Check.

Two potential malfunctions associated with the above use case will be used to illustrate the MXAM qualification approach in this paper:

- a) [E\_MXAM02] Modeling Guidelines Compliance Check: Violation not found.
- b) [E\_MXAM07] Modeling Guidelines Compliance Check: Usage of incorrect check parameters.

Certain malfunctions are overarching and thus check-agnostic. Corresponding qualification activities are check-agnostic as well. Other malfunctions have check-specific aspects and thus require check-specific qualification activities.

Table 2, Table 3, and Table 4 illustrate how this part of tool classification is documented in the qualification kit.

**[E\_MXAM02]:** To detect [E\_MXAM02] in the context of [UC\_MXAM01], subsequent dynamic testing of the model under analysis can be applied ([M\_MXAM01]). However, the probability of detecting such a tool error with [M\_MXAM01] is considered to be medium and therefore the detection level is limited to TD2.

Additional subsequent guideline checking of the implementation model [M\_MXAM03] is another measure to counter [E\_MXAM02]. Again, this method can detect certain but not all instances of [E\_MXAM02] and is therefore rated as TD2 as well.

Table 2: Tool Classification Example I ([UC\_MXAM01] x [E\_MXAM02])

Potential Malfunction	TI	Prevention/Detection Measure	TD	Justification for TD
[E_MXAM02] Modeling Guidelines Compliance Check: Violation not found	TI2	[M_MXAM01] Subsequent dynamic testing of the model under analysis	TD2	Guideline violation does not always result in incorrect or unintended model functionality. Actual model errors can be revealed by dynamic testing ('medium' likelihood of error detection).
		[M_MXAM03] Subsequent modeling guidelines checking of a downstream model, e.g. the implementation model	TD2	Subsequent static checking of the implementation model provides another means to identify guideline violations. However, if a check fails to detect a violation in the design model, it may fail to identify the same issue in the implementation model as well ('medium' likelihood of error detection).
		MEASURE COMBINATION	TD2	TCL 2 ▷ Addressed by MXAM ISO 26262 Qualification Kit.

Even if both measures are combined, a tool error detection better than TD2 could not be achieved. As a result, the tool confidence level for [E\_MXAM02] in the context of [UC\_MXAM01] is TCL2 and the combination [UC\_MXAM01] x [E\_MXAM02] needs to be subjected to tool qualification.

Without a tool-vendor-provided qualification kit, the tool user would need to qualify [UC\_MXAM01] x [E\_MXAM02], e.g., by validating the underlying MXAM functionality with a test suite. In the case of MXAM, this burden on the user is eased by the MXAM ISO 26262 Qualification Kit<sup>5</sup>.

**[E\_MXAM07]:** The second potential malfunction addresses that the check implementation could receive parameters from the MXAM framework that differ from the ones specified by the user (see Table 3). The default check parameters can be overwritten on several levels by the user (e.g., project, document, guideline level). The error that is addressed here might occur due to a defect in the overwrite algorithm that is common to all checks

<sup>5</sup> Please note the lower right cell in Table 2 indicating that the tool user can leverage the qualification kit.

(i.e. check-agnostic). Malfunctions in the check-specific handling of check parameters are covered by [E\_MXAM02].

Table 3: Tool Classification Example II ([UC\_MXAM01] x [E\_MXAM07] – Initial Classification)

Potential Malfunction	TI	Error Prevention/ Detection Measure	TD	Justification for TD
[E_MXAM07] Modeling Guidelines Compliance Check: Usage of incorrect check parameters	TI2	[M_MXAM13] Review of parameter specification details	TD1	Review of the parameter specification details ensures the correctness of the guideline check parameters to be used.
		[M_MXAM11] Check for Error Messages	TD2	Checking logs and compliance reports for error messages helps to detect anomalies such as an incorrect parameter definition.
		MEASURE COMBINATION	TD1	TCL 1

Malfunction [E\_MXAM07] could be mitigated by reviewing the parameter specification details provided by MXAM ([M\_MXAM13]) as these document the actual check parameters used during check execution. Thus, sufficient error prevention is possible (TCL1, cf. Table 3), but the necessary review activities would be very time consuming as they need to be performed for each individual overwritten parameter.

To reduce the amount of necessary prevention/detection activities for the tool user, the MXAM ISO 26262 Qualification Kit covers [E\_MXAM07] as well, meaning that [M\_MXAM13] does not need to be carried out any more. Hence, the effort required by the user to review every check parameter (currently more than 250 in the standard edition) for each project has been replaced by a qualification activity to be carried out by the tool vendor. This is reflected in the revised classification provided in Table 4.

Table 4: Tool Classification Example II ([UC\_MXAM01] x [E\_MXAM07] – Revised Classification)

Potential Malfunction	TI	Error Prevention/ Detection Measure	TD	Justification for TD
[E_MXAM07] Modeling Guidelines Compliance Check: Usage of incorrect check parameters	TI2	[M_MXAM11] Check for Error Messages	TD2	Checking logs and compliance reports for error messages helps to detect anomalies such as an incorrect parameter definition.
		MEASURE COMBINATION	TD2	TCL 2 ▷ Addressed by MXAM ISO 26262 Qualification Kit

### 3.3 Tool Qualification

[E\_MXAM02]: In chapter 3.2, the need to qualify [UC\_MXAM01] x [E\_MXAM02] is outlined. To qualify this use case-error combination, qualification method c) ‘Validation of the software tool’ was chosen. A combination of check-specific test cases and reviews was defined to verify that the corresponding checks identify all guideline violations.

For each check, specific test cases for regression, invariant, and smoke tests were designed and implemented as Simulink models. These tests are executed continuously and in the release build pipeline as Jenkins jobs. The check-specific test cases use a common test framework and share a common structure. Each test case consists of eight sub-tests illustrated in Table 5. The resulting JUnit test report (cf. Figure 3) is provided as qualification evidence in the MXAM ISO 26262 Qualification Kit.

Table 5: Common structure of the tests to qualify [UC\_MXAM01] x [E\_MXAM02]

a)	elementResultsQualifierTest: Qualifiers (type of elements in the check results, such as blocks, lines, etc.) of the elements occurring in the model check results are compared with pre-defined expected results.
b)	elementResultNameValuePairTest: Name value pairs (check-specific information in addition to the standardized attributes and values, such as name, path, message, etc.) occurring in the model check results are compared with pre-defined expected results.
c)	elementResultsPathTest: Path information occurring in the model check results is compared with pre-defined expected results.
d)	elementResultsResultTest: Verdict of the model check (failed, passed, warning) is compared with pre-defined expected results.
e)	elementResultResultMessageTest: Messages occurring in the model check results are compared with pre-defined expected results.
f)	elementResultsSizeCheckTest: Number of the elements results occurring in the model check results are compared with pre-defined expected results.
g)	invariantTest: Results occurring in the model check are compared in two separate analyses.
h)	elementResultsNotWantedTest: Result occurring in the model check is not aborted.

Class MATLAB.DynamicTestSuite

Name	Tests	Errors	Failures	Skipped	Time(s)	Time Stamp	Host
DynamicTestSuite	327	0	0	0	2011.217	2017-10-12T10:11:18	JenkinsChecks

Tests

Name	Status
MATLAB.MATLAB_MAAAB.mcheck_db_0151.v01.tests.InvariantzTest_db_0151.invariantzTest	Success
MATLAB.MATLAB_MAAAB.mcheck_db_0151.v01.tests.InvariantzTest_db_0151.elementResultsNotWantedTest	Success
MATLAB.MATLAB_MAAAB.mcheck_db_0151.v01.tests.ResultTest_db_0151_R2015b.elementResultsQualifierTest	Success
MATLAB.MATLAB_MAAAB.mcheck_db_0151.v01.tests.ResultTest_db_0151_R2015b.elementResultNameValuePairTest	Success
MATLAB.MATLAB_MAAAB.mcheck_db_0151.v01.tests.ResultTest_db_0151_R2015b.elementResultsPathTest	Success
MATLAB.MATLAB_MAAAB.mcheck_db_0151.v01.tests.ResultTest_db_0151_R2015b.elementResultsResultTest	Success
MATLAB.MATLAB_MAAAB.mcheck_db_0151.v01.tests.ResultTest_db_0151_R2015b.elementResultResultMessageTest	Success
MATLAB.MATLAB_MAAAB.mcheck_db_0151.v01.tests.ResultTest_db_0151_R2015b.elementResultsSizeCheckTest	Success

Figure 3: JUnit test report to document the qualification of [UC\_MXAM01] x [E\_MXAM02] (Excerpt)

**[E\_MXAM07]:** For the second malfunction [E\_MXAM07], designated, guideline-agnostic test cases have been developed to assess the correct application of parameters. For a baseline version of MXAM, the correctness of the resulting MXAM analysis report has been established manually. For subsequent versions of the tool, regression tests are conducted (cf. Figure 4). With the help of an MXAM report differ, the MXAM analysis reports resulting from a regression test are compared against the analysis reports from the baseline version.

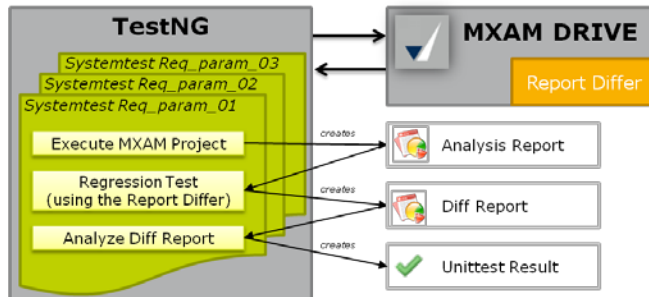


Figure 4: Set-up of system tests and resulting work products to document the qualification of [UC\_MXAM01] x [E\_MXAM07].

In addition, 10 system tests covering different aspects of the usage of global, shared, and check parameters were defined as well to qualify [UC\_MXAM01] x [E\_MXAM07] (cf. Figure 5). These system tests are executed continuously and in the release build pipeline as Jenkins jobs. The resulting Allure report (cf. Figure 5) is provided as qualification evidence in the MXAM ISO 26262 Qualification Kit.

1	Req param 01 - check param default	11s 288ms	PASSED
2	Req param 02 - overwrite check param from guideline	11s 391ms	PASSED
3	Req param 03 - overwrite check param from document	11s 112ms	PASSED
4	Req param 04 - overwrite check param from project	11s 095ms	PASSED
5	Req param 06 - default shared param	8s 092ms	PASSED
6	Req param 07 - overwrite default shared param from guideline	8s 115ms	PASSED
7	Req param 08 - overwrite default shared param from document	8s 285ms	PASSED
8	Req param 09 - overwrite default shared param from project	8s 068ms	PASSED
9	Req param 11 - default global param	8s 003ms	PASSED
10	Req param 12 - global adapter param access	11s 007ms	PASSED

Figure 5: Allure report to document the qualification of [UC\_MXAM01] x [E\_MXAM07].

The guideline-specific tests and the system tests cover the two potential malfunctions used as an example.

Overall, the MXAM ISO 26262 Qualification Kit addresses 17 combinations of use cases and malfunctions that were rated as TCL2 or TCL3 in the tool classification.



### 3.4 User Activities and Savings

Using a tool-vendor-provided qualification kit can significantly streamline the user's tool classification and qualification activities. However, it does not exempt the tool user from having to perform any further activities.

Using the example of the MES Model Examiner®, Figure 6 contrasts the classification and qualification activities to be performed with (right-hand side) and without (left-hand side) the MXAM ISO 26262 Qualification Kit and designates the activities that are being streamlined by using such a kit.

As an example, instead of conducting the entire tool qualification, the user would only need to review and confirm the validity of the documentation of the predetermined tool qualification provided as part of the qualification kit. Regardless of who carries out tool qualification, the user needs to ensure that actual tool usage complies with the constraints and assumptions of tool classification and qualification (e.g., ensuring that the assumed prevention and detection measures are conducted).

If user-created checks are being used, the check-specific qualification activities need to be extended to these checks. As an example, the vendor-provided tests and reviews to qualify ([UC\_MXAM01] x [E\_MXAM02]) need to be augmented by similar tests and reviews for the user-created checks.

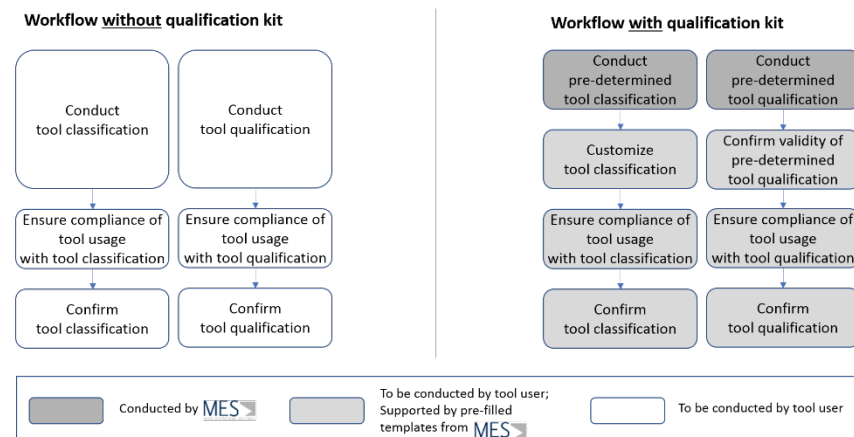


Figure 6: Workshare of tool classification and qualification activities.

According to [CF14], the mean effort for tool qualification can be estimated at 80 hours. If a qualification kit is used, the tool user is relieved of most of this effort if the tool usage is sufficiently aligned with the vendor's usage guidance and constraints. In this case, the user's effort is reduced to reviewing and confirming or adapting the pre-determined tool qualification.

Potential savings for the tool user are much higher if the vendor-provided tool qualification removes the need for the user to conduct certain recurring review activities (e.g. the elimination of the recurring review of the parameter specification details [M\_MXAM13]).

The MXAM ISO 26262 Qualification Kit provides the user with designated and flexible Word and Excel templates for the required ISO 26262 classification and qualification work products. User feedback indicates, that such templates are preferred to other available solutions as they are easy to comprehend and adaptation to established, user-specific workflows is straightforward.

## 4 Summary and Conclusion

Gaining confidence in the tools used to develop E/E systems via tool classification and tool qualification is a base requirement for today's development projects. Modern functional safety standards, such as ISO 26262, call for these activities. The responsibility for conducting these activities rests with the tool user. However, support from the tool vendor can streamline the user activities.

Vendor-provided tool classification already reduces the effort for the user when evaluating the confidence in the tool for the intended use cases. Instead of creating the entire classification, the user only needs to review and confirm or adapt the predetermined tool classification.

However, if the tool classification results in a TCL2 or TCL3 (medium or low tool confidence) for some use cases, users need to apply additional prevention/detection measures or must qualify the tool with respect to these use cases. For some use cases, sufficient tool confidence (i.e., TCL1) can be reached by conducting additional review activities. For other use cases, such additional activities might be infeasible or prohibitive due to high complexity or effort. Tool qualification is the only option here.

Using the example of the MES Model Examiner®, this paper provides insight into the structure and the qualification approach of an actual tool qualification kit for a popular model-based analysis tool. The tool qualification kit utilizes the qualification method 'validation of the software tool'; validation is conducted by a combination of different tests and reviews. Utilizing such a tool-vendor-provided qualification kit significantly reduces the burden on the tool user by minimizing the actual qualification effort and also by reducing the amount of prevention/detection measures they need to carry out.

Further research interests of the authors include a taxonomy of tool classification approaches, the provision of tool classification patterns, and further improvement of the utilized classification and qualification templates.

## References

- [CF11] M. Conrad, I. Fey "ISO 26262 - Exemplary tool classification of Model-Based Design tools". Softwaretechnik-Trends 31 (2011) 3  
[http://pi.informatik.uni-siegen.de/stt/31\\_3/01\\_Fachgruppenberichte/ada/5-CF11-11\\_20110803.pdf](http://pi.informatik.uni-siegen.de/stt/31_3/01_Fachgruppenberichte/ada/5-CF11-11_20110803.pdf)

- [CF14] M. Conrad, I. Fey "Effort and Efficacy of Tool Classification and Qualification". Proc. MBEES X (2014)
- [CF15] M. Conrad, I. Fey "Tool Classification & Qualification According to ISO 26262". 4<sup>th</sup> Int. CTI Conf. ISO 26262 (2015)
- [CMR10] M. Conrad, P. Munier, F. Rauch "Qualifying Software Tools According to ISO 26262". Proc. MBEES VI (2010)
- [Con10] M. Conrad "Software Tool Qualification According to ISO 26262 - An Experience Report". Supplementary Proc. of 21. Int. Symposium on Software Reliability Engineering (ISSRE 2010), pp. 460-466 (2010)
- [Con14] M. Conrad "Tool Classification and Qualification in Practice" 4th VDA Automotive SYS Conference (2014)
- [Con16] M. Conrad, I. Fey "Tool Classification Made Easy - The Making of an ISO 26262 Tool Classification Kit". MES User Forum 2016 (2016)
- [CSM11] M. Conrad, G. Sandmann, P. Munier "Software Tool Qualification According to ISO 26262". SAE 2011 World Congress, Detroit, MI, US, April 2011 (2011)  
doi:10.4271/2011-01-1005
- [DO330] DO-330:2011. "Software Tool Qualification Considerations". RTCA (2011)
- [Glö08] T. Glötzner "IEC 61508 Certification of a Code Generator". ICSS2008 (2008)
- [HK+11] R. Hamann, S. Kriso, K. Williams, J. Klarmann, J. Sauler "ISO 26262 Release Just Ahead: Remaining problems and Proposals for Solutions". SAE 2011 World Congress, Detroit, MI, US, April 2011 (2011)
- [ISO26262] ISO 26262:2011. "Road vehicles - Functional safety". Int. Org. for Standardization (2011-2012)
- [KKG10] J. Klarmann, S. Kriso, M. Gebhardt "Qualification of development tools as per ISO 26262". REAL TIMES, 1/2010, pp. 28-20 (2010)
- [Ko18] S. Kohle "MES Model Examiner® Drive ISO 26262 Qualification Kit". Model Engineering Solutions GmbH, <https://www.model-engineers.com/mxam.html> (2018)
- [Mai09] M. Maihöfer "Umgang mit Entwicklungswerkzeugen in Software-Entwicklungsprozessen der Automobilindustrie - ISO DIS 26262, Band 8, Kapitel 11: Inhalt, Bewertung, Auswirkung und Umsetzung (in German). EUROFORUM Konferenz 'Funktionale Sicherheit nach ISO/DIS 26262', Stuttgart, Germany, Sept. 2009 (2009)
- [SCD+07] I. Stürmer, M. Conrad, H. Dörr, P. Pepper "Systematic Testing of Model-Based Code Generators". IEEE Transactions on Software Engineering, 33 (2007) 9  
doi:10.1109/TSE.2007.70708
- [SML08] S. Schneider, P. Mai, T. Lovric "The Validation Suite Approach to Safety Qualification of Tools" Automotive - Safety & Security 2008 (2008)
- [Spä14] A. Späthe "Den Nagel auf den Kopf zu treffen, reicht nicht - Carmeq führt die ISO-26262-konforme Klassifizierung von Software-Werkzeugen ein". meilenstein 2/2013, pp. 14-15 (2013)  
<http://www.carmeq.com/downloads/Meilenstein-2-2013.pdf>
- [SS07] S. Schneider, O. Slotosch "A validation suite for Model-based Development Tools" 10. Int. Conf. on Quality Engineering in Software Technology CONQUEST (2007)
- [SWP+12] O. Slotosch, M. Wildmoser, J. Philipps, R. Jeschull, R. Zalman "ISO 26262 - Tool Chain Analysis Reduces Tool Qualification Costs". Automotive - Safety & Security 2012 (2012)

## Feature-based Recommendation for Product Configuration in the Software Product Lines

Yibo Wang<sup>1</sup>, Lothar Hotz<sup>2</sup>, Matthias Riebisch<sup>3</sup>

**Abstract:** Software Product Line Engineering (SPLE) is a mature technique enabling companies to create individual products in order to meet needs of different customers. In SPLE, Feature Models are the widely used formalism to capture commonality and variability of all products. In Feature Models, program functionalities or other user-visible aspects are represented as Features. In order to configure a product, product line users (such as product managers) select desired features step by step in the Feature Model. However, it is a challenging task in industrial settings, due to high numbers of features and complex interdependency between features. Configuration support is required.

In this paper, we propose a similarity-based recommender system that provides online recommendation during the user's configuration process. Online means that the recommendation result is based on the current feature selections (partial configuration) by the user. In addition, configurations of all previous products are considered as further input data for the recommender system. Unlike other similarity-based recommender systems, we use "Feature Implication" (it implies coexistence of features in the previous configurations) to measure relations between features. A real case study shows that our approach outperforms other state-of-the-art similarity-based approaches in recommendation quality. In other words, it helps users not only in finding the correct configuration, but also in making decisions more efficient.

**Keywords:** Product Configuration; Recommender Systems; Software Product Lines

### 1 Introduction

Nowadays, mass customization helps companies to meet customer requirements at mass production efficiencies. Software Product Lines Engineering (SPLE) is a mature technique to realize it. SPLE makes it possible to create an individual product by assembling a set of reusable assets. In the domain of SPLE, the Feature Model [Cz12] is one of the most used formalisms, which captures commonality and variability among all products in terms of Features. A Feature [Ka90] is defined as a "prominent or distinctive user-visible aspect, quality, or characteristic of a software system or system". Configuration is one of the most important activities to build an individual product from the product line. In the configuration process, some features will be selected by product line users (followings as users), in order to meet the individual requirements for a customer. But the configuration is more than feature selections. It includes also other activities, such as setting of the feature parameters.

---

<sup>1</sup> University Hamburg, SWK, Vogt-Kölln-Str.30, 22527 Hamburg, Germany, wang@informatik.uni-hamburg.de

<sup>2</sup> HITeC, Vogt-Kölln-Str.30, 22527 Hamburg, Germany, hotz@informatik.uni-hamburg.de

<sup>3</sup> University Hamburg, SWK, Vogt-Kölln-Str.30, 22527 Hamburg, riebisich@informatik.uni-hamburg.de

However, there exist approaches that change feature parameters into a group of exclusive features [KO14]. So we simplify the configuration as feature selections in this paper.

In industrial settings, a configuration is normally a complex task, because of high number of features and complex feature inter-dependencies. Moreover, the configuration is a gradual and iterative process. In each step some features will be selected (partial configuration). Users often don't have an overview of the whole configuration, due to lack of knowledge about impacts of their feature selections. Thus, automatic support is required to give users some guidances on the current feature selections. They exist in two forms: consistency checking and feature recommendation. Consistency checking is a hard form of configuration support. It ensures the validity of the current feature selections. It means that the feature selections that lead to inconsistency are not permitted. On the contrary, recommendation is a soft form. It provides only suggestions (e.g. "you should consider the feature 'X', because it exists in most of the previous configurations"). But these suggestions can be ignored. In this paper, we concentrate on the recommendation part and show how to integrate it with the existing consistency checking approaches (such as SAT solvers).

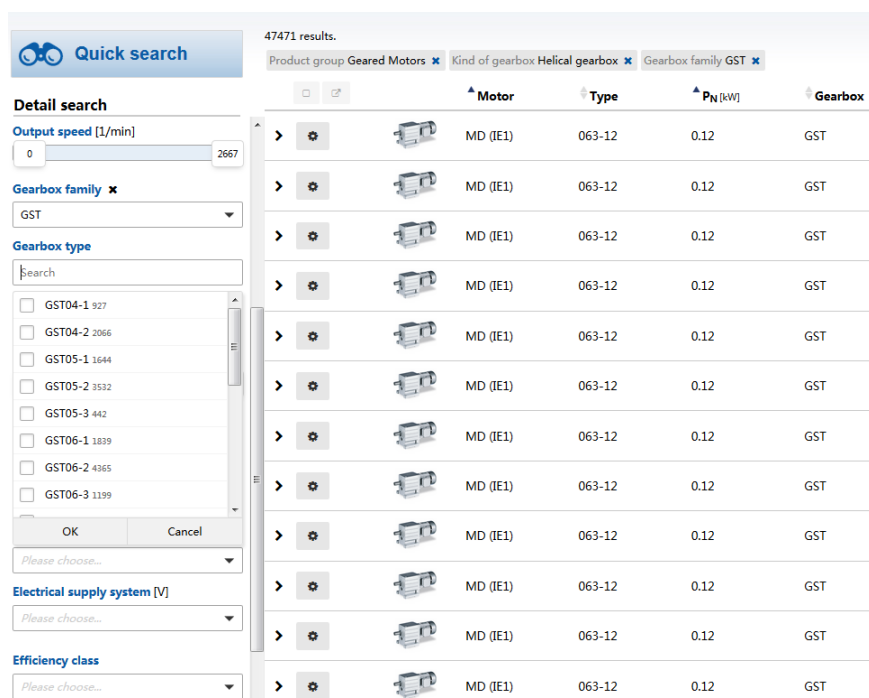


Fig. 1: Lenze configurator

In addition, our research is inspired by the Lenze configurator (Figure 1). In the configurator, if some options have been decided ("GST" selected as Gearbox family), we ask if it is

possible to provide suggestions for other options (recommend "GST06-2" as Gear type), in consideration of the set of all available product variants?

[Pe16] follows the similar ideas to provide recommendations. The authors suggest to use the standard recommenders, such as similarity-based Collaborative Filtering (in the following referred to as CF) and Bias Regularized Incremental Simultaneous Matrix Factorization (in the following referred to as BRISMF), because they provide better recommendation results than other state-of-the-art ones. Our approach goes a step further by adapting the standard recommendation algorithm CF to the specific problem, namely to the feature selection problem.

The contributions are summarized as follows. First we propose a similarity-based feature recommender system, by using "Feature Implication" derived from previous configurations to generate recommendations. Second, we design a tool support by extending a state-of-the-art recommender system. Last but not least, we evaluate the tool (in precision/recall and run time) with a real-world data set. The results show that it provides more precise recommendation results than the traditional similarity-based approaches.

## 2 Background

### 2.1 Product-Line Engineering

SPLE defines two main processes, namely the Domain Engineering and the Application Engineering. The Domain Engineering consists of the activities such as domain analysis, setting up feature model and preparation of reusable assets. The Application Engineering consists of the activities like requirements analysis, product configuration and product generation and integration.

A Feature Model consists of the feature diagram [Ka90] and other additional information such as Cross-Tree Constraints. A feature diagram is a graphical tree-like representation that shows the hierarchical organization of features (such as "mandatory", "optional" and "feature groups"), while Cross-Tree Constraints define relations among hierarchical not directly connected features (such as "requires" and "excludes"). The Feature Constraint is a generic term that refers to any type of existing feature relationship among features (hierarchical relationship plus cross-tree constraints). A configuration with respect to a given Feature Model is represented as an arbitrary combination of features. A configuration is valid in respect of a Feature Model, if and only if all the Feature Constraints are hold. In the Application Engineering, users must choose which features they want to have for an individual product. In this context, this decision process is also referred to as configuration.

### 2.2 Recommender Systems

Recommender Systems provide personalized recommendations for users in making decisions. According to the type of information available for making recommendations, they

can be divided into 3 groups: Collaborative, Content-based and Knowledge-based recommendation. Collaborative recommendation approaches exploit information about the past behaviour or the opinions of an existing user community for predicting which items the current user will most probably be interested in [Ja10]. While Content-based approaches are based on the availability of item descriptions and a profile that assigns importance to these characteristics, Knowledge-based approaches exploit additional and means-end knowledge such as constraints to generate recommendations [Ja10].

Collaborative recommendation is the most researched and used recommendation approaches in the recent years. User-based Collaborative Filtering (CF), item-based CF and Matrix factorization are the most important recommendation techniques in this category. The first two approaches calculate recommendations directly from the user-item ratings. While user-based CF compute recommendations using the similarity between users, item-based CF makes use of the similarity between items. Item-based CF is more apt for offline preprocessing and large-scale problems. Unlike CF, Matrix Factorization (MF) approaches must firstly learn from the user-item ratings and then use the learned model (latent factors) to make predictions. In general, CF approaches provide more precise results than MF approaches, because the full ratings are used for generating the recommendations [Ja10].

### 3 The proposed approach

#### 3.1 Configuration Process with Recommendation Support

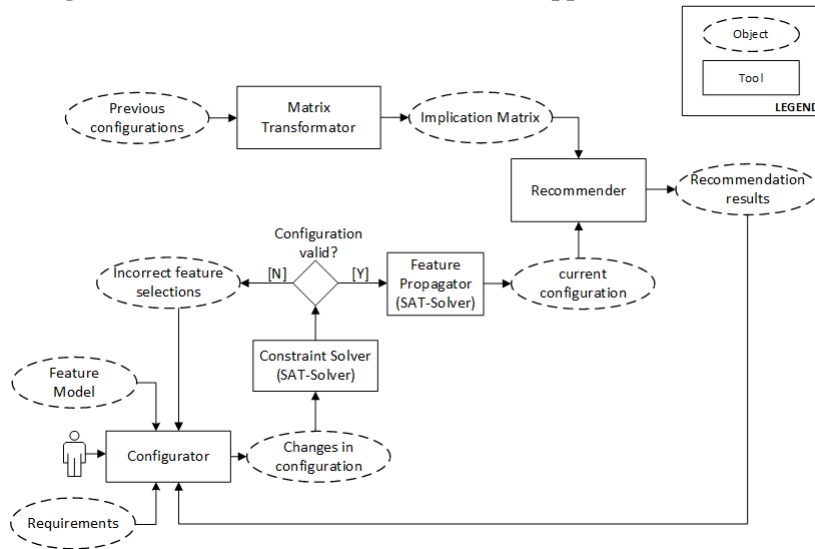


Fig. 2: Configuration process with recommendation support

To start with the configuration process, users should select/deselect features in a configurator (such as EngCon [Kr13], pure::variants [Be08], or FeatureIDE [TKB14]), according to

their individual requirements. The configurator presents the feature model graphically and enables interaction with users. The feature selections could lead to some changes in a configuration. Then, its validity will be checked by the constraint solver. If it is invalid, incorrect feature selections will be shown in the configurator and should be corrected by users. Otherwise, the feature propagator starts to propagate the user's selections onto other features. The current configuration after feature propagation (the lower part of Figure 2) becomes one part of the input for the recommender. The other part (the upper part of Figure 2) comes from the matrix transformation of previous configurations (more details in the following sections). Finally, the recommendation results will be generated by the recommender and given to users. Thus, the recommendations will be considered in the further configuration iterations.

### 3.2 Choosing Recommender Techniques

In our approach, one input for the recommender system are previous configurations. They are very similar to user-item ratings in the similarity-based collaborative filtering approaches. Thus, we initially choose Collaborative Filtering (in the following referred to as CF) as our recommender technique. Because of poor scalability in processing large numbers of configurations in the user-based CF, our approach is based on the item-based CF. In addition, we don't use Matrix Factorization techniques (such as BRISMf in [Pe16]) to improve the recommender, due to its poor performance in respect of run time (waiting time of over 30 seconds to generate a recommendation). In realistic scenarios, users should get a recommendation instantly after the current configuration step.

### 3.3 Formal Definition

In this section, we describe the formal representation of a configuration and definitions for feature recommendation. We follow the basic definitions in [Pe16] and make some amendments and extensions.

A **feature model**  $\mathbb{FM} = (\mathbb{F}, \mathcal{R})$  consists of a tuple of feature states  $\mathbb{F} = \{-1, 0, 1\}^h$  and a set of feature constraints  $\mathcal{R} = \{r_1, r_2, \dots, r_m\}$ , where  $h$  is the number of features and  $m$  the number of constraints. A **feature** has the **state** 1 if it has been selected (*positive decided*); -1 if it has been deselected (*negative decided*); 0 if it has not been decided (*undecided*). A **configuration**  $\vec{c}$  for a given feature model  $\mathbb{FM}$  is a tuple of states for all features, more formally as  $\vec{c} = \{f_1, f_2, \dots, f_h\}$ . A configuration is **complete**, iff each feature has a decided state (state = -1 or 1). A configuration is **partial**, iff it is not complete. In other words, it still has some features in undecided states (state = 0). According to their states, features in a configuration can be divided into 3 groups: PD (*positive decided*), ND (*negative decided*) and UD (*undecided*), such that  $PD \cap ND \cap UD = \emptyset$  and  $PD \cup ND \cup UD = \vec{c}$ . **Previous configurations** (in the following referred to as *PC*) are a collection of valid configurations from all existing products. In our work, it is represented as a matrix of  $n$  rows  $h$  columns,



where  $n$  is the number of previous configurations and  $h$  the number of features. Thus, each row represents an existing configuration.

$$PC = \begin{bmatrix} pc_{11} & pc_{12} & \dots & pc_{1h} \\ \vdots & \vdots & \ddots & \vdots \\ pc_{n1} & pc_{n2} & \dots & pc_{nh} \end{bmatrix} \quad (1)$$

Besides feature constraints directly defined in the feature model, additional feature relations could also be derived from  $PC$ . We call this type of feature relations as **feature implication**. Given two features  $f_x$  and  $f_y$ , it gives a hint on how likely  $f_y$  should also be selected, if  $f_x$  has been selected [Ma14]. It is also known as the confidence between two items in the Market Basket Analysis [TSK05]. The feature implication is reflexive and asymmetric. More formally, it is defined as<sup>4</sup>:

$$\text{confidence}(f_x \xrightarrow{\text{is given}} f_y) = \frac{|f_x \text{ and } f_y|}{|f_x|} \quad (1)$$

The formula in the numerator counts the number of configurations in  $PC$ , which have both  $f_x$  and  $f_y$ , while the formula in the denominator counts the number of configurations which have only  $f_x$ . The calculated confidence is a rational number between 0 and 1. Depending on its value, feature implication takes on different meanings, as described in table 1.

Tab. 1: Meaning of confidence

confidence( $f_x \rightarrow f_y$ ) = 0	If $f_x$ is selected, $f_y$ is never selected.
$0 < \text{confidence}(f_x \rightarrow f_y) < 1$	If $f_x$ is selected, $f_y$ is sometimes selected.
confidence( $f_x \rightarrow f_y$ ) = 1	If $f_x$ is selected, $f_y$ is always selected.

We use **implication matrix**  $I$  to record pairwise implication relation between all features. It is a square matrix with  $n$  rows and  $h$  columns, where  $h$  is the number of features. The element  $I(f_x, f_y)$  in the row  $x$  and the column  $y$  represents the feature implication from  $f_x$  to  $f_y$ , as stated in formula (1). In this way, implication matrix  $I$  can be transformed from previous configurations  $PC$  directly. As stated before, it constitutes an important input for the recommender.

### 3.4 Calculation of Recommendations

In the configuration process, recommendation would help users to make decisions on the undecided features. We calculate recommendation for the undecided features based on the user's current configuration and the previous configurations  $PC$ . More formally, given the feature sets  $PD$  (positive decided),  $ND$  (negative decided) and the implication matrix

<sup>4</sup> We changed "given" from [Ma14] to "is given" because  $f_x$  is selected and  $f_y$  is computed from this input.

$I$ , **recommendation score** for each feature  $f_{UD} \in UD$  (undecided) will be calculated as followings:

$$score(f_{UD}) = \left\| \frac{\sum_{f_{PD} \in PD} I(f_{PD}, f_{UD})}{|PD|} - \frac{\sum_{f_{ND} \in ND} I(f_{ND}, f_{UD})}{|ND|} \right\| \quad (2)$$

In the RHS of formula (2), the first part is the average feature implication from all positive decided features to the undecided feature. The second part is the average feature implication from all negative decided features to the undecided feature. The recommendation score is the absolute value of difference between the two parts. Its value is between 0 and 1. It reflects the relation from all decided features to the undecided feature. We can get the recommendation results based on it.

### 3.5 Implementation

We use the standard Apache-framework mahout<sup>5</sup> to implement the proposed recommender system, because it is open-source and flexibly extensible. The state-of-the-art SAT-solver SAT4J<sup>6</sup> is selected as our reasoning library to perform consistency checks. For the front-end, we are integrating our recommender with the product line configuration framework FeatureIDE[TKB14].

## 4 Evaluation

We evaluate our approach in light of its improvements for feature recommendation, in comparison to other traditional similarity-based CF recommenders. More precisely, we compare the recommendation results achieved from our approach, with those from the used-based CF recommender (in the following referred to as user-CF) and from the item-based recommender (in the following referred to as item-CF).

### 4.1 Configuration Datasets

We use the first dataset<sup>7</sup> from [Pe16] as previous configurations  $PC$ . It is a real-world data set with 170 previous configurations. The feature model is composed of 1652 features and provides a high-level representation of a product line in the business management content for a company. The configuration process is performed as customization of a specific product for each employee.

### 4.2 Experiment Design

To generate recommendations, the parameters for the user-based CF recommender (table 2) are set to the same values as those in [Pe16], because they are optimized for this data set.

<sup>5</sup> Recommender engine, <https://mahout.apache.org/users/recommender/recommender-documentation.html>

<sup>6</sup> SAT-solver, <http://www.sat4j.org/>

<sup>7</sup> ERP System under <http://wwwiti.cs.uni-magdeburg.de/~jualves/PROFile/#datasets>

Tab. 2: Parameter values for different recommenders

Recommender	Parameter	ERP System
user-CF	$\tau$	0.000001
	similarity measure	Jaccard Index
item-CF	similarity measure	Jaccard Index
our approach	similarity measure	<b>Feature Implication</b>

The difference between our approach and the traditional item-based recommenders is that we use feature implication as similarity measure. In the data validation phase, we separate the configurations in *PC* into a training set and a test set, according to the Leave-one-out cross-validation (LOOCV) method. That is to say, the data validation iterates over all configurations. In each iteration, the following steps are performed:

1. A configuration is left out from *PC* and is put into the test set (it is called the **test configuration**);
2. The remaining configurations in *PC* are put into the training set (it is called the **training configurations**);
3. Part of the test configuration is copied to the current configuration;  
In this step, we also simulate the progress of a user in the configuration process and give gradually (stepwise 10%) parts of the test configuration to the current configuration.
4. Recommendations are generated based on the current configuration and the training configurations;
5. The generated recommendations are compared with the test configuration;
6. Results are evaluated against metrics of precision, recall and F-measure at  $\omega$ .  
 $\omega$  is the number of permitted recommendations for a recommender. It is set to 10 as in [Pe16].

Let **relevant features** be the features that are both in the recommendation results and selected in the test configuration. **Precision** is the fraction of relevant features among all recommended features. **Recall** is the fraction of relevant features over the total amount of selected features in the test configuration. F-Measure combines the metrics precision and recall as followings:

$$\text{F-Measure} = \frac{2 \bullet \text{Precision} \bullet \text{Recall}}{\text{Precision} + \text{Recall}}$$

### 4.3 Experimental Results and Discussion

We execute the experiments with the aforementioned 3 recommenders on 9 completeness levels. For each combination, we evaluate the recommender with the average results of 10 runs to avoid bias. All the experiments were performed on a computer with Quad Core@2.90 GHz CPU and 16 GB RAM, running on Windows 7.

Because of space limitation, in Table 3 we show only three results on the completeness levels 10%, 50%, and 90%. The Figure 3 presents the achieved F-Measures and run times taken by 3 recommenders, each with the increasing completeness of the current configuration.

Tab. 3: Evaluation results in F-Measure, Precision, Recall and Run time

Completeness of configuration	Recommender	F-Measure	Precision	Recall	Run time (in seconds)
10%	user-CF	0,2126	0,8941	0,1207	5
	item-CF	0,1839	0,8888	0,1026	4
	Our approach	0,2047	0,9129	0,1152	4
50%	user-CF	0,3267	0,8876	0,2002	11
	item-CF	0,3050	0,8859	0,1842	14
	Our approach	0,3251	0,8976	0,1985	16
90%	user-CF	0,5351	0,5946	0,4864	13
	item-CF	0,5588	0,6220	0,5073	21
	Our approach	0,6332	0,7214	0,5643	24

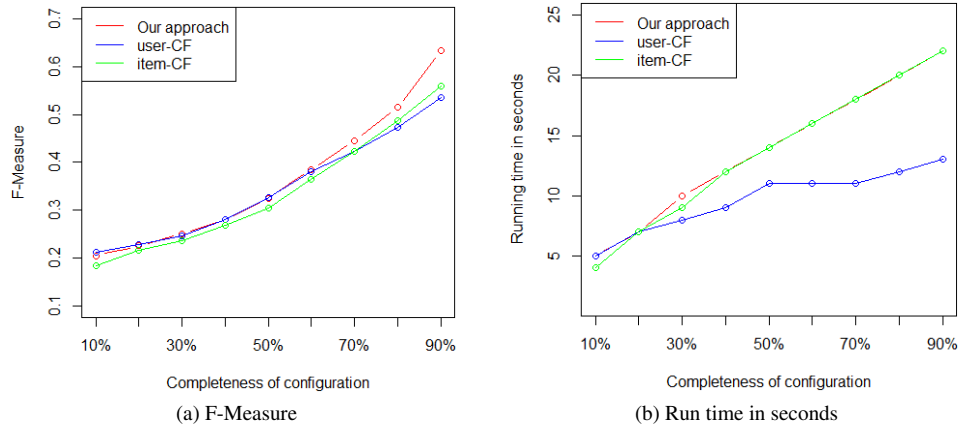


Fig. 3: Evaluation results

We make the following observation:

- Our approach outperforms the traditional item-based CF in F-Measure at all completeness levels.
- Our approach outperforms the traditional user-based CF in F-Measure from the 60% completeness level.
- Our approach has the longer run time than the user-based CF, while it remains comparable with the item-based CF at all completeness levels.

In general, the results show that feature implication helps to find more precise recommendations. This tendency becomes more clear, as the completeness of the current configuration increases. It means our approach will generate better recommendation results in the later phases of a configuration. In addition, recommendations are generated by our approach at a

comparable speed by the traditional item-based CF one. However, it is more slower than the user-based CF recommender. We plan to evaluate our approach with a bigger dataset to investigate this finding.

## 5 Related work

### - Configuration guidance

[Fe10] presents an configuration environment that supports personalised configuration of mobile phones. It integrates recommendation techniques (such as similarity-based ranking) with knowledge-based configuration (such as consistency checking). However, the investigated configuration problems are relative simple, in comparison to the problems in the software product lines. [BE13] develops a mechanism that stakeholders are able to identify the utility of each feature of the product line by answering gamble questions. The advantage of this approach is that it does not need to have any information about existing configurations. But the interaction costs with users are increasing exponentially.

### - Configuration optimization

[HP15] augments multi-objective search-based optimization with constraint solving, in order to find the optimal configuration efficiently. [Hi16] improves the approach by optimising first on the number of constraints that hold and then on the other objectives. Although optimization provides precise results in configuration support. But in general, it demands a high computation cost and can not provide an understandable explanation for the results.

### - Quality aware Feature Model Configuration

[ZYL10] proposes an Analytic Hierarchical Process (AHP) based approach to calculate the relative importance of each feature on a quality attribute. They use these importance values to estimate the impact of feature selections on a quality attribute. [TLL14] proposes an improvement by replace AHP with ELO rating system. [SSPS10] extends the traditional SPLE with a feedback approach in order to improve the configuration of NFPs. The estimation of quality attributes for a configuration is based on the measurement results on a generated testing set. A similar idea is also followed by [Si11]. Unlike our approach, these approaches utilize either the experts' knowledge or the measurements on a small test set to generate estimation.

## 6 Conclusion and future work

In this paper, we extend the traditional item-based recommender with "feature implication" as a customized similarity measure. Results show that "feature implication" helps to get more precise recommendation results than other similarity-based approaches.

In the next step, we will categorize recommendation types and evaluate them with a more complex data set. Then, the integration with the consistency checker will be implemented. Other recommendation possibilities, such as configuration sequences, will be investigated later.

## References

- [Be08] Beuche, Danilo: Modeling and building software product lines with pure::variants. Proceedings - 12th International Software Product Line Conference, SPLC 2008, p. 358, 2008.
- [BE13] Bagheri, Ebrahim; Ensan, Faezeh: Dynamic decision models for staged software product line configuration. *Requirements Engineering*, 19(2):187–212, 2013.
- [Cz12] Czarnecki, Krzysztof; Grünbacher, Paul; Rabiser, Rick; Schmid, Klaus; Wasowski, Andrzej: Cool features and tough decisions. In: *VaMoS '12 The 6th International Workshop on Variability Modeling of Software-Intensive Systems*. ACM Press, New York, New York, USA, pp. 173–182, 2012.
- [Fe10] Felfernig, Alexander; Mandl, Monika; Tiihonen, Juha; Schubert, Monika; Leitner, Gerhard: Personalized user interfaces for product configuration. In: *Proceedings of the 15th international conference on Intelligent user interfaces - IUI '10*. ACM Press, New York, New York, USA, p. 317, 2010.
- [Hi16] Hierons, Robert M; Li, Miqing; Liu, Xiaohui; Segura, Sergio; Zheng, Wei: SIP: Optimal Product Selection from Feature Models Using Many-Objective Evolutionary Optimization. *ACM Transactions on Software Engineering and Methodology*, 25(2):1–39, April 2016.
- [HP15] Henard, Christopher; Papadakis, Mike: Combining multi-objective search and constraint solving for configuring large software product lines. In: *ICSE' 15 Proceedings of the 37th International Conference on Software Engineering*. pp. 517–528, 2015.
- [Ja10] Jannach, Dietmar; Zanker, Markus; Felfernig, Alexander; Friedrich, Gerhard: *Recommender Systems*. Cambridge University Press, Cambridge, 2010.
- [Ka90] Kang, K.; Cohen, S.; Hess, J.; Novak, W.; Peterson, S.: *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-021, 1990.
- [KO14] Karatas, Ahmet Serkan; Oguztuzun, Halit: Attribute-based variability in feature models. *Requirements Engineering*, pp. 185–208, 2014.
- [Kr13] Krebs, T.: EngCon. In (Felfernig, A.; Hotz, L.; Bagley, C.; Tiihonen, J., eds): *Knowledge-based Configuration – From Research to Business Cases*, chapter 23, pp. 337–346. Morgan Kaufmann Publishers, 2013.
- [Ma14] Martinez, Jabier; Ziadi, Tewfik; Mazo, Raul; Bissyande, Tegawende F.; Klein, Jacques; Traon, Yves Le: Feature Relations Graphs: A Visualisation Paradigm for Feature Constraints in Software Product Lines. In: *2014 Second IEEE Working Conference on Software Visualization*. IEEE, pp. 50–59, sep 2014.
- [Pe16] Pereira, Juliana Alves; Matuszyk, Pawel; Krieter, Sebastian; Spiliopoulou, Myra; Saake, Gunter: A feature-based personalized recommender system for product-line configuration. i, *ACM Press, New York, New York, USA*, pp. 120–131, 2016.
- [Si11] Siegmund, Norbert; Rosenmüller, Marko; Kuhlemann, Martin; Kästner, Christian; Apel, Sven; Saake, Gunter: SPL Conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, 20(3-4):487–517, jun 2011.

- [SSPS10] Sincero, Julio; Schröder-Preikschat, Wolfgang; Spinczyk, Olaf: Approaching non-functional properties of software product lines: Learning from products. Proceedings - Asia-Pacific Software Engineering Conference, APSEC, pp. 147–155, 2010.
- [TKB14] Thüm, T; Kästner, C; Benduhn, F: FeatureIDE: An extensible framework for feature-oriented software development. Science of Computer Programming, 79:70–85, 2014.
- [TLL14] Tan, Lei; Lin, Yuqing; Liu, Li: Quality Ranking of Features in Software Product Line Engineering. In: 2014 21st Asia-Pacific Software Engineering Conference. volume 2. IEEE, pp. 57–62, dec 2014.
- [TSK05] Tan, Pang-Ning; Steinbach, Michael; Kumar, Vipin: Introduction to Data Mining, (First Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.
- [ZYL10] Zhang, Guoheng; Ye, Huilin; Lin, Yuqing: Quality attributes assessment for feature-based product configuration in software product line. In: Proceedings - Asia-Pacific Software Engineering Conference, APSEC. pp. 137–146, 2010.

## Feature-oriented Domain-specific Languages

Philipp Ulsamer<sup>1</sup>, Tobias Fertig<sup>2</sup>, Peter Braun<sup>3</sup>

**Abstract:** Domain-specific Languages (DSL) describe applications of particular domains that can be understood by domain experts. However, common users have to learn a new domain language. Moreover, developing similar software products from scratch is a time consuming and tedious task. Therefore, we developed an internal DSL to describe RESTful systems. In order to use that one must have very good knowledge about REST. Feature models are a common paradigm in Product Line Engineering to represent the features of the products as reusable parts. They can be easily communicated with customers. That is why we have extended our approach with a model-to-model transformation. This transformation provides a model for an underlying REST generator. While a model-to-code (M2C) generator is developed domain-specific, excluding expansion by a new domain, our approach is to make it more efficient than developing a M2C generator for every single domain. So that non-experts can configure REST solutions by using feature models without knowledge of REST.

**Keywords:** MDSD; DSL; REST; Hypermedia; RESTful; SPL; Domain Engineering; Feature Modeling; Model Transformation; M2M

### 1 Introduction

Model manipulation plays the central role in Model-Driven Engineering. In 2015 we presented our project Generating Mobile Applications with RESTful Architecture (GeMARA) [SB15] in which we proposed a model-driven approach for creating RESTful APIs. For the purpose of focusing on a particular domain, we developed an external Domain-specific Language (DSL) using the Eclipse plugin Xtext [EB10] for defining the grammar of the DSL and the programming language Xtend [Be13] to implement the software generator. The limited expressiveness and the fluency from combined individual expressions, and the domain focus makes our DSL easy to be understood by domain experts [Fo10]. However, there are several shortcomings. A user has to learn a new language which requires considerable time to learn.

---

<sup>1</sup> Univesity Of Applied Science Wuerzburg-Schweinfurt, FIW, Sanderheinrichsleitenweg 20, 97074 Wuerzburg, Germany philipp.ulsamer@fhws.de

<sup>2</sup> Univesity Of Applied Science Wuerzburg-Schweinfurt, FIW, Sanderheinrichsleitenweg 20, 97074 Wuerzburg, Germany tobias.fertig@fhws.de

<sup>3</sup> Univesity Of Applied Science Wuerzburg-Schweinfurt, FIW, Sanderheinrichsleitenweg 20, 97074 Wuerzburg, Germany peter.braun@fhws.de



Also, developing, evolving, and maintaining complex external DSLs requires significant effort. As our project matured, we switched to an internal DSL using Java as the General Purpose Programming Language (GPL) [SB15].

Product Line Engineering (PLE) has the goal of developing software systems from reusable parts instead of developing them from scratch [Ap13]. The feature-oriented approach was originally proposed as part of the Feature-Oriented Domain Analysis (FODA) method [Ka90]. The main idea is to express every product configuration on a higher level of abstraction using the feature models paradigm. While development is easier to implement by Java developers, the knowledge of REST is indispensable to understand the domain in our GeMARA project.

We address this problem by extending this approach with a model-to-model (M2M) transformation. Features of a given feature model will be transformed to a RESTful finite-state machine ( $\epsilon$ -NFA) [ZBD11]. Connecting those features will lead to an  $\epsilon$ -NFA that represents a valid specification of the product. Using M2M transformation allows less experienced users to define their product using feature-based configuration files. Therefore, the user will not need any knowledge about describing RESTful systems as  $\epsilon$ -NFAs. Additionally, the automatic generation of program code by a generator is characteristic of generative programming. This automatic generation is also known as model-to-code (M2C) transformation. In order to carry out an M2C transformation, a generator must be developed for each individual domain, the development effort which is always different and thus represents a time-consuming task. The M2M transformation provides a model for the underlying REST generator in GeMARA, which performs a model transformation on the incoming feature model. This facilitates the addition of various forms of a domain and their maintenance.

First, we will summarize related work and confirm that feature modeling used as an abstraction in hypermedia for M2M transformation is a missing task. Afterwards we will discuss challenges of using REST in model transformation. Furthermore, we will propose our approaches for solving these challenges. Finally, we will give a short outlook and discuss our future work.

## 2 Related Work

Voelter and Viser propose the method of using DSLs in PLE [VV11]. They show how DSLs can fill the gap between feature modeling and programming languages, when a feature model exceeds its limit. Expressing complex features in their behavior to each other is not possible with feature models. Feature models can be used to select among a set of few predefined behaviors, composition rules or rationale in FODA [Ka90]. Picturing all possible behaviors would lead to a confusing feature model. In order to fix this issue, the behavior could be implemented in a GPL that would lower the abstraction that feature models offer, reducing the understandability, as well as the possibility for non-programmers to participate. DSLs can counteract and be the mediator between the composition rules and the lack of restrictions of a GPL. A missing topic is the abstraction via M2M transformation. While the transformation of models is under the control of the domain engineer when working with DSLs we need to lighten the usage of the DSL.

Cuadrado and Molina deal with embedded DSLs and its model transformation in their work. [CM07] A key aspect of MDSD is Metamodeling, which defines the abstract syntax of a DSL. Since graphical editors are the most favorable way to define Metamodels, developing one is a time-consuming task. Nevertheless, they propose a M2M transformation approach transforming a UML model into a Java model. Our approach focuses on feature models to grant a certain degree of reuse of our features. For a definition of a new feature selection, you only have to choose between a preconfigured set of allowed feature combinations instead of defining a UML for each selection.

## 3 GeMARA

The goal of our research project is to develop automated Generators for distributed Mobile Applications based on RESTful Architecture (GeMARA) by providing a model-driven approach. Accordingly, applications are no longer described as source code, but as an abstract model from which source code and other artifacts are generated.

We proposed our approach in 2015 in [SB15]. Back then we were able to generate RESTful APIs including the persistence layer. Our generators were written with Xtext [EB10] and Xtend [Be13]. However, as our project matured, the maintenance of our generators got a more and more time consuming task. Therefore, we cut loose from Xtext and Xtend and developed our own internal DSL in Java. The new DSL describes RESTful Systems as finite state machines ( $\epsilon$ -NFA) according to Zusak et al. [ZBD11]:

States are defined by the HTTP Verb and the resource. Transitions define the hypermedia links. Secondary States are defined by the HTTP verb, the resource and the subresource. Secondary States can be used to describe relations between resources, for example between users and their addresses.

We can generate every RESTful System described as  $\varepsilon$ -NFA. By now we have also generators for Android applications and Polymer frontends. Moreover, we are generating test cases for the RESTful API [FB15] and Hypermedia Tests [VFB17]. The generated source code can be deployed out of the box. However, sometimes manual adjustments are necessary. Those are also supported by our approach using dependency injection.

## 4 Challenges

MDSD is a discipline in software engineering to generate source code from an abstract model [SVC06] to successfully develop, evolve and maintain software. Instead of using a GPL to generate code with a set of statements, conditions and loops, MDSD abstracts the development on a higher level. While models are only useful if they provide a better understanding of the software system, the same applies to their graphical representation (e.g. UML) [MV06]. However, a feature model consists not only of a feature diagram, but also of the additional information relevant to the project. Thus, a brief description of the features is necessary for the later understanding. Also, boundary conditions and priorities are documented (e.g., to express the interest of the stakeholders or customers in a certain future) [Ka90]. In our work we will match all these conditions in our GeMARA model in order to reduce effort and complexity involved in the development and adoption process.

Unfortunately, developers are required to have programming skills and a thorough understanding of the principles of REST in order to develop models within our project GeMARA [SB15], which leads to complexity in our domain engineering process. In order to eliminate this problem, we will abstract every  $\varepsilon$ -NFA [ZBD11] that represents a valid specification of the product into a named feature. A domain users job is only to select a valid feature selection to generate a product. Therefore, the user does not need any knowledge about describing RESTful systems as  $\varepsilon$ -NFA.

The model transformation process is the emphasis of our approach. In order to perform a M2C transformation, we need to develop a generator for every single domain, whose development effort is always different, indeed a time-consuming task. With M2M transformation we will provide a model for our underlying REST generator that performs model transformation for every incoming domain model. Furthermore, the risk is that templates becomes bulky in scope in some target architectures or languages. From a certain complexity of the transformations and abstraction of the output model a M2M transformation is indispensable.

## 5 Feature & REST Modeling

A feature is a domain abstraction and represents requirements, similarities or differences of program variants and serves as a means of communication between stakeholders and developers [Ap13]. A feature diagram is a graphical representation of all possible and valid feature configurations of a specific domain. It consists of mandatory, optional, alternative and logical-OR features and other requires rules. As can be seen in Figure 1, the feature Events with all its subfeatures and dependencies as an abstract from the feature model of the domain martial arts academies is displayed.

Instead of pursuing a procedure of developing an internal or external DSL with its own grammar, a model transformation is used to derive a DSL that can only be reduced to a selection of features and specification of meta information. There is no special knowledge about REST needed for the use of the model.

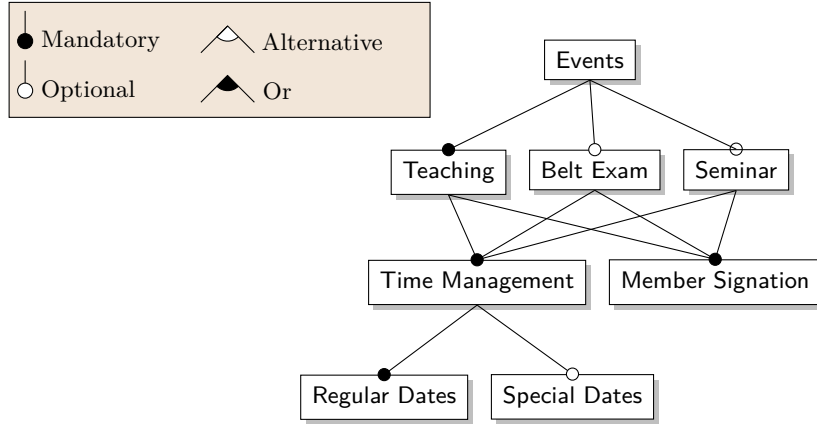
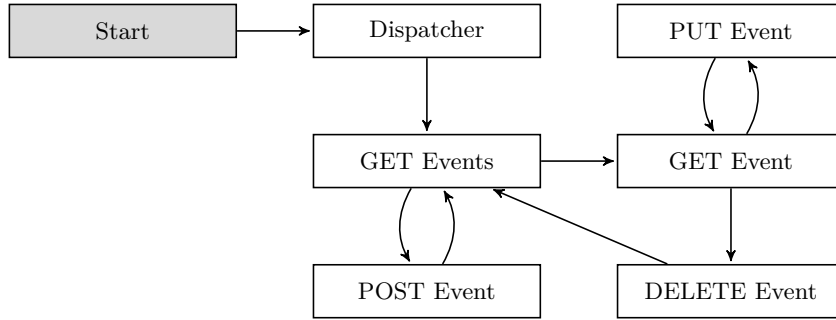


Fig. 1: Feature selection of the domain martial arts academies.

While GeMARA is based on REST, the selected feature model must also be mapped into REST in order to be able to transform. We created an abstract model of the technical domain of REST based on our understanding of Fieldings constraints [Fi00]. The GeMARA model has one dispatcher state and multiple application states for all CRUD operations. The dispatcher state represents the initial state of the  $\epsilon$ -NFA. Every application state is defined by an HTTP verb and a resource. A client can navigate from one application state to another via transitions, which is defined by hyperlinks. Figure 2 shows the state transition diagram of the feature Events.

Fig. 2: Example of a  $\varepsilon$ -NFA of the resource Event.

We put a new domain model on top of the technical REST model. Every feature in a domain model is represented as an  $\varepsilon$ -NFA using the given feature type by the feature model. A feature type represents which HTTP verbs are supported for every required resource. This allows the creator of the domain to limit the CRUD operations that can be used on a resource. Thus, a feature of a valid feature selection may have multiple resources. These are usually generated separately from each other. With the help of Secondary States, the  $\varepsilon$ -NFA of the resources are linked together in order to be able to fully describe a feature.

## 6 Model Transformation

Our M2M transformation consists of two steps. Firstly, the domain developer has to provide a feature factory and a feature validator for their domain. Secondly, the user has to define the required features within a YAML file.

Our generic implementation parses the YAML file and uses the given feature validator of this domain. If the configuration of the user passes the validators tests it can be transformed into a GeMARA model. The tests checks whether the combination of mandatory, optional and alternative features is valid. Therefore, the transformer translates the YAML file into an internal representation of features. The feature factory of the developer is used within this step. The developer has to define the  $\varepsilon$ -NFA for every feature a user can define in his feature factory.

We divided the transformation process into three steps: resource, state and transition transformation. Our Transformer uses the generic sub-transformers to create a GeMARA model that can then be used to generate the RESTful System. (see Figure 3).

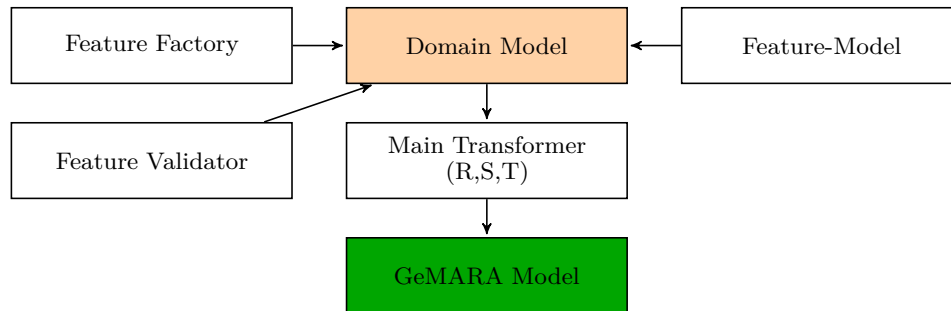


Fig. 3: Feature-oriented approach of a model transformation

In order to work with the DSL, the user has to know defined rules for the feature selection. A list of features and their detailed description can be found in the domain dictionary. In a YAML text document, the user defines only a selection of features that he would like to have in his RESTful system and must also provide database information (Listing 1).

```

featureList:
  - Events
  - Teaching
  - Member Signation
  - [...]

databaseName: Lee
databaseHost: localhost
databaseUserName: user
databasePassword: 1234

```

List. 1: Example of a feature selection in YAML for the domain Martial Arts Academy

Afterwards, the correct YAML document, the feature factory, and the validator for the domain have to be selected for the model transformation (Listing 2).

```

domainModel = new domainModel( );
gemaraModel = domainModel.setYamlPathName( "xyTest.yaml" )
                    .setFeatureFactory( new xyFeatureFactory( ) )
                    .setValidator( new xyValidator( ) )
                    .build( );
final ModelGenerator modelGenerator = new ModelGenerator(gemaraModel);
modelGenerator.generate( );

```

List. 2: Generation of a RESTful system of the domain Martial Arts Academy

## 7 Discussion

Our vision is to generate RESTful Systems that can be configured by users who are less experienced in programming or REST. Feature models have been shown to be generally understandable, to illustrate functions, their similarities, differences and restrictions. By configuring different models, reuse is possible because many features are mandatory components of each product.

Previously, we had to develop the resources, states and transitions separately. The model transformation allows us to draw their development to a higher level of abstraction that they can be generated. This can reduce manual implementations as well as repetitive routine tasks that would normally occur in an M2C approach. In addition, the development effort of a transformation in its extension is much more efficient than code generation. Not only does the domain model allow additional domains to be modeled by a few lines of code, because most of the components of the model transformation can be reused, but the variability of the feature model provides a multitude of possibilities, such as the user of a domain by specifying a feature selection, it can generate a complete RESTful System. A similar development with the M2C transformation would require the development of code generators for each domain. This would also be more time consuming than our approach.

The abstract representation of a model transformation not only makes the development of a domain model simpler, but also its use. The transformation from the domain model to a GeMARA model simplifies the generation of a RESTful System by only having to make a selection of features after a specific domain policy. The feature-oriented DSL only requires the user to enter a list of features in a YAML text document and to provide additional meta-information.

## 8 Outlook

The development of an M2M transformation has shown that a higher level of abstraction can automate many recurring tasks. Even if its enhancement requires less implementation than an M2C transformation, defining features in a Feature Factory of the domain model is still a time-consuming but undemanding task. In turn, a code generator could be written that generates the information for a feature, its resources and attributes. Also, for the development of the Feature Validator of a domain, a separate DSL could be written to simplify and streamline statements related to the constraints of a feature model and minimize error proneness. Thus, the extension of a domain could also be reduced to a few lines of code. Finally, language workbenches are becoming more and more powerful and user friendly [Fo05]. Their development and offer to tools far exceeds the current development of DSLs. Future work will focus on M2M transformation using language workbenches for our RESTful systems.

## References

- [Ap13] Apel, S.; Batory, D.; Kstner, C.; Saake, G.: *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer Publishing Company, Incorporated, 2013.
- [Be13] Bettini, L.: *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing, 2013.
- [CM07] Cuadrado, J. S.; Molina, J. G.: *Building Domain-Specific Languages for Model-Driven Development*. *IEEE Software* 24/5, 2007.
- [EB10] Eysholdt, M.; Behrens, H.: *Xtext: Implement Your Language Faster Than the Quick and Dirty Way*. In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*. OOPSLA '10, ACM, Reno/Tahoe, Nevada, USA, pp. 307–309, 2010.
- [FB15] Fertig, T.; Braun, P.: *Model-driven Testing of RESTful APIs*. In: *Proceedings of the 24th International Conference on World Wide Web Companion*. WWW '15 Companion, International World Wide Web Conferences Steering Committee, Florence, Italy, pp. 1497–1502, 2015.
- [Fi00] Fielding, R.: *REST: Architectural Styles and the Design of Network-based Software Architectures*, Doctoral dissertation, University of California, Irvine, 2000.
- [Fo05] Fowler, M.: *Language Workbenches: The Killer-App for Domain Specific Languages?*, 2005, URL: <http://www.martinfowler.com/articles/languageWorkbench.html>.
- [Fo10] Fowler, M.: *Domain Specific Languages*. Addison-Wesley Professional, 2010.
- [Ka90] Kang, K. C.; Cohen, S. G.; Hess, J. A.; Novak, W. E.; Peterson, A. S.: *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, tech. rep., Carnegie-Mellon University Software Engineering Institute, Nov. 1990.
- [MV06] Mens, T.; Van Gorp, P.: *A Taxonomy of Model Transformation*. *Electron. Notes Theor. Comput. Sci.* 152/, pp. 125–142, Mar. 2006.
- [SB15] Schreibmann, V.; Braun, P.: *Model-Driven Development of RESTful APIs*. In: *Proceedings of the 11th International Conference of Web Information Systems and Technologies*, Lisbon, Portugal. INSTICC, SciTePress, pp. 5–14, May 2015.
- [SVC06] Stahl, T.; Voelter, M.; Czarnecki, K.: *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.



- [VFB17] Vu, H.; Fertig, T.; Braun, P.: Towards model-driven hypermedia testing for RESTful systems. In: WEBIST 2017 - Proceedings of the 13th International Conference on Web Information Systems and Technologies. 2017.
- [VV11] Völter, M.; Visser, E.: Product Line Engineering Using Domain-Specific Languages. In: Software Product Lines - 15th International Conference, SPLC 2011, Munich, Germany, August 22-26, 2011. Pp. 70–79, 2011.
- [ZBD11] Zuzak, I.; Budiselic, I.; Delac, G.: Formal Modeling of RESTful Systems Using Finite-State Machines. In (Auer, S.; Díaz, O.; Papadopoulos, G. A., eds.). Springer Berlin Heidelberg, chap. Web Engineering: 11th International Conference, ICWE 2011, Paphos, Cyprus, June 20-24, 2011, pp. 346–360, 2011.

## Using PLC Programming Languages for Test-Case Specification of Hardware-in-the-loop Tests

David Thönnessen,<sup>1</sup> Stefan Kowalewski<sup>2</sup>

**Abstract:** Testing cyber-physical production systems (CPPSs) pose particular problems to testing control systems like Programmable Logic Controllers (PLCs). CPPS continuously underlie reconfiguration such that testing before commissioning is no longer sufficient. Because of that, it is crucial to have a testing environment supporting adaptive test cases and allowing for an efficient test case specification. Our approach consists of a modular specification architecture using slightly extended PLC programming languages. By doing this, we avoid the change of methodology observed when using dedicated test languages and corresponding tools. Our hypothesis is that this will lead to faster and more reliably changeable test case descriptions and thus will create the desired agility.

**Keywords:** Programmable Logic Controller; Automation; Testing; Simulation

### 1 Introduction

This paper deals with testing of cyber-physical production systems (CPPSs) [BtHVVH14] before and after commissioning. Testing CPPSs is essential because these systems underlie reconfiguration and changing environmental conditions. Our approach is to optimize Hardware-in-the-Loop (HiL) testing for the application in automation [Gu07] and allow testing after commissioning a control system. With this goal, we extended Programmable Logic Controller (PLC) programming languages [IE13] to qualify them for test case specification. We will describe the test process as well as the introduced language extensions in Section 4. Our initial scenario consists of a plant controlled by a PLC. The plant provides sensor information that are captured by the PLC. A control program processes the inputs and generates outputs for the actuators of the plant. We will call the PLC more generally System Under Test (SUT) in this scenario. When testing with HiL, the so-called HiL simulator replaces the plant. It simulates the sensors of the plant and, in return, evaluates the control signals of the SUT, in the following named actual signals. One approach for evaluating these signals is to compare them to a specification as done in the tools WCOMP or Arttest [Wi17, Zh07]. The tester has to implement this specification defining the reference behavior of the SUT as a function of its output signals. A complete specification allows calculating a reference signal for every actual signal. Within the HiL simulator we compare

---

<sup>1</sup> RWTH Aachen University, Informatik 11 - Embedded Software, Ahornstraße 55, 52074 Aachen, Germany  
thoennessen@embedded.rwth-aachen.de

<sup>2</sup> kowalewski@embedded.rwth-aachen.de

these reference signals against the actual signals in order to decide whether the behavior of the SUT is valid or not [Th17]. This paper is an abbreviated version of our submitted contribution [TK18].

## 2 Architecture

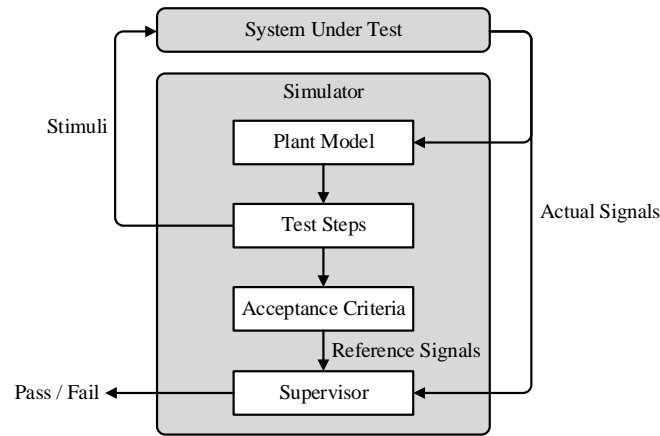


Figure 1: Simplified information flow within the HiL simulator.

We partition a test-case specification into the modules Plant Model, Test Steps, and Acceptance Criteria. Figure 1 shows the information flow between them. Every module is a Program Organization Unit (POU) of type program [IE13] and thereby can be modified independently from the others. Besides these three modules that are specified by the tester there is the so-called Supervisor. The Supervisor carries out the comparison between the reference and the actual signals. The reference signals come from the Acceptance Criteria and the actual signals are captured by the HiL simulator. The Supervisor accepts signals in form of sample points in discrete value- and time-domain (Sec. 4.1).

### 2.1 Modules

The Plant Model consists of a physical model of the plant and updates it with the output signals of the SUT in order to deliver realistic inputs to the SUT. Realistic means in this context that the generated data has to comply sufficiently with those in normal operation of the plant. The goal of the tester is to specify the Plant Model with as little effort as possible and as detailed as necessary. We will point this out at the example of a temperature-regulated liquid tank. The tank is equipped with a heater and we assume the liquid level to be constant 90 %. In normal operation, the SUT regulates the heater on basis of a temperature sensor. It enables the heater if the temperature falls below 50 °C and disables it as soon as it exceeds

55 °C. For testing the SUT the HiL simulator has to model the temperature sensor value. We assume that there are no external influences, so the temperature is solely influenced by the heater. The physical model starts with an initial temperature defined by the tester and models the heating or cooling of the liquid during execution, on basis of the control signal of the heater. The HiL simulator delivers the resulting sensor value to the SUT. For other, more complex examples, the Plant Model has to calculate additional information that is not transmitted to the SUT but required to calculate sensor values. We denote such information as virtual signals. An example for this is a conveyor belt with a workpiece on it. The belt is driven by a motor and equipped with a light barrier at both the beginning and the end. While the SUT is only interested in the sensor information of the light barriers we have to model the exact position of the workpiece on the belt. The position serves as calculation basis for the switching status of the light barrier but is not handed over to the SUT. The modules Test Steps and Acceptance Criteria can make use of virtual signals. For example, the tester could simulate pressing the emergency stop switch when the workpiece passes a given position on the belt. Within the module Test Steps the tester has the possibility to stimulate the SUT. In the beginning phase of the test the tester uses stimuli in order to transfer the SUT to a state to test. After this phase the tester uses stimuli to provoke test scenarios, e.g. activating the emergency stop. Stimuli can be seen as manipulations of sensor values leading to a state change of the SUT. If a tester wants to intervene in the system they have to manipulate the normal state of the system. Figure 2 shows the special

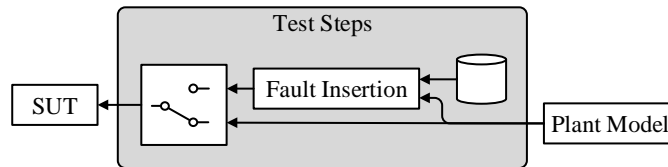


Figure 2: Manipulation of sensor data of the Plant Model by Test Steps.

case of a so-called Fault Insertion Test [IE11, Na16]. In this context, the tester has the possibility to switch between single sensor values of the Plant Model and own values. When they switch to own sensor values they are able to arbitrary manipulate the sensor values and pass them to the SUT. Within the Acceptance Criteria, the tester defines expected results for a set of output signals of the SUT. They specify the set but do not have to incorporate all signals. The simulator ignores signals in its evaluation which are not included in this set. When specifying those signals the tester is able to access data from the Plant Model and Test Steps (see virtual signals). The Supervisor acts as a superordinate instance of the test case. It compares the reference signals specified in the Acceptance Criteria to the actual signals captured by the SUT. It evaluates the test as passed if the signals equal, and, in return, as failed if there are deviations. The goal of the presented architecture is to ensure efficiency and adaptability of test cases. The first aspect is the reusability of the plant model. As soon as the tester specified the model they can reuse it for subsequent test cases. The second aspect results from the fact that the outputs of the module Acceptance Criteria are isolated from the modules Plant Model and Test Steps and thereby have no effect on the

test execution. Using this, the simulation can be recorded and run again at a later point in time, even without having the SUT connected. This is called postsimulation. For recording the simulator stores all inputs and outputs of the SUT during test execution. Running a simulation again allows the tester to exchange the Acceptance Criteria because they have no effect on the SUT. Thus, the tester can analyze other signals or adapt the reference signal. Furthermore, the simulator can accelerate the postsimulation because the simulation time does not have to coincide with the physical time. For example, a simulation that originally took 3 hours could finish in 20 seconds.

## 2.2 Execution

The dependencies between the single modules are shown in the information flow in Figure 1 on page 2. Modules are relying on the outputs of other modules in order to perform own calculations on them. For example, the Test Steps have to receive the sensor values from the Plant Model before they are able to pass them to the SUT. Consequently, the simulator has to execute the modules in the following sequence:

1. Read inputs
2. Update Plant Model
3. Execute Test Steps
4. Calculate Acceptance Criteria
5. Apply Supervisor
6. Write outputs

When executing this sequence cyclically the HiL simulator can continuously receive data from and transmit data to the SUT. In addition to this, the intermediate steps 2 - 5 allow to evaluate the behavior of the SUT. This approach was inspired by the cyclic execution model of a PLC [Be11]. In the presented cycle the execution is separated into three parts: reading inputs, executing the program, and writing outputs. We replaced the program with steps 2 - 5. However, it should be noted that the goal of our approach is not to be executed on a PLC. Our goal is to demand as little familiarization time from the tester as possible compared to the programming of PLCs by following the same execution scheme.

## 3 Test Case Specification

The specification of test cases can be completely done in PLC programming languages. Currently, our concept supports the PLC languages Structured Text (ST) and Sequential Function Chart (SFC) [IE13]. The choice of which programming language to use can be individually made for Plant Model, Test Steps, and Acceptance Criteria. For example, the tester can implement the Plant Model in SFC and the Test Steps in ST. When implementing, the tester can make use of the accustomed functions of the programming languages. We

slightly extended the languages to allow for a complete test case specification of Hardware-in-the-Loop-Tests (HiL-Tests). This incorporates the ability to override signals of the Plant Model within the Test Steps (see Section 2.1). For every signal  $Q$  of the Plant Model there is a variable  $OVERRIDE\_Q$  of type Boolean within the Test Steps. When assigned with TRUE, the Test Steps override variable  $Q$  with its own value. Additionally to this, we extended the languages by the specification of signal tolerances which we will describe in the next Section in detail.

## 4 Signal Tolerances

The architecture of the HiL simulator requires the specification of reference signals which are then matched to the actual signals of the SUT. Up to now, we considered a test case to pass if these signal pairs equal. However, this assumption cannot be applied in practice. On the one hand, we have to account that a SUT usually has a response time  $t$  to changed input signals. The reaction of the SUT can be measured at the earliest time  $t + r$ . If the tester does not consider this delay in the Acceptance Criteria, the test fails regardless of the actual reaction time requirements. On the other hand, the time requirements for the SUT are often ranging from a few hundred milliseconds to several seconds. The requirements can therefore be much lower than the measurement resolution of the HiL simulator. In order to allow the tester to capture these cases our approach provides the specification of signal tolerances in time- and value-domain.

### 4.1 Signal Representation

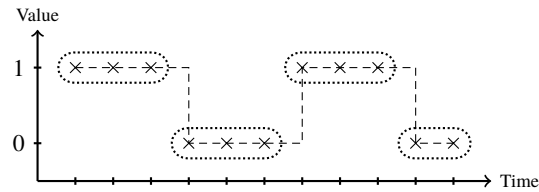


Figure 3: Discretization and grouping of signals.

Firstly, we have to clarify how signals are represented within the HiL simulator. The simulator samples inputs and writes outputs cyclically, resulting in a discrete-time sampling of the input signals. Analogously, the simulator writes the outputs with a fixed time resolution complying with its execution frequency. Signals inside the simulator are digital, so that an analog-to-digital or digital-to-analog conversion is needed when reading inputs or writing outputs. This process has, analog to the time domain, a discrete quantization of the input signals as well as a fixed resolution of the output signals [Ho68]. In summary, signals inside the simulator consist of a set of time and value discrete points, as marked by crosses in Figure 3. The original signal is denoted by a dashed line and describes a signal

generated by Pulse-width modulation (PWM). Reference signals are treated the same way but need some post-processing. The Supervisor creates so-called reference sets, as indicated by dashed ovals in Figure 3. If a sample point has the same value like the sample point in the time step before it is added to its set. As soon as the value changes, the Supervisor closes the set and opens a new one. Once a set is closed, no sample points can be added to it anymore. After the reference sets are created, the Supervisor can carry out the matching with the actual signal. It assigns sample points of the actual signal to the reference sets with the aim of determining a valid mapping.

## 4.2 Definition

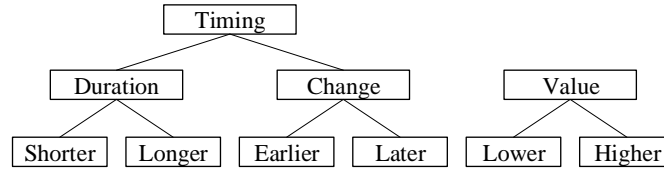


Figure 4: Tolerance Domain

In order to specify a tolerable deviation of two signals, we first have to define quantitative measures. We distinguish between tolerances in time and value domain, as shown in Figure 4. In value domain, we distinguish between a deviation of higher and lower value. In time domain, we have more options: we allow for an earlier or later value change and we allow single segments of the signal to vary in duration. Every reference set defines a signal segment, i.e. a sequence of sample points with constant value. The tester has the opportunity to set these tolerances in his test case for signals to test. The range of signals is not limited to output signals, but also allows virtual signals defined in the Plant Model (see Section 2.1).

## 4.3 Specification in Structured Text

We extended ST to allow for tolerance specifications in the variable declaration of its program code [Th17]. Our syntax allows for an inheriting specification, as given in Figure 4. When setting a tolerance it applies to all of its subtypes, too. We illustrate this in the following example in ST syntax:

```

Timing := T#5s;
Timing.Duration.Longer := T#2s;
Timing.Change.Earlier := T#3s
Value := 10;

```

We extensively use inheritance in this example. In the first step, we set the tolerance in time domain to 5 s so that the duration of signal segments as well as the value change may vary

by 5 s. In the following instruction we change the duration tolerance to allow segments to be up to 5 s shorter but not more than 2 s longer. Line 3 shortens the signal offset to the front to 3 s. Finally, we set the value tolerance to the value 10 which allows signals with a value higher or lower by 10 than the reference signal. There exist test cases requiring the tester to change tolerances during execution. Consider the transient response of a system as an example. While the tester may allow a high value tolerance during the transient period, they want to restrict it to a small band after a certain time. For this reason we introduced the concept of tolerances cases, illustrated in the following example:

```
VAR_OUTPUT
  Q: INT := 1 WITH TOLERANCE
    TOLERANCE_CASE SETTLED = TRUE:
      VALUE := 10;
    END_TOLERANCE_CASE
  TOLERANCE_DEFAULT
    VALUE := 80;
  END_TOLERANCE_DEFAULT
END_TOLERANCE
END_VAR
```

The code defines variable Q of type integer with an initial value of 1. Case TOLERANCE\_DEFAULT defines the default tolerance in value domain to 80. Thus, the tester allows a deviation of  $\pm 80$  during the transient period. In the steady state, the Test Steps (not shown here) set the variable SETTLED to TRUE, resulting in a value tolerance of 10. By following this principle, the tester can vary the tolerances during test execution.

#### 4.4 Specification in Sequential Function Chart

In SFC, the tolerance specification integrates into the specification of the reference signal. The example given in Figure 5 shows the procedure: Initially, the pump is disabled. When

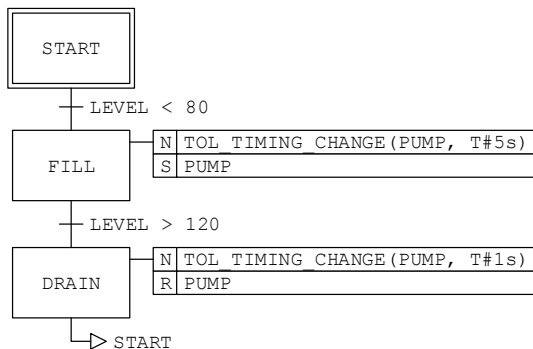


Figure 5: Adjusting tolerances in Sequential Function Chart.



the level falls below 80, the tester wants the pump to activate with a time tolerance of  $\pm 5$  s. The tolerance is applied just when the execution switches to state FILL. If the level exceeds 120, the tester wants the pump to deactivate with a time tolerance of  $\pm 1$  s set by the DRAIN state. This specification aims to imitate the state- and action-based implementation of SFCs by an action-based specification of signal tolerances.

## 5 Application

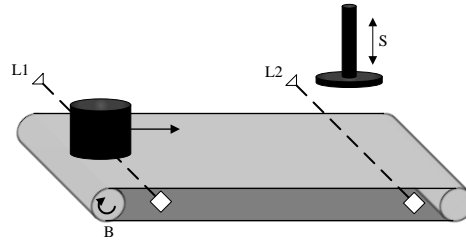


Figure 6: Conveyor belt as a sample application.

This section will discuss the application of the presented approach using a simple, exemplary plant. The plant is shown in Figure 6 and consists of a belt B, a punch S, and two light barriers L1 and L2. As soon as a workpiece interrupts L1, the belt starts to run and stops as soon as it interrupts L2. The punch then operates the workpiece and the belt carries it away. We will now show the implementation of the modules Plant Model and Test Steps in ST and Acceptance Criteria in SFC. The specification of signal tolerances is done in ST. Please consider the shown implementation to be minimal and to make simplified assumptions in order to focus on the understanding of the concept. The Plant Model is implemented as follows:

```

IF B THEN
  POSITION := POSITION + 1;
END_IF
IF POSITION > 5 THEN
  L1 := FALSE;
END_IF
L2 := (POSITION >= 50 && POSITION <= 55);

```

The position is initialized with 0 and stored as an internal variable. When the belt moves, the position is increased in every cycle by 1. As soon as the position exceeds 5, L1 is defined to be non-interrupted because the workpiece is not in its measuring field anymore. L2 is interrupted just when the position is in the range of 50 to 55. This implementation covers the sensor data of the SUT and thus is an adequate plant model. The next step is to implement the Test Steps. In this example, the SUT is stimulated by placing a workpiece on the belt, i.e. interrupting L1. Consequently, the implementation consists of one instruction that sets the sensor value of L1 to TRUE. The last step is the implementation of Acceptance

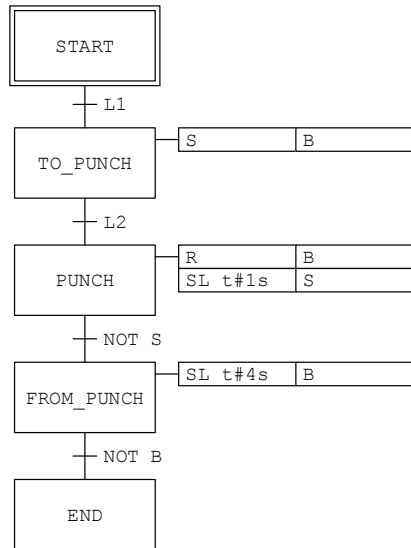


Figure 7: Acceptance Criteria implemented in Sequential Function Chart.

Criteria, subdivided into the specification of the reference signal and the tolerances. Figure 7 shows the implementation of the reference signal in SFC that is completed by the following tolerance specification:

```

VAR_OUTPUT
  B := BOOL WITH TOLERANCE
    TIMING := t#500ms;
  END_TOLERANCE
  S := BOOL WITH TOLERANCE
    TIMING.CHANGE := t#500ms;
    TIMING.DURATION := t#100ms;
  END_TOLERANCE
END_VAR

```

The execution switches to state TO\_PUNCH when the Test Steps stimulate the SUT by interrupting L1. In its action, it defines the belt B to run. The tolerances define that switching the belt on or off may vary by 500 ms. As soon as the Plant Model declares L2 as interrupted, the execution switches to state PUNCH. For operating the punch, the tester requires the belt to stop and trigger the punch for 1 second. Switching the belt and the punch on or off may vary by 500 ms and the duration of punching by 100 ms. After this step, the workpiece has to be carried away by activating the belt for 4 s in state FROM\_PUNCH.

## 6 Conclusion

The presented approach allows for the implementation of test cases by slightly extended PLC programming languages (see Section 3 and 4.2). Made possible by the modular architecture, i.e. the division into Plant Model, Test Steps, and Acceptance Criteria, the tester can adapt parts of the test case efficiently. We aim for an agile testing process by this modularity and the closeness to PLC programming languages. In future work, we will present and evaluate an implementation of the presented concept, based on the Soft-PLC Twistturn [OKK14].

## References

- [Be11] Berger, H: Automatisieren mit STEP 7 in AWL und SCL: Speicherprogrammierbare Steuerungen SIMATIC S7-300/400. Publicis Publ., 2011.
- [BtHVVH14] Bauernhansl, Thomas; ten Hompel, Michael; Vogel-Heuser, Birgit: Industry 4.0 in Produktion, Automatisierung und Logistik. Werkstattstechnik, 103(March):648, 2014.
- [Gu07] Gu, Fangming; Harrison, William S.; Tilbury, Dawn M.; Yuan, Chengyin: Hardware-in-the-loop for manufacturing automation control: Current status and identified needs. In: Proceedings of the 3rd IEEE International Conference on Automation Science and Engineering, IEEE CASE 2007. pp. 1105–1110, 2007.
- [Ho68] Hoeschele, David F: Analog-to-digital/digital-to-analog conversion techniques. 1968.
- [IE11] IEC 26262-4: Road vehicles - Functional safety - Part 4: Product development at the system level. IEC, 2011.
- [IE13] IEC 61131-3: Programmable controllers - Part 3: Programming languages. IEC, 2013.
- [Na16] National Instruments: Hardware-in-the-Loop (HIL) Test System Architectures. 2016.
- [OKK14] Obster, Mathias; Kalkov, Igor; Kowalewski, Stefan: Development and execution of PLC programs on real-time capable mobile devices. In: Emerging Technology and Factory Automation (ETFA). IEEE, pp. 1–8, 2014.
- [Th17] Thönnessen, D.; Reinker, N.; Rakel, S.; Kowalewski, S.: A Concept for PLC Hardware-in-the-loop Testing Using an Extension of Structured Text. In: Emerging Technology and Factory Automation (ETFA). IEEE, pp. 1–8, 2017.
- [TK18] Thönnessen, D.; Kowalewski, S.: Agiles Testen von cyber-physischen Produktionssystemen. atp edition, 2018. Manuscript submitted for publication.
- [Wi17] Wiechowski, Norbert; Rambow, Thomas; Busch, Rainer; Kugler, Alexander; Hansen, Norman; Kowalewski, Stefan: Arttest – a New Test Environment for Model-Based Software Development. In: SAE Technical Paper. SAE International, pp. 1–11, 2017.
- [Zh07] Zhang, Peng; Luk, Wai Shing; Song, Yu; Tong, Jiarong; Tang, Pushan; Zeng, Xuan: WCOMP: Waveform Comparison Tool for Mixed-Signal Validation Regression in Memory Design. Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC, pp. 209–214, 2007.

## Finding Inconsistencies in Design Models and Requirements by Applying the SMARTD Process

Stefan Kriebel<sup>1</sup> Evgeny Kusmenko<sup>2</sup> Bernhard Rumpe<sup>2</sup> Michael von Wenckstern<sup>2</sup> ✉

**Abstract:** The development of safety critical systems requires a highly automated integrated methodology supporting the design, verification and validation of the overall product. Such a methodology maintains the consistency and correctness of all artifacts at all development stages ideally incorporating requirements changes into the corresponding models, tests, and code. This paper shows how the SMARTD process uses formalized SysML diagrams to identify inconsistencies in architectural designs and requirements. An adaptive light system serves as illustrative running example.

### 1 Introduction

Safety critical software systems undergo a complex development process before they are introduced onto the market. The manufacturers are obliged to make their systems ISO26262 compliant and to guarantee the correctness of their implementation. Usually such systems become very large and have to fulfill hundreds of intertwined requirements developed by different teams. Therefore, an urgent need for automated means of consistency checking of requirement specifications identifying errors in early design stages has been arising.

At BMW, one of Germany's largest automotive companies, the new SMARTD process tries to tackle this problem. One of its goals is to ensure artifact consistency for the different phases of the development process by using model-based software engineering (MBSE), particularly formalized SysML diagrams.

This paper shows how the SMARTD process uses formalized SysML diagrams to **identify inconsistencies in design decisions and requirements**. Thereby, we focus on artifacts of the logical layer of the SMARTD process and demonstrate the idea on the requirement specification of a real-world Adaptive Light System (ALS).

Section 3 recaps the main ideas of the SMARTD process. Section 4 shows how Component & Connector (C&C) views reveal structural design inconsistencies derived from different

---

<sup>1</sup> BMW Group, Munich, Germany; stefan.kriebel@bmw.de

<sup>2</sup> RWTH Aachen University, Software Engineering, Ahornstraße 55, 52074 Aachen, Germany;  
{kusmenko, rumpe, vonwenckstern}@se-rwth.de

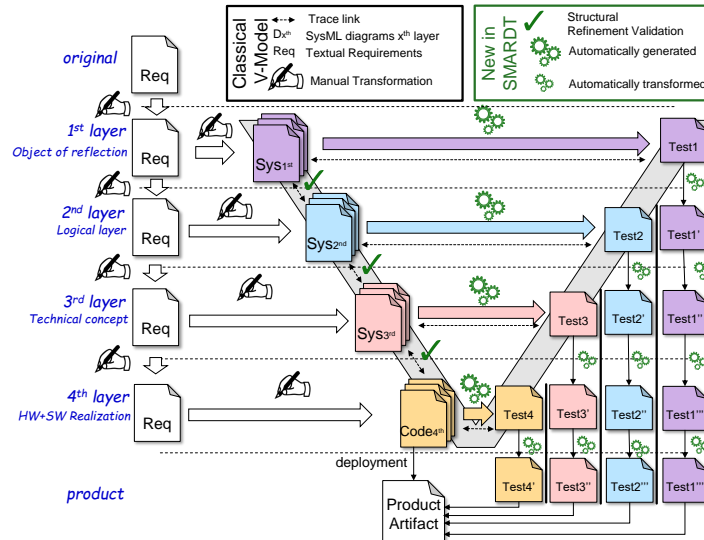


Fig. 1: Overview of the SMARTDT methodology (copied from [Hi18]).

requirement specifications. Section 5 presents ideas how formalized Activity Diagrams (ADs) help to detect behavioral inconsistencies in requirements.

## 2 Running Example

To demonstrate the feasibility and the advantages of the proposed methodology we use seven real-world requirement specifications of an ALS [Be17]. The ALS model controls adaptive high and low beam, turn signals as well as cornering and ambient light. Adaptive high and low beam adjust headlamps to the traffic situation and provide optimized illumination without dazzling others. Cornering light illuminates the area to the side of a vehicle to take a look around the bend. Ambient light welcomes the driver with an indirect light. For inspection purposes all 93 original requirements of the ALS are available at our homepage<sup>3</sup>.

## 3 The SMARTDT Methodology

The SMARTDT (*Specification Methodology Applicable to Requirements, Design, and Testing*<sup>4</sup>) approach [Hi18] extends the German V-Model [BD95], which is the official project management methodology of the German government. The SMARTDT process

<sup>3</sup> <http://www.se-rwth.de/materials/cncviewscasestudy/>

<sup>4</sup> The original abbreviation SMArDT is related to the German term "Spezifikations-Methode für Anforderung, Design und Test"

delivers reliable specification methodologies by introducing formalized SysML diagrams [OM15] with a well-defined semantics [HR04] to specify the functionality of automotive systems. This allows to automatically ensure a permanent consistency between all abstraction layers of the V-Model. This task can become particularly tedious if maintained by hand in agile development processes as those are mostly iterative, incremental, and evolutionary [Be01]. New validations enabled by SMARTD process are: (1) backward compatibility checks [Ru15, Ri16, Be16, Ku18] for software maintenance and evolution between different diagram versions of the same layer, (2) refinement checks [Ru96, HRvW17] between diagrams of different layers allowing to detect specification inconsistencies between different layers, as well as (3) advanced structural or extra-functional property [Ma16] checks on SysML diagrams using OCL [Ma17].

The key principles of SMARTD for a formal specification of requirements, design, and testing of system engineering artifacts according to ISO 26262 are illustrated in Figure 1. The methodology is structured in four layers: (1) The object of reflection layer contains a first description of the object under consideration and shows its boundaries from a customer's point of view. (2) The logical layer containing functional specifications without details of their technical realizations. (3) The technical concept, e.g. C code or Simulink models, belongs to the third layer. Finally, the fourth layer represents the software and hardware artifacts present in the system's implementation.

In SMARTD consistency between different layers is ensured by verification and model-based testing of the final product against the requirements of all layers. More specifically, SMARTD enables structural verification as explained in Section 4 between each layer indicated by the green check marks in Figure 1. Furthermore, SMARTD enables a systematic and fully automatic derivation of test cases for each layer as discussed in [Hi18] and illustrated on the right side of Figure 1. The previous SMARTD paper [Hi18] showed how this methodology is applied to develop a self-driving racing vehicle; its main focus was the use of formalized ADs enriched by OCL constraints for automated test case derivation. In contrast, this paper focuses on detecting structural and behavioral inconsistencies between different artifacts in the logical layer of the SMARTD process.

## 4 Architecture Specification using Views

Embedded software systems are often created in Simulink as C&C models describing functional, logical or software architectures [TMD09] in terms of components executing computations and connectors effecting component interaction via typed and directed ports. An advantage of this approach is that complex components such as ALS can be hierarchically decomposed into other smaller components to be developed by different teams. Interaction between components occur only via connectors. SysML and Modelica are two other famous representatives for C&C modeling languages.

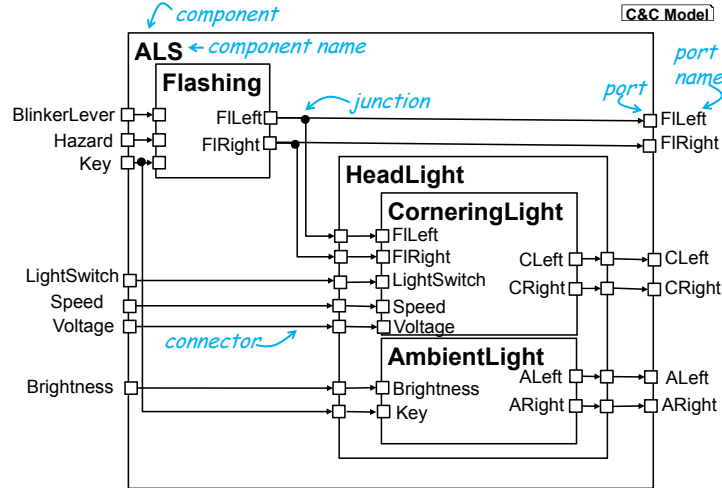


Fig. 2: Very simple example C&amp;C model for ALS.

C&C views, as presented in [MRR13], are developed to focus on important view points (excerpts) of large C&C models without being required to model all the other (for the view point unimportant) information. For example a view can only show how a component is decomposed into subcomponents without showing any ports and connections of the subcomponents. The main aim of C&C views concept is to have many but therefore small, precise and good-readable view points of one large C&C model. For this reason C&C views introduce four major abstractions: hierarchy, connectivity, data flow, and interfaces of C&C models. The hierarchy of components in C&C views is not necessarily direct (intermediate components may be skipped); abstract connectors can cross-cut component boundaries and they can also connect components directly (if the concrete connected port is not important for this view); abstract effectors describe data flow abstracting over chains of components and connectors, and C&C views do not require complete interfaces with port names, types and array sizes. Intuitively, a C&C model satisfies a C&C view iff all elements and relations shown by the view have a satisfying concretization in the model. The formal definitions of C&C model, C&C view, and their satisfaction are available in [MRR13] and from supporting materials website of paper [Be17].

**C&C View Verification and Witnesses for Tracing** Figure 2 shows the example C&C model of an adaptive light system (ALS). This simplified model controls the flash LEDs for turning as well as the left and right light bulbs for cornering and ambient lights. The ALS consists of the two subcomponents *Flashing* and *HeadLight*. Component *HeadLight* is hierarchically decomposed into two further components: *CorneringLight*, and *AmbientLight*. The C&C view *ALS1* shown in Figure 3 describes the *CorneringLight* component illuminating on intersections the road where the driver wants to turn into. Thus, the input port *BlinkerLever*

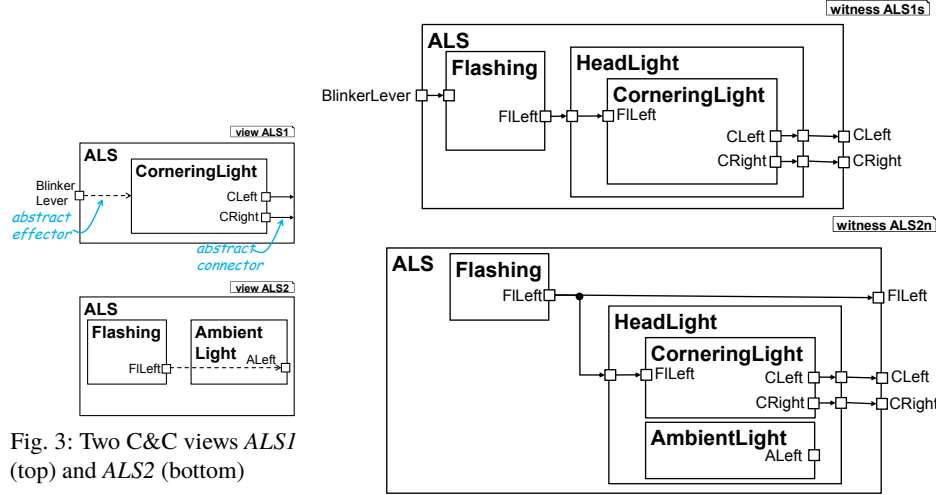


Fig. 3: Two C&C views *ALS1* (top) and *ALS2* (bottom)

C&C Model is missing an effector from the component *Flashing* to the component *AmbientLight* (from port *FLeft* to port *ALeft*)

Fig. 4: Witness for satisfaction *ALS1s* (top) and witness for non-satisfaction *ALS2n* (bottom)

of the ALS component has effect (modeled by an abstract effector) to at least one input port of the *CorneringLight* component. The calculated light values (*CLeft*, and *CRight*) of the *CorneringLight* component are passed directly (without being modified anymore) to the output ports of the ALS component (modeled by two abstract connectors). The C&C view *ALS2* is about the relation between the *Flashing* and the *AmbientLight* component. It specifies that the *FLeft* (flashing left) output of components *Flashing* effects the ambient left light (port *ALeft* of component *AmbientLight*); e.g. brighter left ambient light when the left parking mode is activated. The model *ALS* satisfies the view *ALS1*.

C&C view verification, as presented in [MRR14], gets as input a C&C model and a C&C view. Besides a Boolean answer whether the C&C model satisfies the C&C view, the verification algorithm produces a local minimal satisfaction or one or more local non-satisfaction witnesses. As an example, a witness for satisfaction *ALS1s* is shown in Figure 4 and demonstrates how the C&C model satisfies *ALS1*. The SMARTD approach uses the generated witnesses to automatically generate traceability information between SysML artifacts of layer 2 (C&C views) against the large SysML model of layer 3 (C&C models). The witness is itself a well-formed model. The witness contains all view's components (here ALS, and *CorneringLight*) as well as their parent components to show the complete hierarchy between two components specified in the view. Therefore, the witness contains also the *HeadLight* component. The *positive satisfaction witness* also contains all ports corresponding to a view, therefore the witness contains *BlinkerLever* port of ALS as well as *CLeft*, and *CRight* ports of *CorneringLight*. Additionally the witness contains all model connectors and



all data-flow paths. The abstract connector from `CorneringLight` (port `CLeft`) to `ALS` (port *unknown*) introduces the following elements in the witness: (1) port `CLeft` of component `HeadLight`; (2) connector of ports `CLeft` from component `CorneringLight` to component `HeadLight`; and (3) connector of ports `CLeft` from component `HeadLight` to component `ALS`. For the abstract effector from `ALS` (port `BlinkerLever`) to `CorneringLight` the following elements in the chain serve as witness: (1) component `Flashing`; (2) ports `BlinkerLever` and `FLeft` of `Flashing`; (3) connector of ports `BlinkerLever` from `ALS` to `Flashing`; (4) connector of ports `FLeft` from `Flashing` to `HeadLight`; and (5) connector of ports `FLeft` from `HeadLight` to `CorneringLight`.

The model `ALS` does not satisfy the view `ALS2`. Every *negative non-satisfaction witness* contains a minimal subset of the C&C model and a natural-language text, which together explain the reason for non-satisfaction. These witnesses are divided into five categories: `MissingComponent`, `HierarchyMismatch`, `InterfaceMismatch`, `MissingConnection`, `MissingEffector` (see [MRR14]). A witness for non-satisfaction `ALS2n` (case `MissingEffector`) is shown in Figure 4. It shows all outgoing connector-effector chains starting at port `FLeft` of component `Flashing` as well as the abstract effector’s target port, `AmbientLight`’s `ALeft`, which is not reachable. Removing the effectors in the view `ALS2` would cause the model to satisfy this modified view even though `Flashing` and `AmbientLight` are direct siblings in the C&C view and are not direct siblings in the C&C model; C&C views allow to abstract away the intermediate component `HeadLight`.

**Identifying Design Inconsistencies with C&C Views** The previous paragraphs showed how C&C views verification can be used to check structural consistencies and how to generate tracing witnesses between artifacts of layer B (view models) against artifact of layer C (concrete logical C&C models). This part focuses on identifying structural design inconsistencies between different artifact models of layer B. Figure 5 shows two requirements about the *cornering light*. Since the last requirement **AL-139** is a safety feature for armored vehicles, a special team is responsible for it. Thus, the architectural designs (view `AL-122`, and view `AL-139`) for the two requirements are developed by two different teams. The top design shows the `CorneringLight` with two modes (subcomponents `Cornering_Active`, `Cornering_Deactive`, and `MultiSwitch`), whereby the mode `Cornering_Deactive` is selected if the voltage is too low. In the bottom design model the `min` block in combination with the `Switch` one deactivates indirectly the cornering light by propagating 0% as light value to `OvervoltageProtection`’s input ports when the `DarknessSwitch` port has the value `true`. C&C view synthesis [MRR13] is the process in deriving one valid C&C model which satisfies all given C&C views. The first part of this algorithm checks whether views contain design contradictions. When all views as in our example are positive views (only expressing what should be present; and do not contain negations such as component A should not contain port B), then the contradiction check can be done in polynomial time and scales very well to many hundreds views.

The contradiction check for the both views `view AL-122` and `view AL-139` would result in an

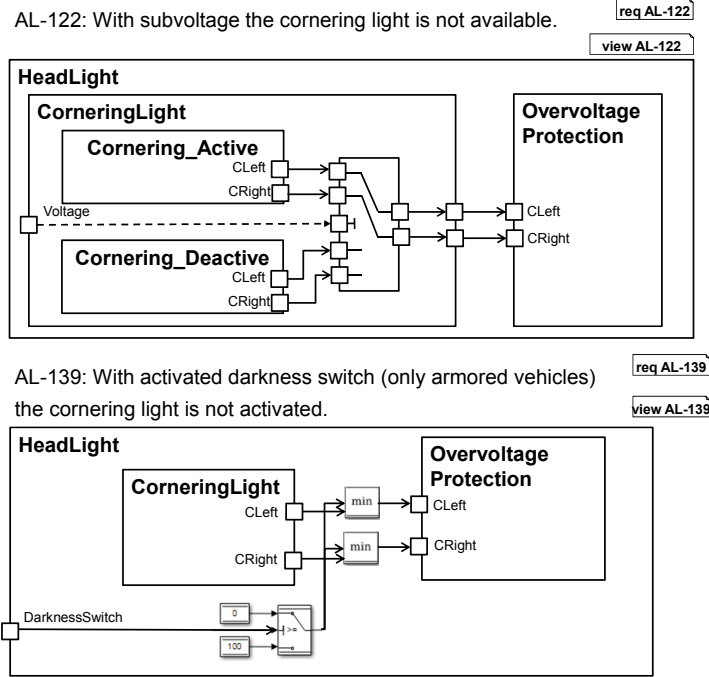


Fig. 5: Design Inconsistency of two C&amp;C views

error as in the top view the port `CLeft` of component `CorneringLight` is directly connected (without modifying the value) to the component `OvervoltageProtection` whereas in the bottom view the value from `CorneringLight`'s `CLeft` is manipulated by the `min` component before it goes to the `OvervoltageProtection`'s `CLeft` port. Similar to the C&C view verification process presented above, the contradiction algorithm generates an intuitive witness to highlight incompatible parts of two views. The formalized C&C view verification problem with its derived contradiction problem enables early analysis of structural design models in the SMARTD process to detect as early as possible inconsistencies between different artifacts created by different persons or teams, to avoid problems when integrating at a later time step different software modules developed on inconsistent designs.

## 5 Formalized Activity Diagrams

In [Hi18] we showed how formalized ADs can be used in combination with OCL constraints in order to generate test cases. In this paper, we demonstrate how formalized ADs can help finding inconsistencies in requirement specification. Consider the Figures 6 and 7 illustrating two ADs describing the steering column stalk and the hazard lights behavior based on the requirements *AL-40* and *AL-41* from [Be17], respectively. In Figure 6 the

**AL-40 Direction indicator (left):** When the steering column stalk is moved into the position blinking (left) all left direction indicators (rear left, exterior mirrors left, front left) start blinking synchronously with an on/off ration of 1:1. req AL-40

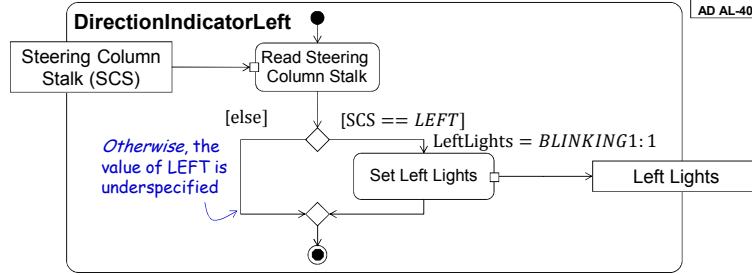


Fig. 6: Activity diagram for AL-40 describing the steering column stick behavior.

**AL-41 Hazard lights:** As long as the hazard light switch is pressed, all direction indicator lamps blink synchronously. If the ignition key is inside the lock, the on/off ration is 1:1. Otherwise, the on/off ratio is 1:2. req AL-41

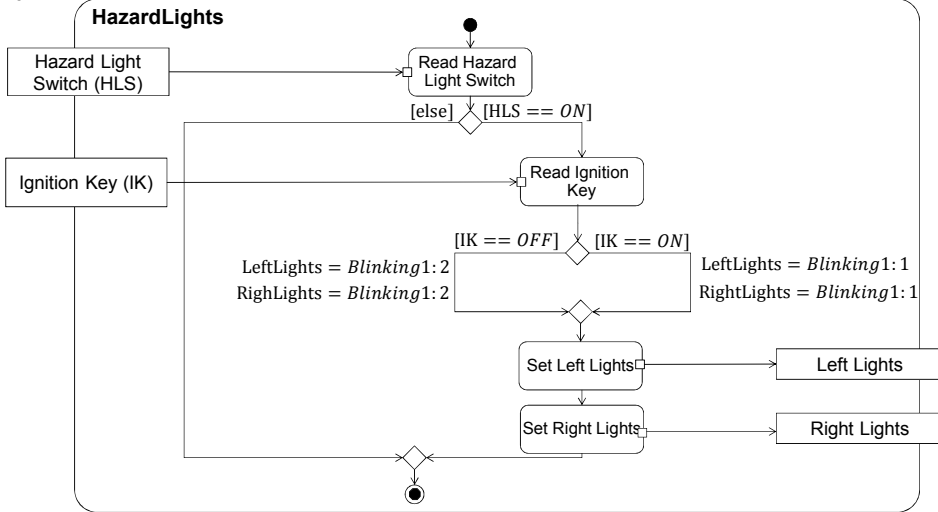


Fig. 7: Activity diagram for AL-41 describing the emergency lights behavior.

model would receive the state of the steering column stalk through the corresponding input port. If the position of the steering column stalk is set to left, the left lights of the car are set to the blinking mode with an on/off ratio of 1:1, denoted as `BLINKING1:1`. This value is written to the output port of the AD. The *else* branch is underspecified, i.e., nothing is said about the output for the cases if the steering column stalk is set to right or to straight. Note, that this is in accordance with the underlying requirement.

In the AD of Figure 7, the hazard light switch is read in order to decide whether to turn

the lights on or not. However, to determine the concrete blinking mode, the position of the ignition key is required. If the key is inserted, the 1:1 blinking mode is activated, otherwise the 1:2 mode is used to save battery power. The *else* branch is underspecified again. Note, that according to the requirement the same physical lights are used for hazard blinking as for direction indication. This leads to a contradiction which becomes obvious in the two ADs: if the key is not inside the lock and the hazard light switch is pressed, according to **AL-41**, the vehicle's indicators lamps should blink with a ratio of 1:2. Now imagine that at the same time the steering column stalk is set to *blinking (left)*. According to **AL-40** the blinking ratio of the left direction indicators should be 1:1. Considering that a faster blinking drains the battery in a shorter amount of time, this specification error might even become critical for human lives in cases of emergency. The error is discovered automatically by creating a table mapping the three input signals to the specified lamp light modes and filling it with all *specified* combinations. For Figure 6 we would obtain one single combination, namely:  $(SCS = LEFT, HLS = DC, IK = DC) \rightarrow (LeftLights = BLINKING1 : 1, RightLights = DC)$  where *DC* stands for "don't care". Figure 7 produces two combinations one of which is  $(SCS = DC, HLS = ON, IK = OFF) \rightarrow (LeftLights = BLINKING1 : 2, RightLights = BLINKING1 : 2)$ . Expanding the *DC* fields to all possible values reveals that the specification requires two contradictory outputs for the same input.

## 6 Conclusion

In this paper we discussed the need for automated consistency checks for requirement and design artifacts of safety critical systems. Therefore, we developed a methodology supporting the SMARTD process - a formalized version of the widely used V-Model - by identifying wrong or contradicting requirements at early development stages using C&C views and activity diagrams. The methodology was demonstrated on a real-world ALS requirement specification. Using the proposed concepts we were able to detect structural and behavioral inconsistencies on the logical layer of the SMARTD process, i.e., long before the actual implementation would be created. Future work comprises analysis of further formalized diagrams and their respective combinations as well as detailed case studies of the complete SMARTD process and the proposed tooling.

**Acknowledgements** This research was supported by a Grant from the GIF, the German-Israeli Foundation for Scientific Research and Development, and by the Grant SPP1835 from DFG, the German Research Foundation.

## References

- [BD95] Brühl, Adolf-Peter; Dröschel, Wolfgang: Das V-Modell. München, Wien: Oldenburg-Verlag, 1995.
- [Be01] Beck, Kent; Beedle, Mike; Van Bennekum, Arie; Cockburn, Alistair; Cunningham, Ward; Fowler, Martin; Grenning, James; Highsmith, Jim; Hunt, Andrew; Jeffries, Ron et al.: Manifesto for agile software development. 2001.

- [Be16] Bertram, Vincent; Roth, Alexander; Rumpe, Bernhard; von Wenckstern, Michael: Extendable Toolchain for Automatic Compatibility Checks. In: OCL'16. 2016.
- [Be17] Bertram, Vincent; Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard; von Wenckstern, Michael: Case Study on Structural Views for Component and Connector Models. In: MODELS. 2017.
- [Hi18] Hillemacher, Steffen; Kriebel, Stefan; Kusmenko, Evgeny; Lorang, Mike; Rumpe, Bernhard; Sema, Albi; Strobl, Georg; von Wenckstern, Michael: Model-Based Development of Self-Adaptive Autonomous Vehicles using the SMARDT Methodology. In: MODEL-SWARD'18. 2018.
- [HR04] Harel, David; Rumpe, Bernhard: Meaningful Modeling: What's the Semantics of "Semantics"? IEEE Computer, 37(10), 2004.
- [HRvW17] Heithoff, Malte; Rumpe, Bernhard; von Wenckstern, Michael: Anforderungsverifikation von Komponenten- und Konnektormodellen am Beispiel Autonom Fahren Autos. GI Softwaretechnik-Trends, 37(2), 2017.
- [Ku18] Kusmenko, Evgeny; Shumeiko, Igor; Rumpe, Bernhard; von Wenckstern, Michael: Fast Simulation Preorder Algorithm. In: MODELWARD. 2018.
- [Ma16] Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard; Wenckstern, Michael von: Consistent Extra-Functional Properties Tagging for Component and Connector Models. In: ModComp. 2016.
- [Ma17] Maoz, Shahar; Mehlan, Ferdinand; Ringert, Jan Oliver; Rumpe, Bernhard; von Wenckstern, Michael: OCL Framework to Verify Extra-Functional Properties in Component and Connector Models. In: ModComp. 2017.
- [MRR13] Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard: Synthesis of component and connector models from crosscutting structural views. In: FSE. 2013.
- [MRR14] Maoz, Shahar; Ringert, Jan Oliver; Rumpe, Bernhard: Verifying component and connector models against crosscutting structural views. In: ICSE. 2014.
- [OM15] OMG: OMG Systems Modeling Language (OMG SysML). Technical Report Version 1.4, 2015.
- [Ri16] Richenhagen, Johannes; Rumpe, Bernhard; Schloßer, Axel; Schulze, Christoph; Thissen, Kevin; von Wenckstern, Michael: Test-driven Semantical Similarity Analysis for Software Product Line Extraction. In: SPLC. 2016.
- [Ru96] Rumpe, Bernhard: Formale Methodik des Entwurfs verteilter objektorientierter Systeme. Herbert Utz Verlag Wissenschaft, 1996.
- [Ru15] Rumpe, Bernhard; Schulze, Christoph; von Wenckstern, Michael; Ringert, Jan Oliver; Manhart, Peter: Behavioral Compatibility of Simulink Models for Product Line Maintenance and Evolution. In: SPLC. 2015.
- [TMD09] Taylor, Richard N.; Medvidovic, Nenad; Dashofy, Eric: Software Architecture: Foundations, Theory, and Practice. Wiley, 2009.

## Applying DSE for Solving the Deployment Problem in Industry 4.0

Tarik Terzimehic,<sup>1</sup> Sebastian Voss,<sup>1</sup> Monika Wenger,<sup>1</sup> Vincent Aravantinos<sup>1</sup>

### Abstract:

The fourth industrial revolution requires a dynamically reconfigurable control software, in order to cope with frequent product and production process changes of manufacturing systems. To dynamically reconfigure control software, we have to calculate valid or optimal deployment configurations. Previous research applies Design Space Exploration (DSE) techniques embedded into model-based design methodologies for solving this deployment problem. However, current research either aims at domains other than industrial automation or applies simple and, for real-life problems, not applicable constraints and objectives. Thus, the deployment of software components to hardware components is still an exhausting and manual task. In this work, we take first steps towards automatically optimized deployment of industrial automation systems. We propose the concept of DSE for solving the deployment problems of IEC 61499 based control software applications. In order to reduce the exploration space, we identify the industry 4.0 specific constraints and objectives. Furthermore, we extend the IEC 61499 system and application models' descriptions by proposing relevant hardware and software annotations. The presented concept could be used in conjunction with different optimization and satisfiability solving algorithms, such as SMT or linear programming.

**Keywords:** Design Space Exploration; Model-based Development; Deployment; IEC 61499; Industry 4.0

## 1 Introduction

Frequent product and production process' changes in current manufacturing systems make predetermined processes and system configurations infeasible [Sp16]. Thus, dynamic reconfigurability and flexibility of control software are named as key prerequisites towards the fourth industrial revolution, referred to as industry 4.0 [Wa16]. In order to reconfigure the control software consisting of software components (SWCs), a proper configuration has to be generated. The configuration can comprise a particular deployment (or mapping) of SWCs, i.e. deployable software units, to hardware components (HWCs), e.g. Programmable Logic Controllers (PLCs), further referred to as solely *deployment*. A valid deployment satisfies certain constraints, such as maximal hardware cost, energy consumption etc. Since Industry 4.0 requires an optimal manufacturing process [Bu17], we may strive to generate a

---

<sup>1</sup> fortiss GmbH, Model-based systems engineering, Guerickestraße 25, Munich, Germany, {terzimehic,voss,wenger,aravantinos}@fortiss.org

deployment that is optimal with respect to the given objectives (e.g. calculate a deployment configuration that leads to the minimal energy consumption). Additionally, a scheduler (e.g. the execution order of the SWCs or temporal order of communication [BV13]) can be generated as part of the configuration.

Due to the increased complexity of current manufacturing systems (i.e. the growing number of SWCs and HWCs, their properties and specifications), it is difficult to manually find a valid or an optimal deployment configuration [Ed17]. When manual design decisions are difficult due to a large exploration space, design space exploration (DSE) techniques are applied. DSE defines the process of finding a solution (or set of solutions) within a constrained solution space [Sc10].

DSE can be used in conjunction with model-based development (MbD), which provides a system abstraction with suitable models. Different models describe the system from different perspectives. For example, a logical or software model describes the functionality of the system, whereas a technical or hardware model describes the platform/hardware specific aspects of the system. Thereby the system's functionality can be developed independently from the hardware specification, which is an important advantage of this approach especially for complex systems [Ar15].

Numerous research studies apply DSE techniques embedded into MbD methodologies with the aim to find appropriate configurations, i.e. valid or optimal deployments and schedulers [Ed17, ZKC16, AB17]. For example, [Ed17] uses the Satisfiability modulo theories (SMT) solver to explore the design space. However, it does not fit industrial control software, as (1) it applies automotive-specific constraints, objectives and annotations and (2) the used model-based tool is based on the FOCUS theory [BS01], which is not common in industrial control software and therefore probably unfamiliar for process engineers and technicians.

Common industrial control software applications are based on the IEC 61131-3 standard for PLCs, developed with the main goal to unify the way of programming PLCs. Moreover, the standard introduced function blocks (FBs) to improve the quality, modularity, and reusability of control applications. However, the usage of global variables links the FBs via hidden interfaces. The IEC 61499 standard, which represents an extension of the IEC 61131-3 standard, also uses FB as the main element of function encapsulation, but completely forbids the usage of global data. Therefore, the FBs are decoupled from its environment and can be used and tested separately, thereby greatly increasing the reusability and reconfigurability of IEC 61499-based control applications. Furthermore, the IEC 61499 defines basic reconfiguration services, which provide the possibility to reconfigure the application during runtime [Le11].

In [SJC14], a SMT solver is applied to generate valid reconfigurations of IEC 61499-based control systems. This work describes the control software architectures and the high-level requirements with logical formulae. The logical formulae are used as input for a SMT solver. The SMT solver calculates deployment configurations for deploying FBs multiple

times across HWCs according to the given high-level requirements. Nevertheless, tackling real-life problems would require identifying more complex constraints and considering the optimization problem, i.e. finding objectives for the industrial automation domain. Additionally, the specifics of IEC 61499-based systems should be considered more thoroughly, such as relations between devices and resources, inter-device communication and configurations overheads etc.

In this work, we take first steps towards an automatically optimized deployment of industrial automation systems. Thereby, we explore the specifics and needs of the industrial automation domain while focusing on IEC 61499-based systems, as they provide, among others, means for dynamic redeployment. In particular, we identify software and hardware annotations, as well as constraints and objectives that can be used to specify desired deployment configuration. Moreover, we exhibit the applicability of DSE with the identified annotations, constraints and objectives on an industry 4.0 relevant case study.

## **2 Model-based Development with IEC 61499**

In this section we describe the IEC 61499 models, that we extend by deployment relevant annotations, constraint and objectives. The IEC 61499 standard for distributed industrial control systems proposes the usage of different models in order to achieve software and hardware abstractions.

### **2.1 Software Abstraction with the IEC 61499 Application model**

IEC 61499 applies an application centric design, where the functionality of the system is defined within the first step. This is done within the Application model, which is independent of the hardware configuration. The main element of the Application model is a FB. A FB represents a software component that encapsulates functionality and interacts with other components via data and event interfaces. Thus, the Application model is created by instantiating and interconnecting FBs [ZV09].

The IEC 61499 FBs are event driven, which is one of the major differences to other languages in the domain of industrial automation. Thus, FBs require input events (i.e. red connections delivered to the top left interface part of the FB, see upper part of Figure 1) to start processing input data (i.e. blue connections delivered to the bottom left interface part of the FB) [We16].

### **2.2 Hardware Abstraction with the IEC 61499 System & Distribution models**

The System model is used in the second step to specify the hardware configuration, i.e. the control devices and communication systems (bottom of Figure 1). It comprises devices



and the communication infrastructure. A device (e.g. a PLC) can contain several resources that represent an independent execution environments for FB networks. They may provide specific functionality (e.g. special computation hardware), but may as well solely be used for the logical separation of the device onto smaller independent entities [ZV09].

Finally, the functionality from the Application model is mapped to the System model, which results in the Distribution model (illustrated by arrows in Figure 1). In this step the application's FBs will be deployed to the control devices where they are supposed to be executed.

### 2.3 Dynamic Redeployment with the IEC 61499 Management Model

To create and modify an IEC 61499 FB network as flexible as possible, the standard offers a management interface with special management commands. These commands can be run offline or online in order to, e.g., create or delete particular FB instances or their connections, start or stop the FB instances (or whole applications), as well as query the data values or FB attributes [SC05]. In this way the IEC 61499 facilitates the mechanism for deploying or dynamically redeploying control software applications.

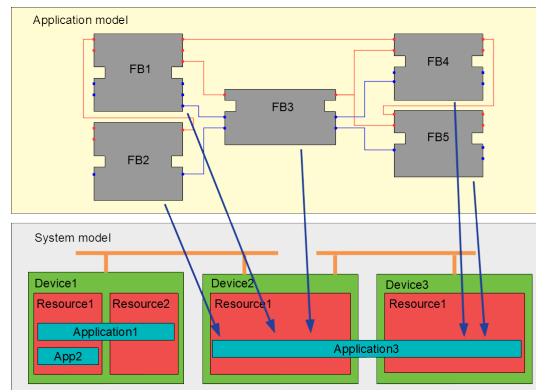


Fig. 1: IEC 61499 Application, System and Distribution models

## 3 Applying DSE to Resolve the Deployment Problem in Industry 4.0

Deployment in the industrial automation domain is still a manual and exhausting task of experienced automation engineers. The widely used IEC 61131 control applications are developed in close conjunction with hardware platforms, i.e. used HWCs, and thus, mapping considerations are not necessary. The IEC 61499 standard provides means to develop SWCs independently from the hardware platform and then manually map particular SWCs (i.e. FB networks) to the desired HWCs, which results in the Distribution model. While a manual

Tab. 1: Constraints, objectives and annotations for IEC 61499-based systems

Constraint	Objective	SW Annotation	HW Annotation
<b>Cost</b> limit costs of HWCs	Min. costs	-	Cost
<b>Number of HWCs</b> limit number of used HWCs	Min. number of used HWCs	-	-
<b>HW capabilities</b> allocate SWCs to required HWCs	-	Required HW capabilities	Available HW capabilities
<b>RTE capabilities</b> allocate SWCs to compatible RTEs	-	Required FB types	Available FB library
<b>Parallel execution</b> execute SWCs parallel	-	Parallel SWCs	Max. number of resources
<b>Functional coupling</b> group coupled SWCs	Max. cohesion, min. coupling	Coupled SWCs	-
<b>Memory</b> prevent memory exceeding	Min. memory usage pro HWC	Memory size	Available memory
<b>SIL</b> allocate SWCs to reliable HWCs	Min. total SIL	Safety level	Safety level
<b>Redundancy</b> allocate SWCs to several HWCs	-	Number of copies	-

deployment is acceptable for a small number of components, constraints and objectives, systems that are more complex require this process to be optimally automated. In order to do so, an automation engineer has to be able to specify the deployment problem.

We specify the deployment problem through constraints that limit the set of possible allocations due to different attributes. Moreover, we can express optimization demands, i.e. objectives, that optimize a limited set of solutions towards certain directions. Table 1 presents the constraints, objectives and corresponding annotations, applicable for IEC 61499 based systems.

**Cost constraint and objective** The economical aspect is crucial for the construction and operation of automation plants. For that purpose we require a hardware cost annotation, which is also often applied in the automotive domain [ZKC16]. For example, the automation engineer can specify the cost of HWCs (e.g. PLCs) while modeling the hardware architecture (e.g. within System model of IEC 61499 architecture). Subsequently the engineer can specify the constraint, so that the total cost of whole applied hardware infrastructure does not exceed a certain maximal value. We can also strive to minimize the whole cost, in which case we define an optimization objective.

**Number of devices constraint and objective** Due to economical (i.e. cost reduction) or/and technical aspects (i.e. easier maintenance, decreased latency as result of increased

intra-node communication etc.) we can limit or minimize the number of used HWCs, while calculating valid or optimal deployment configurations, respectively.

**HW capabilities constraint** Some SWCs or applications require certain hardware capabilities, such as special computation hardware, access to particular sensors or actuators, communication interfaces etc. Thus, an engineer should be able to either (1) manually allocate particular SWCs to particular HWCs and calculate the mapping of remaining components according to other specified constraints and/or objectives (as it is done in [Ed17]), or (2) specify the required and offered capabilities within the Application and System models, respectively. For example, a SWC (i.e. FB network in the IEC 61499-based systems) requires access and a HWC (i.e. device in the IEC 61499-based systems) provides access to the light barrier sensor within a conveyor. By applying this constraint, we assure deployment of the SWC to the required HWC.

**RTE capabilities constraint** If we want to dynamically (re)deploy applications, we need to consider the run-time environment (RTE) running on the HWC. Questionable is whether the RTE of the HWC we want to deploy the SWC(s) to, contains all necessary FB types, since only available FB types can be instantiated. Thus, it should be possible to describe the capabilities of the RTE, i.e. to specify its FB library. The required FB types are already specified within the Application model. Hence, only HWCs whose RTEs contain all necessary FB types can be considered as deployment targets.

**Parallel execution constraint** A device (i.e. HWC) in a System model can contain resources, which can be regarded as independent execution containers for FB applications. Resources provide a logical separation of the device onto smaller independent entities. By deploying applications onto multiple resources, we can achieve a quasi parallel execution<sup>2</sup>. However, a device cannot contain an arbitrary number of resources without jeopardizing its computational and memory performances. Thus, the annotation of the maximal number of resources within one device should be possible. On the other hand, the Application model should provide mechanisms to denote parallel FB networks, i.e. the parts of the applications that are supposed to run concurrently.

**Functional coupling constraint and objective** Similarly, we may want to couple some applications, so that they are executing within the same device, or even within the same resource. Motivation for that could be the lower latency between applications, i.e. increasing the intra-device and decreasing inter-device communications. Thus, we can define the functional coupling constraint by selecting the FB networks supposed to be executed within the same device. Alternatively, we can specify an objective, in order to maximize cohesion and minimize coupling, i.e. maximize the intra-device and minimize inter-device communications.

**Memory constraint and objective** The annotation of hardware's memory capacity, software's memory size and the constraint, that prohibits exceeding the hardware memory

---

<sup>2</sup> quasi parallel since resources can only be processed in parallel if they run on different cores of the CPU

limits, is well known in the automotive domain [ZKC16]. Such memory constraint could be useful in the IEC 61499 systems, whereas the Application model's size could be calculated automatically, according to the used FB instances. If the goal is to dynamically add new components to the device, we can optimize memory usage by applying memory objectives.

**SIL constraint and objective** Safety is the most important process plant characteristic [Wa17]. One way to increase the safety is to deploy SWCs annotated with certain SIL only to HWCs of the same or higher SIL [VS13], i.e. to HWCs that are reliable enough. Thereby, by applying the SIL constraint, safety-critical components are equipped with reliable software and hardware.

**Redundancy constraint** Sometimes it is required that particular software components are deployed multiple times across hardware resources, in order to improve the reliability [SJC14]. Therefore, we introduce the redundancy annotation for IEC 61499 FB networks and define the redundancy constraint, with the aim to deploy particular FB networks to the several HWCs.

## 4 Evaluation

This section provides a domain-specific case study to exhibit the DSE applicability for solving the deployment problem.

### 4.1 Cold Rolling Mill Demonstrator

With the aim to demonstrate the DSE application for calculation of deployment configurations, we consider the aluminum cold rolling mill demonstrator as an example. The aluminium cold rolling mill consists of two main parts, the rolling process and the pallet transportation system (PTS). For the sake of simplicity, this case study focuses solely on the PTS. The PTS transports pallets with coils to different working steps of the plant. The PTS is composed of several roller conveyors, slider conveyors and turn tables. The roller conveyors are supposed to perform the entire transportation task. Each roller conveyor has four light barriers to detect the pallet position. The pallet position is used to adapt the transportation speed and detect the movement direction. The slider conveyors move pallets between different rows of roller conveyors. The turn tables rotate the coil to change the winding orientation of the aluminium band on the coil.

### 4.2 Case Study: Failure Recovery Without Downtimes

Availability, i.e. continuous operation of the plant without downtime, is named as one of the most important characteristics of rolling mills [Wa17]. Hence, we examined the applicability

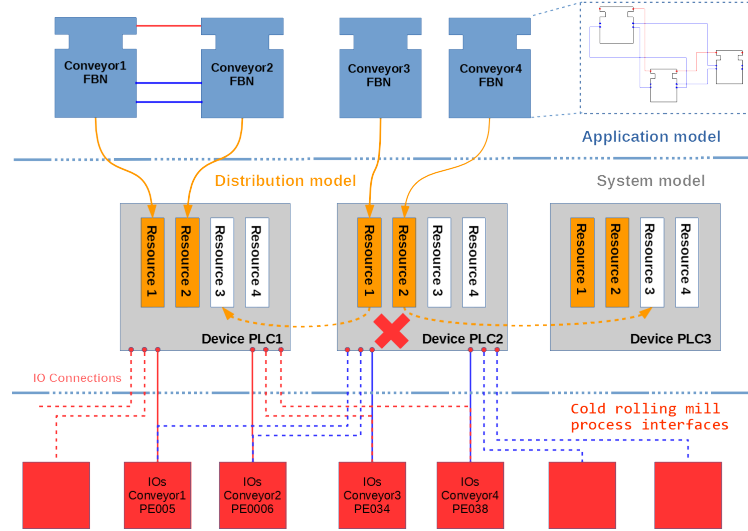


Fig. 2: Concept for IEC 61499 Distribution model generation and redeployment

of the identified constraints, objectives and annotations while striving for high availability of such plants.

While creating the Application model (depicted in upper part of Figure 2), we specify software annotations for each SWC (i.e. FB network, represented as blue blocks in Figure 2). For instance, we specify required hardware skills (e.g. access to conveyor's IOs) and safety levels to require a higher safety measures for the safety-critical components. In the second step, we take care of the hardware architecture, i.e. System model, presented in middle part of Figure 2. In order to save costs, it is possible to use one device (i.e. PLC, represented as grey block in Figure 2) to control several PTS's components, as the components require less than ten IOs. One device can have several resources to independently run several FB networks. However, we annotate the maximal number of resources within a device to avoid jeopardizing its computational and memory performances. Moreover, we consider redundancy in case one of the devices loses its functionality. Thus, every device will have access to several PTS's components, which are annotated as available hardware capabilities. In the example from Figure 2, each device can have up to four resources running. Two of them are used to independently control two different components (depicted with red and blue solid lines in Figure 2). For redundancy reasons, each device has access to four other components (depicted with red and blue dashed lines in Figure 2), controlled by another devices. Thus, it is possible to redeploy FB networks without changing the connections between hardware components.

After modeling the logical and hardware architectures, as well as specifying their annotations, we define constraints for calculating valid deployment configurations. We may strive to

deploy SWCs to the HWCs with required SIL, HW and RTE capabilities, while limiting the number of running resources within each HWC. When the desired constraints are specified, a solver (e.g. SMT solver) is launched to generate a set of valid deployment configurations. To increase the quality of the solution, a quality attribute such as HWC load is defined (i.e. minimize number of SWCs on each HWC). Thus we select valid configuration for which HWCs are less stressed and then perform the deployment according to the selected configuration.

During runtime we monitor the state of the components. In case a certain HWC losses its functionality or becomes overloaded (e.g. PLC2), we select another valid configuration from the set of the calculated configurations. Here we apply the second best configuration according to the quality attribute (i.e. HWC load), since the first one is invalidated due to the PLC2 failure. Alternatively, we can strive for a configuration that leads to less HWCs' changes. After configuration selection, we redeploy necessary SWCs using IEC 61499 reconfiguration services.

This way, we achieve a failure recovery without downtimes, as we apply another valid configuration without a time exhaustive recalculation. The recalculation of the configurations considering actual system status can be performed afterwards, during the system runtime.

## 5 Conclusion & Future Work

By analyzing the specifics of the industrial automation domain, more precisely IEC 61499 based systems, we identified constraints, objectives and annotations applicable for deployment problem definition. Thereby, we took first steps towards an automatically optimized deployment of industrial automation systems. The domain-specific case study exhibits the applicability of the DSE approach in conjunction with IEC 61499 models in an industry 4.0 relevant scenario, namely failure recovery without downtime.

Within the next step, we intend to implement and validate the proposed concept by using Eclipse 4diac, an open source, IEC 61499-compliant engineering framework [SZV08]. We plan to extend the framework by the identified annotations, constraints and objectives, as well as the deployment synthesis model. Moreover, we will continue identifying relevant constraints, objectives and annotations.

## References

- [AB17] Alexander Diewald, Sebastian Voss; Barner, Simon: A Lightweight Design Space Exploration And Optimization Language. In: CEUR Workshop Proceedings. 2017.
- [Ar15] Aravantinos, Vincent; Voss, Sebastian; Teufl, Sabine; Hölzl, Florian; Schätz, Bernhard: AutoFOCUS 3: Tooling concepts for seamless, model-based development of embedded systems. In: CEUR Workshop Proceedings. 2015.

- [BS01] Broy, Manfred; Stolen, Ketil: Specification And Development Of Interactive Systems Focus On Streams Interfaces And Refinement. Springer, 2001.
- [Bu17] Bundesministerium für Wirtschaft und Energie (BMWi): Beziehungen zwischen I4.0-Komponenten – Verbundkomponenten und intelligente Produktion. Technical report, 2017.
- [BV13] Becker, Klaus; Voss, Sebastian: Towards Dynamic Deployment Calculation for Extensible Systems using SMT-Solvers. In: 1st Open EIT ICT Labs Workshop on CPS Eng. 2013.
- [Ed17] Eder, Johannes; Zverlov, Sergey; Voss, Sebastian; Khalil, Maged; Ipatiov, Alexandru: Bringing DSE to life: exploring the design space of an industrial automotive use case. In: 20th Int. Conf. on M&E Lang. and Syst. (MODELS). 2017.
- [Le11] Lepuschitz, Wilfried; Zoitl, Alois; Vallée, Mathieu; Merdan, Munir: Toward self-reconfiguration of manufacturing systems using automation agents. IEEE Trans. on Systems, Man and Cybernetics, 2011.
- [SC05] SC65B, IEC: , IEC 61499-1 Function Blocks for Industrial Process Measurement and Control Systems. Part 4: Rules for compliance profiles, 2005. Geneva.
- [Sc10] Schätz, Bernhard; Schätz, Bernhard; Hölzl, Florian; Lundkvist, Torbjörn: Design-Space Exploration through Constraint-Based Model-Transformation. In: 17th IEEE Int. Conf. and Workshops on Eng. of Comp. Based Syst. 2010.
- [SJC14] Sinha, Roopak; Johnson, Kenneth; Calinescu, Radu: A scalable approach for re-configuring evolving industrial control systems. In: 19th IEEE Int. Conf. on Emerging Technologies and Factory Automation. 2014.
- [Sp16] Spindler, M.; Aicher, T.; Vogel-Heuser, B.; Günthner, W.: Efficient Control Software Design for Automated Material Handling Systems Based on a Two-Layer Architecture. In: Int. Conf. on Adv. Log. and Trans. 2016.
- [SZV08] Strasser, Thomas; Zoitl, Alois; Valentini, Antonio: Framework for Distributed Industrial Automation and Control (4DIAC). In: 6th IEEE Int. Conf. on Ind. Info. 2008.
- [VS13] Voss, Sebastian; Schatz, Bernhard: Deployment and scheduling synthesis for mixed-critical shared-memory applications. In: Proceedings of the Int. Symp. and Workshop on Eng. of Comp. Based Syst. 2013.
- [Wa16] Wang, Shiyong; Wan, Jiafu; Li, Di; Zhang, Chunhua: Implementing Smart Factory of Industrie 4.0 : An Outlook. Int. Journal of Dist. Sensor Net., 2016, 2016.
- [Wa17] Wagner, Constantin; Epple, Ulrich; Metzger, Alfred; Debus, Kai; Treivous, Vadim; Tarnow, Matthias; Helle, Christoph: Requirements for the Next Generation Automation Solution of Rolling Mills. In: 43rd Annual Conf. of the IEEE Ind. Elec. Society. 2017.
- [We16] Wenger, Monika; Zoitl, Alois; Blech, Jan Olaf; Peake, Ian; Fernando, Lasith: Cloud based monitoring of timed events for industrial automation. In: Proceedings of the Int. Conf. on Parallel and Dist. Syst. 2016.
- [ZKC16] Zverlov, Sergey; Khalil, Maged; Chaudhary, Mayank: Pareto-efficient deployment synthesis for safety-critical applications in seamless model-based development. In: Proceedings of the 8th European Congress on Embedded Real Time Software and Systems. 2016.
- [ZV09] Zoitl, Alois; Vyatkin, Valeriy: IEC 61499 architecture for distributed automation: The "glass half full" view. IEEE Industrial Electronics Magazine, 2009.

# Exploration of hardware topologies and complexity reduction

Johannes Eder<sup>1</sup>

**Abstract:** This paper shall give an overview over a dissertation project in the area of design space exploration for distributed, embedded systems. As the engineering of distributed embedded systems is getting more and more complex due to increasingly sophisticated functionalities demanding more and more powerful hardware, automation is required in order cope with this rising complexity. Using a model based systems engineering approach enables design space exploration methods which provide such automations, formalizing the problem in order to be solvable e.g. by SMT solvers. However, due to the complexity of the systems, those problems cannot be efficiently solved. It is thus furthermore required to provide mechanisms to reduce the complexity. This can be achieved by using domain specific knowledge of the systems and apply machine learning approaches to learn about the problem and thus provide means to reduce its complexity.

## 1 Introduction - System engineering and its complexity

The design of distributed, embedded systems is characterized by an ever-growing complexity of those systems. This is caused by an increasing set of functionality, as well as an increase in complexity in the underlying hardware architecture and topology. Considering e.g. the automotive domain, the introduction of more and more autonomous functions not only increases the amount of software in the vehicles but also demands hardware capable of executing this software.

Using a classical systems engineering approach like the V-Model, as proposed e.g. in the ISO 26262 for functional safety of automotive vehicles, the development process is divided into the product development on software and hardware level and the integration of those two parts. In the following the rising complexity of not only software, hardware and integration but also of the aspect of variability will be outlined which demands automation support in the development process of distributed embedded systems.

**Software complexity** The main driver of the complexity of software for distributed embedded systems are increasingly sophisticated functionalities e.g. in the automotive domain, due to the trend towards more and more autonomous vehicles resulting in a rising number of driver assistance and autonomous functions. In 2007, e.g., a premium car contained already 270 user functions resulting in 2500 'atomic' software functions

---

<sup>1</sup> fortiss GmbH, MbSE, Guerickestrasse 25, 80805 Munich, eder@fortiss.org



[Pr07]. A substantial part of this complexity is caused by the intricate dependencies of these functions, complicated by the different, and most often contradicting, requirements of these functions.

A further driver of complexity are new arising scenarios like plug&play capability of the systems enabling software updates, addition of new software and moving of software functions from one ECU to another ECU. As a next step this capability has to be provided over the air. Consequently, a further scenario arises, where vehicle functions are executed in the cloud. Figure 1 provides an outline of such future scenarios in the top row.

All of these scenarios share one common problem: it has to be ensured/verified that the requirements of the system are still met. This entails that mechanisms have to be provided which enable the verification of those systems at any point in time. As of now these mechanisms can only be provided during development and due to extensive testing. Considering the above mentioned plug&play scenarios, the vehicles themselves have to be able to verify on their own if their (software) system works correctly, in the future.

**Hardware complexity** The main driver of complexity considering the hardware in distributed embedded systems is the mere number of heterogeneous electrical control units (ECUs) connected via multiple networks (different bus systems and gateways). In 2007, those hardware architectures contained up to 67 ECUs [Pr07]. In 2010, automotive hardware architectures were already consisting out of up to 100 ECUs [ZP12][PBK10].

As a result, the trend goes toward more centralized, multi-core architectures as the hardware architectures would otherwise too complex on the one hand, and on the other hand increasingly sophisticated driver assistance and autonomous functions require more powerful hardware [So13]. Regarding figure 1, the trend from today's distributed hardware (E/E) architectures goes over domain centralized E/E architectures to vehicle centralized E/E architectures [GL16], hence, towards more integrated hardware architectures including cloud interoperability. In the aerospace domain the concept of centralized architectures can already be found in the IMA or ARINC 653 standard.

Consequently, hardware architecture layouts/topologies are changing from approximately 100 ECUs connected via various buses and gateways to smaller more centralized architectures with more powerful resources considering e.g. computational capabilities. Yet, figure 1 only proposes a vague vision of an E/E (hardware) architecture of the future. The future hardware topology of an automotive E/E is thus an open question.

**Integration complexity** Concerning the development according to a standard like ISO 26262 as mentioned before, integration between both software and hardware is a substantial part. One part of this integration is the mapping of software to hardware where its functionality will be executed. Considering a distributed embedded system this mapping has

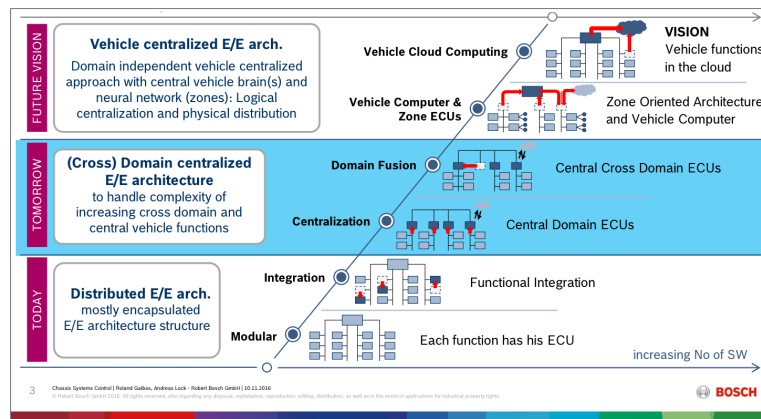


Fig. 1: Trends in Automotive E/E Architectures [GL16]

a big implication as it poses a certain demand on the communication resources which are used, due to the fact, that executing a certain software function on one ECU in the system may require input from another software function located on a different ECU. Moreover, such deployment has to meet constraints which have to hold. Considering the ISO 26262, especially concerning safety constraints. Taking into account that the number of software functions is ever growing while the hardware architectures of the future will change from large distributed to more centralized systems, the integration of the parts themselves is getting more complex, too.

Variability is a fourth driver of complexity as products are not only more and more tailored towards individual needs but also due to regionalization covering aspects such as customer needs and legal issues.

Considering (software) functions of a system, the number of functions differs due to customer needs and region specific variants. This means that some functions may be optional and some functions may be required only in specific regions or even forbidden in some other regions.

Considering the underlying hardware, the above mentioned functions may have an impact as e.g. less functions would require less ECUs which results in a cheaper product. On the other hand, the ECUs or networks themselves may exist in different variants considering cost, performance and safety attributes to mention just a few.

So during system development one has to cope with the rising demand of individualization of its products while at the same time trying to minimize in particular the cost of the product regarding all possible product variants.

## 2 Problem statement

In order to effectively manage the ever increasing complexity in the design of embedded systems, development processes in general, and model-based approaches in particular, support the development. They are assuming an idealized (component-based) model of computation, abstracting away from implementation issues like interference aspects of the execution platform resulting from shared computation or memory resources.

However, as requested by the ISO 26262, those simplifying abstractions must be met by development steps which ensure that the assumptions behind these abstraction are not violated by the properties of the implementation platform. For instance, during SW-/HW-integration, platform mechanisms must avoid an overload in bus communication, such that messages do not arrive too late or even get lost. The distribution of software across different execution platforms is thus highly affecting different aspects of such an integration. Due to the complexity of the design alternatives, automated support for selecting optimal solutions is needed. A question whose answer is more and more demanded in industry considering systems engineering is hence also one of the most challenging ones:

**"What is a good or even optimized/optimal system design and how to find it - automatically?".**

The definition of what a "good" or even optimal system design looks like, requires a detailed (formalized) understanding of the different parts of the system which are concerned, as well as the requirements, namely the constraints and optimization objectives, which limit the set of possible designs. It is furthermore highly dependent on the various aspects of system design. In particular, considering a (optimal) synthesis of a technical HW platform, one has to integrate knowledge from various parts, to answer this question:

1. Software parts (components) of the system which represent its functionality, pose resource demands w.r.t. to the application considering e.g. a certain computing power or communication.
2. Hardware parts (components) which actually provide computing and communication capabilities and are composed according to certain hardware layout rules.
3. The deployment which assigns software parts to hardware parts where their functionality can be realized.
4. Variability of software parts and hardware parts leading to different deployments and thus to different variants of system designs.

The (automatic) identification of valid designs, in particular valid hardware designs, is a manually unsolvable task, especially considering not only the rising complexity within the

development of software and hardware, but also considering the different variants in the development especially of the hardware part. A sensor for example may exist in different variants differing in properties such as cost or performance but also in the set of possible interfaces enabling a connection to differing communication units (buses). A decision in favor of a certain sensor is thus affecting the whole hardware topology as its interface limits the set of possible communication resources which can be used. Striving for an optimal hardware topology is furthermore closely intertwined with the deployment of SW components to HW components due to the fact that software is demanding a certain amount of computational resources (memory, performance,...) in order to be executed as well as communication resources enabling exchange of information. So, the problem of finding an optimal hardware topology together with a deployment, while at the same time handling variability in order to come up with an optimal system design, demands automation.

### 3 Contribution

The envisioned contribution of this thesis is two-folded. Firstly, we want to propose a synthesis methodology capable of synthesizing a distributed embedded systems architecture while at the same time considering variability aspects of this architecture. Secondly, we want to provide a classification of complexity for this domain specific scenario in order to be able to increase the efficiency of the proposed methodology.

#### 3.1 Hardware Topology Synthesis

Considering the future vision of E/E architectures depicted in figure 1, the envisioned contribution of this thesis goes even one step further such that the E/E architecture is not fixed at all. Using the 150% model approach - which is common in variability research - we want to propose an approach that synthesizes a E/E architecture topology. This entails that given 150% of all possible elements of an E/E architecture and a (100%) component-based network of all SW-components (functions) of the system under development, the goal of this thesis is not only to choose 100% of the needed E/E elements but also to set up a respective topology of these elements.

At the same time, all requirements for this system have to be considered. On the one hand, constraints which limit the set of possible solutions. Firstly, those constraints may be deployment specific such that the limit they set of possible deployments of certain SW-Components to certain ECUs due to e.g. safety requirements. Secondly, those constraints may be topology specific, such that they restrict the set of possible connections in the E/E architecture, e.g. the connection of ECUs to communication devices such as buses or gateways. On the other hand, optimization objectives have to be considered, which optimize the topology into certain directions such as the overall cost of the E/E architecture or the number of ECUs.

Specifically, this entails the following sub-contributions.

1. Dedicated models of the technical (E/E) architecture, covering variability as well as topological aspects.
2. A respective language to formalize the synthesis problem. A first step has been done in [Ed17] covering the deployment aspects of the methodology. The language has to be extended in order to be able to express topological and variability aspects.
3. A transformation of the language into the SMTLIB2 language standard in order to be able to solve problems expressed in this language with state-of-the-art SMT solvers.
4. A dedicated exploration methodology providing an overall concept for such a synthesis. (Categories, RuleSets, Process, In and Output Models)

The resulting technical architecture may even be disruptive, as the outcome can not be foreseen. It specifically takes into account the available 150% of the technical architecture elements and calculates, regarding the constraints which have to hold, an optimal topology considering the given optimization objectives. So the resulting technical architecture may differ very largely from today's E/E architectures.

### **3.2 Classification of Complexity for (this) domain-specific synthesis-scenario**

State of the art SMT solvers like Z3 have become powerful tools in order to solve problems like the calculation of an optimal topology or the deployment of SW components to hardware components. However, already a deployment problem such as mentioned in [Ed17] (31 SW components, 22 HW components (ECUs and buses), 71 constraints, 6 objectives) takes about an hour to be solved. Considering that a topology synthesis takes a 150% model of HW components the number of technical components alone raises the problem complexity such that solving this problem will take more than a few hours if not days. Reducing the complexity of such a problem is thus an inevitable step to still make use of an automated solving of such problems.

In this thesis we want to use machine learning to learn about the complexity of the given problem. In particular, we want to build up a connection between industrial input models, which, given to our proposed synthesis, produce a hardware topology as output. This entails the following contributions:

1. Using a supervised learning approach, we want to provide an estimation of the calculation time for a topology synthesis given a specific input model. This includes the classification of input architectures according to certain criteria which would enable the prediction of the time which is needed for a synthesis for a specific input model.

2. Using a supervised learning approach we want to provide a configuration for SMT solvers which suits best for the given input models such that we can reduce the calculation time for a topology synthesis. Considering that e.g. the Z3 SMT solvers provides thousands of different user configurable tactics choosing the right tactic may have strong impact on the solving time. This also includes a classification of input architectures according to certain criteria which would enable the selection of proper solver configurations, namely tactics, in order to solve the given input model in the least possible time.

## 4 Related work

This section shall give a brief overview over the existing work in the area of design space exploration, focusing on the area of solving the deployment problem especially in the automotive domain 4.1. Moreover, also outlining the work which has been conducted on topological synthesis 4.2. Lastly, a very brief overview over reduction of complexity, especially, for SAT based problems will be provided in 4.3.

### 4.1 Deployment Synthesis

There has been a variety of works conducted in the area of deployment synthesis often combined with schedule synthesis. Voss and Schätz proposed the joint synthesis of deployment and schedules for mixed critical, shared-memory applications in [VS13] and with special focus on automotive safety requirements (ISO 26262) in [SVZ15]. Both approaches [VS13] and [SVZ15] optimize a deployment of software tasks to computing resources by multiple optimization criteria, considering different constraints and additionally calculating schedules for these deployments. However they are only considering a fixed hardware topology with no variability considerations.

The deployment problem in the context of fail operational systems is presented by Becker and Voss in [BV15] focusing especially on failure scenarios in a system and the resulting change considering the deployment of software functions. This approach focuses more on the fail operational aspect of embedded systems by pre-calculating possible deployments to enable graceful degradation in case of a system failure. Thus, a fixed hardware topology and no variability is regarded in this approach.

[Ro17] describe the synthesis and exploration of multi-level, multi perspective architectures of automotive embedded systems. This work considers multiple layers in the system specifically the deployment of (software) functions or components to the hardware layer considering execution units, communication units and the power topology. However, even though they are considering variability through all layers, they are considering a fixed hardware topology, varying only in the type or optionality of components.

## 4.2 Topology Synthesis

Bajaj et al. propose an optimized calculation of cyber-physical system architectures in [Ba15] with focus on safety and reliability. Taking into consideration the reliability of the system by using its interconnection structure, they are synthesizing topologies. This approach calculates an optimal hardware network topology using different optimization objectives while at the same time regarding constraints which have to be met. Thus, this approach starts with a completely unfixed hardware topology calculating an optimal topology of hardware resources. However, they do not consider the deployment of software tasks which could (heavily) influence the topology considering the computation and communication resource usage. They are furthermore not considering variability.

A multi objective routing and topology optimization approach for networked embedded system is presented by [Gl08]. They are allocating a network of communication processes to a set of resources which are themselves connected. Although this work considers the optimization of a network topology of an embedded system, the network itself is fixed. Thus the topology of the system may only change in the usage or non-usage of resources. Variability of resources is not considered in this work.

## 4.3 Complexity reduction

There has been a variety of work conducted on complexity reduction particularly in reducing the complexity of boolean satisfiability problems.

[Cr96] and [De16] e.g. especially focused on finding symmetry breaking predicates considering SAT problems in order to gain massive speed ups in solving those problems.

In [KJS10] Kang et al. applied symmetry breaking predicates in a design space exploration context aiming to reduce the complexity. They showed the applicability of there by using a classical deployment problem.

## 5 Conclusion

Concluding, this thesis shall firstly provide a hardware topology synthesis approach and methodology using models on software and hardware level. On the hardware different variants of elements shall be regarded. Secondly, the complexity of such problems shall be provided. This can be achieved by using domain specific knowledge and the use of machine learning mechanisms providing information to .

## Literaturverzeichnis

- [Ba15] Bajaj, Nikunj; Nuzzo, Pierluigi; Masin, Michael; Sangiovanni-Vincentelli, Alberto: Optimized Selection of Reliable and Cost-Effective Cyber-Physical System Architectures. Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015, S. 561–566, 2015.
- [BV15] Becker, K.; Voss, S.: Analyzing Graceful Degradation for Mixed Critical Fault-Tolerant Real-Time Systems. In: 2015 IEEE 18th International Symposium on Real-Time Distributed Computing. S. 110–118, April 2015.
- [Cr96] Crawford, James; Ginsberg, Matthew; Luks, Eugene; Roy, Amitabha: Symmetry-breaking predicates for search problems. KR, 96:148–159, 1996.
- [De16] Devriendt, Jo; Bogaerts, Bart; Bruynooghe, Maurice; Denecker, Marc: Improved static symmetry breaking for SAT. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 9710:104–122, 2016.
- [Ed17] Eder, Johannes; Zverlov, Sergey; Voss, Sebastian; Khalil, Maged; Ipatiov, Alexandru: Bringing DSE to life: exploring the design space of an industrial automotive use case. In: 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS). 2017.
- [Gl08] Glaß, Michael; Lukasiwycz, Martin; Wanka, Rolf; Haubelt, Christian; Teich, Jürgen: Multi-objective routing and topology optimization in networked embedded systems. Proceedings - 2008 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, IC-SAMOS 2008, S. 74–81, 2008.
- [GL16] Galbas, Roland; Lock, Andreas: , Trends in Automotive E/E architectures. [http://www.safetrans-de.org/de\\_21\\_Industrial\\_Day.php](http://www.safetrans-de.org/de_21_Industrial_Day.php), 2016.
- [KJS10] Kang, Eunsuk; Jackson, Ethan; Schulte, Wolfram: An approach for effective design space exploration. Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems, S. 33–54, 2010.
- [PBK10] Prasad, K. V.; Broy, M.; Krueger, I.: Scanning Advances in Aerospace & Automobile Software Technology. Proceedings of the IEEE, 98(4):510–514, April 2010.
- [Pr07] Pretschner, Alexander; Broy, Manfred; Krüger, Ingolf H.; Stauner, Thomas: Software engineering for automotive systems: A roadmap. FoSE 2007: Future of Software Engineering, S. 55–71, 2007.
- [Ro17] Ross, Jordan A.; Murashkin, Alexandr; Liang, Jia Hui; Antkiewicz, Micha?; Czarnecki, Krzysztof: Synthesis and exploration of multi-level, multi-perspective architectures of automotive embedded systems. Software and Systems Modeling, S. 1–29, 2017.
- [So13] Sommer, Stephan; Camek, Alexander; Becker, Klaus; Buckl, Christian; Zirkler, Andreas; Fiege, Ludger; Armbruster, Michael; Spiegelberg, Gernot; Knoll, Alois: RACE : A Centralized Platform Computer Based Architecture for Automotive Applications. 2013.
- [SVZ15] Schätz, Bernhard; Voss, Sebastian; Zverlov, Sergey: Automating Design-space Exploration: Optimal Deployment of Automotive SW-components in an ISO26262 Context. Proceedings of the 52Nd Annual Design Automation Conference, S. 99:1–99:6, 2015.



- [VS13] Voss, Sebastian; Schatz, Bernhard: Deployment and scheduling synthesis for mixed-critical shared-memory applications. Proceedings of the International Symposium and Workshop on Engineering of Computer Based Systems, (April):100–109, 2013.
- [ZP12] Zeller, Marc; Prehofer, Christian: Modeling and efficient solving of extra-functional properties for adaptation in networked embedded real-time systems. JOURNAL OF SYSTEM ARCHITECTURE, S. 1067–1082, 2012.

## Ein Mittel zur Wiederverwendung – Komponentenbasierte Architekturen in der Automatisierungstechnik

Constantin Wagner,<sup>1</sup> Julian Grothoff,<sup>2</sup> Prof. Dr.-Ing. Ulrich Eppel<sup>3</sup>

### 1 Einleitung

Im Rahmen einer fortschreitenden Vernetzung und zunehmenden Flexibilisierung der industriellen Produktion durch Initiativen wie Industrie 4.0 und IoT rückt die wandelbare Fabrik und die rentable Produktion mit der Losgröße eins in den Fokus. Zur Umsetzung der damit verbundenen neuen Anforderungen ist eine Anpassung der Produktionssysteme nötig. Dazu müssen sich Hardware und Software, sowie Embedded Systeme gleichermaßen auf die jeweiligen neuen Aufgaben einstellen lassen. Da es nur schwer möglich ist alle Aufgaben im Lebenszyklus einer wandelbaren Fabrik zum Zeitpunkt ihrer Planung vorherzusehen, muss diese so entworfen sein, dass sie sich an neue Aufgaben anpassen kann [Wa17]. Zusätzlich müssen Konzepte entwickelt werden, die den Aufwand für das Engineering dieser Systeme auf das absolut nötige begrenzen. Ein Ansatz dafür ist die Wiederverwendung von bestehenden Lösungen, seien es Hardware- oder Softwarelösungen, zu unterstützen. Die Vorteile der Wiederverwendung wurden in [Li94] untersucht, und umfassen eine Verbesserung der Markteinführungszeit, der Produktqualität und der Produktivität. Die Herausforderungen bzw. die Hemmnisse für die Wiederverwendung bestehender Lösungen sind organisatorischer und menschlicher Natur [Me88]. Zusätzlich ist die Entwicklung von wiederverwendbarem Code bis zu 480% teurer als von konventionellen Programmen [Li94].

In diesem Beitrag werden, ausgehend von einer Definition des Komponenten Begriffs, verbreitete und akzeptierte Architekturen auf Basis von Komponenten in der Automatisierungstechnik vorgestellt. Anschließend wird der Nutzen von komponentenbasierten Architekturen als Grundlage für die Wiederverwendung in der Automatisierungstechnik diskutiert. Abschließend werden bestehende Herausforderung im Hinblick auf die Wiederverwendung dargestellt und im letzten Abschnitt wird für diese ein Lösungsweg skizziert.

---

<sup>1</sup> Lehrstuhl für Prozessleittechnik, RWTH Aachen University, Turmstraße 46, D-52064 Aachen, c.wagner@plt.rwth-aachen.de

<sup>2</sup> Lehrstuhl für Prozessleittechnik, RWTH Aachen University, Turmstraße 46, D-52064 Aachen, j.grothoff@plt.rwth-aachen.de

<sup>3</sup> Lehrstuhl für Prozessleittechnik, RWTH Aachen University, Turmstraße 46, D-52064 Aachen, eppel@plt.rwth-aachen.de

## 2 Komponenten in der Automatisierungstechnik

Der Begriff der Komponente ist weitverbreitet und wurde u. a. in der DIN SPEC 40912 [DIN14] und von der Object Management Group [OMG15] definiert. In [DIN14] wird eine (technische-) Komponente als „vorgefertigte, in sich strukturierte und unabhängig hantierbare Einheit zur Realisierung einer konkreten Rolle in einem System“ definiert. Wenn die Forderung nach einer definierten Schnittstelle als Bestandteil der Hantierbarkeit aufgefasst wird, ist die Definition vergleichbar zu der Definition von Softwarekomponenten in UML [OMG15]. Ein Überblick über verschiedene Definitionen sowie die charakterisierenden Eigenschaften von Softwarekomponenten ist in [Di02] zu finden. In diesem Beitrag werden Komponenten im Sinne der technischen Komponenten als Soft- und Hardware, sowie Mischformen wie eingebettete Systeme verstanden. Dies ist für die Automatisierungstechnik ein essentieller Gedanke, da ihre Kernkompetenz in der Integration von Funktionalität liegt, die interdisziplinär in Hard- und Software realisiert wird. Durch die strukturierte Art der Komponenten kann eine Hierarchisierung erreicht werden. Damit existieren im Kontext der Betrachtung zwei Arten von Komponenten: atomare und aggregierte Komponenten (Komponentensystem). Atomare Komponenten können nicht unterteilt werden und sind Black-Boxen. Aggregierte Komponenten bestehen aus atomaren und aggregierten Komponenten und ihr interner Aufbau ist erkundbar. Komponenten sind durch ihre Handhabbarkeit in mehreren Systemen einsetzbar und damit direkt eine Quelle der Wiederverwendbarkeit (vgl. Abschnitt 3,[Di02]).

Beispiele für komponentenbasierte Architekturen in der Automatisierungstechnik sind Funktionsbausteine aus der IEC 61131 [IEC14] und IEC 61499 [IEC05], Package Units, die Betrachtung von Sensoren und Aktoren in Anlagen, sowie Rohrleitungs- und Instrumenten-Fließbilder. Funktionsbausteine und Hardwarekomponenten sind klassische Beispiele für komponentenbasierte Architekturen, die sich über Jahrzehnte etabliert haben. Aber auch die Elemente von Fließbildern können im Weiteren als Komponenten aufgefasst werden, da sie in ihrer Umgebung einzeln handhabbar sind, einen konkreten Zweck haben und definierte Schnittstellen bereitstellen.

Für den Aufbau des Softwareteils der Automatisierungslösung (System für die Automatisierung z. B. einer Anlage, vgl. [Vo14]) werden in der Regel die Programmiersprachen der IEC 61131 verwendet [JT00]. Zur Strukturierung hat sich die Verwendung von Funktionsbausteinen in Funktionsbausteinnetzwerken etabliert. Dabei werden einzelne Aufgaben in einem Funktionsbaustein gekapselt, der über definierte (Signal-)Schnittstellen, sogenannte Ports, verfügt und innerhalb des Automatisierungssystems instanziiert, gelöscht und manipuliert werden kann. Der rückwirkungsfreie Austausch von Daten wird mit Signalverbindungen zwischen den Ports von Bausteinen realisiert. Diese komponentenbasierten Softwarearchitektur hat sich historisch aus der Hardware Komponententechnik (Einzelgerätetechnik) entwickelt, was die vereinheitlichte Betrachtung von Hard- und Softwarekomponenten als "technische Komponenten" weiter motiviert. Als Erweiterung der klassischen Bausteintechnik im Hinblick auf Verteilbarkeit von Lösungen und der Ausführungssemantik wurde die IEC 61499 vorgestellt. Für die

Realisierung einer eventbasierten Ausführung wurden neben den klassischen Ports Eventports eingeführt. Analog dazu existieren Eventverbindungen, um den Fluss von Events zwischen den Bausteinen abbilden zu können. Funktionsbausteine können zu aggregierten Funktionsbausteinen (CFB) zusammengefasst werden. Durch die Verwendung von globalen Variablen in einem Funktionsbaustein ist dieser allerdings keine Komponente im Sinne der vorgestellten Definition. Der Funktionsblock ist dann nicht mehr abgegrenzt und kann nicht in anderen Steuerungsprogrammen ohne die deklarierte Variable eingesetzt werden. Hierbei wird deutlich, dass die Hantierbarkeit der technischen Komponente essentiell für deren Wiederverwendbarkeit ist.

Wie erwähnt werden auch Hardwareteile als Komponenten in einer Anlage und damit als Bestandteil der Automatisierungslösung angesehen. Dies sind beispielsweise Sensoren und Aktoren, wie Pumpen und Temperatursensoren. Auch die Hardwarekomponenten verfügen über eine definierte Schnittstelle (z. B. Zweidraht über 4 bis 20 mA oder einen Profibus Anschluss). Eine Form der Aggregation von Hardwarekomponenten findet sich beispielsweise bei der Verwendung von Package Units. Dabei werden Module einer Anlage als komplette Einheit bereitgestellt, beispielsweise bei Verpackungsmaschinen. Diese können über eine eigene Automatisierungslösung verfügen, die sich entweder in ein übergeordnetes Leitsystem einfügt oder eigenständig betrieben wird. Die Wiederverwendung von Hardwarekomponenten kann dabei analog zu der Klasse-Instanz Beziehung in Softwaresystemen gesehen werden. Ein Pumpentyp kann mehrfach produziert und anschließend in verschiedenen Anlagen eingesetzt werden. Der Unterschied sind im wesentlichen die Grenzkosten in der Herstellung von Instanzen, die in Softwaresystemen nahezu Null sind.

Im Prozess der Planung einer Prozessanlage entstehen Diagramme, die den geplanten Aufbau der Anlage bestehend aus den Rohren und Instrumenten darstellen. Sie bilden die Grundlage für die Planung der Automatisierungslösung, da sie den Zweck (Rolle) der zur Prozessführung benötigten Komponenten beschreiben. Die einzelnen Bestandteile der Diagramme (z. B. Messstellen und Aktoren) sind mit den jeweiligen Anforderungen an sie und den Verbindungen zwischen ihnen dargestellt und können als Komponenten aufgefasst werden. Die Diagramme haben insofern zwei Aufgaben, als das sie auf der einen Seite ein Rollenmodell für die zu bauende Anlage und ihre Komponenten darstellen, auf der anderen Seite sind sie selbst Modelle, die sich aus einzelnen Komponenten zusammensetzen. Als elektronisches Austauschformat wurde dafür mit PandIX eine CAEX Bibliothek auf Basis der IEC 62424 [IEC16] vorgestellt.

Durch die zunehmende Verwendung von OPC UA [IEC10, GPP16] und der damit verbundenen Modellierung von Daten und Zusammenhängen in objektorientierten Informationsmodellen wird die Verwendung von komponentenorientierten Architekturen zunehmen. In diesen Informationsmodellen werden viele Informationen verschiedener Organisationseinheiten gespeichert und zugänglich gemacht werden. Daher ist davon auszugehen, dass diese zunehmend größer und komplexer werden. Es muss ein Anliegen

sein, den Aufwand für die Erstellung und Pflege dieser Modelle so gering wie möglich zu halten.

An den vorgestellten Beispielen ist zu erkennen, dass komponentenbasierte Architekturen in der Automatisierungstechnik weit verbreitet sind. Worin sich die Automatisierungstechnik von anderen Fachdisziplinen unterscheidet, ist die Betrachtung von interdisziplinären Systemen [Ho13]. Aufgrund der Rückwirkung auf den automatisierten Prozess (z. B. ein Produktionsprozess) muss neben dem Softwareteil der Automatisierungslösung zusätzlich noch die Ankopplung an den Prozess und damit die verbaute Hardware berücksichtigt werden [Vo14]. Insbesondere die Abhängigkeiten zwischen Software- und Hardware, sowie deren Modellierung stellen eine große Herausforderung dar. Im Kontext einer wandelbaren Fabrik sind diese Informationen allerdings wichtig, um das Gesamtsystem betreiben zu können.

### **3 Wiederverwendung**

Wiederverwendung bedeutet die mehrfache Nutzung eines Assets in mehr als einem System [ISO11]. Wie in Abschnitt 1 bereits ausgeführt, bietet die Wiederverwendung das Potential zur Verbesserung der Markteinführungszeit, der Produktivität und der Qualität von Softwareprodukten [Li94]. Nicht nur in der Softwareentwicklung, sondern auch in der Automatisierungstechnik und der Hardwareentwicklung wird seit langem versucht die Wiederverwendung von Assets zu unterstützen [Sc16]. Die Entwicklung von neuen Konzepten zur Unterstützung der Wiederverwendung, z. B. durch eine verbesserte Beherrschung der Variantenvielfalt, steht allerdings weiterhin auf der Agenda [Vo14]. Ein Überblick über die verschiedenen Stufen der Wiederverwendung, wie beispielsweise der mehrfachen Verwendung von Instanzen eines Komponententyps oder des Copy & Paste, ist in [Sc16] dargestellt.

In Abschnitt 2 wurden komponentenbasierte Architekturen in der Automatisierungstechnik vorgestellt. Diese Komponenten selbst und die teils implizite Nutzung von Typ-/Instanz-Konzepten sind der erste Schritt zur Unterstützung der Wiederverwendung. So werden beispielsweise mehrere Pumpen des gleichen Typs in verschiedenen Anlagen eingesetzt oder Funktionsbausteine in unterschiedlichen Automatisierungslösungen verwendet. Organisatorische Beschränkungen und eine fehlende explizite Verwaltung führen jedoch dazu, dass vorhandenen Komponenten, nicht jedem potentiellen Nutzer bekannt sind. Zusätzlich zur Wiederverwendung von vorhandenen Komponenten wird versucht die Struktur von verwendeten Lösungen (z. B. Reglerstrukturen) erneut zu nutzen, dies stellt eine Wiederverwendung von Designvorlagen dar. Werden die Lösungen selbst wiederverwendet handelt sich in der Regel um ein Copy & Paste oder die Verwendung eines Templates. Möglichkeiten zur Ausgestaltung dieser Ansätze für Funktionsbausteine wurden in [WGE16] vorgestellt.

Es ist zu erkennen, dass der Entwurf und die Nutzung von Komponenten an sich ein sehr guter Ansatz für die Wiederverwendung sind, was in der Automatisierungstechnik bereits genutzt wird. Allerdings bietet sich noch Potential im Bereich der Wiederverwendung von aus Komponenten zusammengesetzten Lösungen (Komponentensysteme), wie beispielsweise Funktionsbausteinnetzwerken, Schrittketten oder Package Units. In diesen Bereichen ist es oft erforderlich Lösungen zu entwickeln, die sich nur in Nuancen unterscheiden. Daher ist eine Unterstützung bei der Wiederverwendung und Modifizierung von Lösungen erforderlich. Dabei müssen die Abhängigkeiten zwischen den verwendeten sowie möglicherweise modifizierten Komponentensystemen und dem Original abgebildet werden können, um die Wartbarkeit durch die Propagierung von Updates zu erleichtern. Zudem werden Rollenmodelle, die den Zweck der später eingesetzten Komponenten bestimmen, oft nicht explizit definiert und zusammen mit den realisierenden Komponenten über den Lebenszyklus der Anlage verwaltet. Dies erschwert die Zerlegung der Lösungsarchitektur in generische und spezielle Teile. Zusätzlich ist die Einbindung von unterschiedlichen Versionen von Assets und die Darstellung von den Abhängigkeiten zwischen diesen eine noch nicht abschließend gelöste Herausforderung und sollte ebenfalls berücksichtigt werden.

## **4 Ausblick**

In diesem Beitrag wurden komponentenbasierte Architekturen in der Automatisierungstechnik vorgestellt und ihr Nutzen für die Wiederverwendung verdeutlicht. Darüber hinaus wurden Herausforderungen im Bereich der Wiederverwendung von Komponentensystemen in Bezug auf die Unterstützung und die Bekanntheit von bestehenden Lösungen identifiziert. Zusätzlich müssen durchgängige Lösungen für hybride Systeme, d. h. aus Hard- und Softwarekomponenten bestehende Systeme, entwickelt werden.

Ein Weg die aufgezeigten Herausforderungen zu lösen ist es, ein Modell zu entwerfen, das Software- und Hardwarekomponenten gleichermaßen beschreibt. Dieses Modell muss von den Unterschieden abstrahieren und Komponenten als ganzes handhabbar machen. Durch dieses Modell sollen die verschiedenen Komponenten eines Systems und die Verbindungen zwischen diesen beschrieben werden. Ein solches Modell für Funktionsbausteine wurde in [En01] vorgestellt. Dieses Model bildet die Basis für die Beschreibung von hybriden Systemen als Varianten. So kann die Wiederverwendung von aus Komponenten zusammengesetzten Lösungen unterstützt werden.

Für den Einsatz in der Praxis sind geeignete Prozesse und Vorgehensmodelle nötig, die die Wiederverwendung unterstützen und sich in bestehende Prozesse einfügen. So wird die Akzeptanz und Nutzung der Wiederverwendung gesteigert. Damit die Modelle nicht alle händisch erzeugt werden müssen, sind automatisierte Modelltransformationen nötig, die den Nutzer soweit wie möglich von diesen wiederholenden und fehleranfälligen Tätigkeiten entlasten.

## Literaturverzeichnis

- [Di02] Dietzsch, Andreas: Systematische Wiederverwendung in der Software-Entwicklung. Deutscher Universitätsverlag, Wiesbaden and s.l., 2002.
- [DIN14] DIN SPEC 40912: Kernmodelle - Beschreibung und Beispiele, 2014.
- [En01] Enste, Udo: Generische Entwurfsmuster in der Funktionsbausteintechnik und deren Anwendung in der operativen Prozeßführung: Zugl.: Aachen, Techn. Hochsch., Diss., 2000, Jgg. 884 in Fortschritt-Berichte VDI Reihe 8, Meß-, Steuerungs- und Regelungstechnik. VDI-Verl., Düsseldorf, als ms. gedr. Auflage, 2001.
- [GPP16] Grüner, Sten; Pfrommer, Julius; Palm, Florian: RESTful Industrial Communication with OPC UA. IEEE Transactions on Industrial Informatics, S. 1, 2016.
- [Ho13] Holm, Thomas; Schröck, Sebastian; Fay, Alexander; Jäger, Tobias; Löwen, Ulrich: Engineering von "Mechatronik und Software" in automatisierten Anlagen: Anforderungen und Stand der Technik. In: Software Engineering (Workshops). S. 261–272, 2013.
- [IEC05] IEC 61499: Function blocks, 2005.
- [IEC10] IEC 62541. OPC Unified Architecture Part 1-10, Release 1.0, 2010.
- [IEC14] IEC 61131: Programmable controllers - Part 3: Programming languages, 2014.
- [IEC16] IEC 62424: Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE tools, Juli 2016.
- [ISO11] ISO/IEC 25010: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models, 2011.
- [JT00] John, Karl-Heinz; Tiegelskamp, Michael: SPS-Programmierung mit IEC 61131-3: Konzepte und Programmiersprachen, Anforderungen an Programmiersysteme, Entscheidungshilfen. VDI-Buch. Springer, Berlin and Heidelberg and New York and Barcelona and Hongkong and London and Mailand and Paris and Singapur und Tokio, 3., neubearb. Aufl. Auflage, 2000.
- [Li94] Lim, W. C.: Effects of reuse on quality, productivity, and economics. IEEE Software, 11(5):23–30, 1994.
- [Me88] Meyer, Bertrand: Object-oriented software construction, Jgg. 2. Prentice hall New York, 1988.
- [OMG15] OMG Unified Modeling Language, 2015. Version 2.5.
- [Sc16] Schröck, Sebastian: Interdisziplinäre Wiederverwendung im Engineering automatisierter Anlagen: Anforderungen, Konzept und Umsetzungen für die Prozessindustrie. VDI Verlag GmbH, 2016.
- [Vo14] Vogel-Heuser, Birgit; Diedrich, Christian; Fay, Alexander; Jeschke, Sabine; Kowalewski, Stefan; Wollschlaeger, Martin; Göhner, Peter: Challenges for Software Engineering in Automation. Journal of Software Engineering and Applications, 07(05):440–451, 2014.
- [Wa17] Wagner, Constantin; Grothoff, Julian; Epple, Ulrich; Drath, Rainer; Malakuti, Somayeh; Grüner, Sten; Hoffmeister, Michael; Zimmermann, Patrick: The role of the Industry 4.0 Asset Administration Shell and the Digital Twin during the life cycle of a plant. In: ETFA 2017 - 22nd IEEE International Conference on Emerging Technologies and Factory Automation. 2017.
- [WGE16] Wagner, Constantin; Grüner, Sten; Epple, Ulrich: Portabilität und Wiederverwendbarkeit von auf Funktionsbausteinnetzwerken basierenden Anwendungen. In: Entwurf komplexer Automatisierungssysteme. Magdeburg, 2016.

## Integrating a Signaling Component Model into a Railway Simulation

Daniel Schwencke<sup>1</sup>

**Abstract:** The validation of software component models is an important activity: The software product usually is derived more directly from a model than from a classical textual specification. Modeling tool suites support this activity by providing code generation and model simulation features. For simulation, an environment for the component needs to be set up: This may be stubs, or the real environment. We report on a third approach where a signaling component model is integrated into a railway simulation. We present the steps taken and what needs to be considered, the integration architecture, and give an account of which kinds of problems have been detected thanks to the simulation. From the successful integration we conclude that the generic model as well as the configuration data used for the integration are valid. Finally, we compare the aforementioned approaches to environments for model simulation.

**Keywords:** model; validation; simulation; railway; signaling; Radio Block Center; RBC; ETCS

### 1 Introduction

Model-based approaches have gained visibility over the last years. They are promising both in terms of higher product quality — due to the precision and compactness of models which can help to cope with system complexity — and greater efficiency — due to applicability of automated tools which verify properties or transform the model. Considering also the initial modeling effort, the latter in particular pays off when changes need to be made to a system as part of a development cycle or maintenance; they only must be made to the model instead of different documents and artifacts.

Models can be used for different purposes and at different stages of a system life-cycle. Here we are interested in using models in the context of system validation. For example, models can be used for system specification, and this specification can then be executed in order to validate it. Thus the model allows for testing already in this early life cycle-phase; in case of standardized specifications even independent from a concrete realization project. Models can also be used as reference or for test case generation ("model based testing") in order to validate real system implementations.

In some application areas model-based approaches have spread widely. For example in the automotive sector it seems a convenient way to support variant management. In the current

---

<sup>1</sup> German Aerospace Center (DLR), Institute of Transportation Systems, Lilienthalplatz 7, 38108 Braunschweig, Germany; daniel.schwencke@dlr.de



paper we present an example from the railway domain, where their adoption seems to be significantly lower. Besides being generally more safety-critical and conservative, other reasons for that may be the generic nature of railway signaling components (i.e. they need to be configurable for arbitrary track layouts) and (anticipated) risks for the mandatory component certification.

Nevertheless, Deutsche Bahn (DB) has produced a set of interface specifications for their new "digital interlockings" using models (NeuPro project) in the Systems Modeling Language (SysML) and has started to carry that approach to a European level (EULYNX, see [EUL18]). Starting with the particular interface to the so-called Radio Block Center (RBC) component, we have created an RBC SysML model using PTC Integrity Modeler. We have been reporting on that activity in the article [SHC17] of the previous MBEES workshop. In the current paper, we share our experiences with integrating and running the RBC model as part of a railway simulation in order to validate it. Note that this differs from the validation efforts of DB who validate single state charts through expert users which are provided with a graphical user interfaces for that state chart.

The paper is organized as follows: Sect. 2 recalls basic information on the RBC model used in our case study, and Sect. 3 gives an overview of the working steps necessary in order to integrate it into the railway simulation. Sect. 4, 5, 6 and 7 describe conditions on this undertaking exported by the different working step activities according to our experience. Sect. 8 and 9 contain information on running the model as part of the simulation and on its validation. Sect. 10 discusses alternatives to using a railway simulation environment, and Sect. 11 summarizes the paper and points to remaining work.

The author would like to thank his colleagues Mirko Caspar and Jonas Grosse-Holz for RBC integration support on the railway simulation side.

## **2 The Radio Block Center Model**

The signaling component chosen for modeling and integration into a railway simulation is the RBC. Being part of the European Train Control System (ETCS), an RBC provides a radio interface between interlocking and train (see Fig. 1). It receives information on the current state of signals and switches from the interlocking as well as on the current position from the train. Based on that information plus data on the railway infrastructure in its area, its main task is to issue so-called "movement authorities" to the train, telling how far and how fast the train may go. Other functionalities include for example emergency stopping of trains or managing temporary speed restrictions. An RBC is a safety critical system and needs to adhere to the highest standards for railway signaling systems (SIL4). Important aspects that led us to choose the RBC are the suitable level of complexity (reasonably complex, but manageable) and that it essentially is a software system.

The RBC model has been developed using PTC Integrity Modeler and basically consists of SysML block definition diagrams and state charts as well as additional C++ program

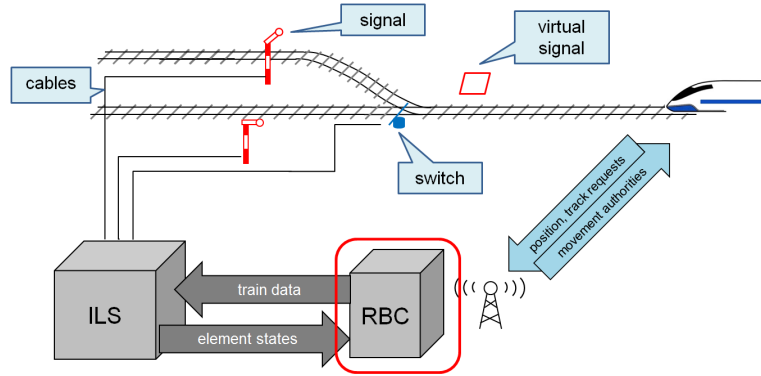


Fig. 1: The RBC as part of the ETCS Level 2 (simplified presentation, ILS = interlocking system).

code. The model exhibits no complete implementation of an RBC, only core functionality necessary to operate a demonstration line has been included. However, the functionality available has been modeled in a realistic scope; for example, the RBC is multi-train and multi-interlocking capable, can be used for arbitrary track layouts, and can be extended upon demand to include more special functionalities. For more information on the model as well as on the underlying standards, tools and languages we refer the reader to the articles [SH16; SHC17].

### 3 Approach

The railway simulation into which the model was to be integrated was given as the one available in DLR's RailSiTe® lab. It is regularly used for ETCS on-board unit tests, for which the lab is accredited, so it supports the simulation of train rides controlled by ETCS. Also, it is regularly used for studies conducted in a train driver simulator. This latter set-up was utilized for our simulation since it naturally supports validation of an RBC: The outcome of the most important RBC actions is visible on the driver machine interface (DMI), and the time of passing a balise group which triggers some RBC action can be detected from the front window.

Fig. 2 shows the working steps that have been performed in order to run the RBC model as part of DLR's railway simulation. The arrows indicate the order (dependencies) of the steps; iterations (backwards arrows) occur in many places in practice but have been omitted here to avoid a cluttered picture. First of all, the model to be integrated has been developed in a way that code could be automatically generated from it. On the lab side, the necessary interfaces have been prepared directly on program code level. All of those activities are generic (colored blue), i.e. they are independent of a particular railway infrastructure (track topology, signal locations, gradient and speed profiles etc.). This is important for signaling systems as a new development for each specific infrastructure would be unaffordable; but on

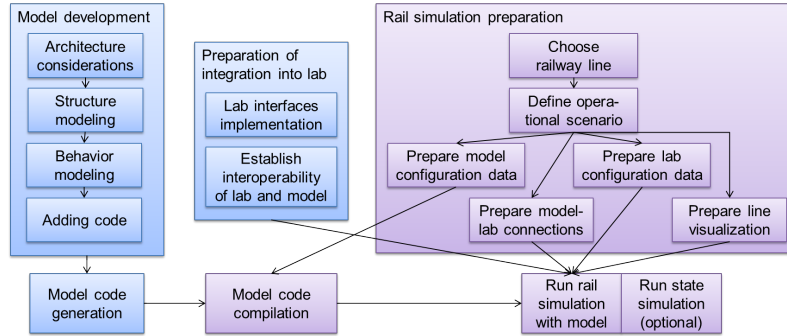


Fig. 2: Working steps towards running a signaling component model as part of a railway simulation. Generic development steps are colored blue, steps requiring a specific track layout are colored purple.

the other hand this poses challenges for system validation since the system can be configured for arbitrary infrastructures and needs to ensure safe operation in any case. For the purpose of this paper, we shall not discuss this issue further but limit ourselves to showing that we have kept our model generic and that it can indeed be configured for arbitrary infrastructures, as well as the lab (the corresponding activities are colored purple). Finally, the railway simulation of the lab can be run together with the model, i.e. a concrete operational scenario is simulated on the specific infrastructure for which the systems have been configured. As the modeling tool suite used provides state simulation capabilities, this can optionally be run in parallel, resulting in animated model diagrams during the railway scenario simulation.

#### 4 Creating a Model Suitable for Integration into a Railway Simulation

As shown in Fig. 2, model development basically comprises four kinds of activities: architectural considerations, structure modeling, modeling of the behavior and adding program code to the model. The goal of integration into a railway simulation already influences the first activity, since it includes the definition of interfaces of the model, some of which may be connected to the simulation later. In the case of the RBC the latter applies to the interlocking and train (on-board unit) interfaces, cf. Fig. 3. Luckily, in both cases we could build on existing standards (the German SCI RBC [SCI14] for the interlocking interface and ETCS Subset 026 [SS16a] for the train interface) which precisely define the messages that can be exchanged. Those have been implemented in the model (and made available in the lab, see Sect. 6 below). While it made sense to choose the original messages, it was decided to deviate from the original underlying communication mechanisms: Instead of the DB RaSTA protocol via UPD/IP Ethernet communication demanded by the SCI RBC and instead of the GSM-R radio communication demanded by ETCS, for both interfaces the messages are transferred via TCP/IP Ethernet between model and lab. This choice did not only imply easier implementation of the communication, but also could be realized

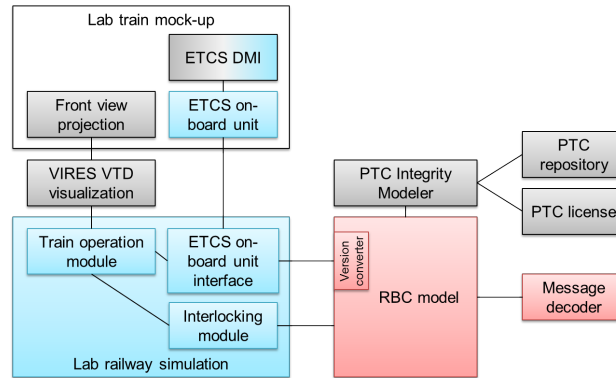


Fig. 3: Components and communication links for the integration of the RBC model into the railway simulation. The red components constitute the RBC, the light blue ones are the further relevant components of the railway simulation, and the grey ones pertain to visualization.

without special radio equipment. However, one needs to bear in mind the differences (e.g. concerning delay or reliability of transmission) that are introduced into the simulation compared to the original communication. In order to be able to work also with original components in the future, the RBC interfaces have been modeled in a flexible way so that arbitrary communication mechanisms may be plugged in.

One precondition on the model has been that it can be executed at least in real time, since this is desirable when including a train simulator in the railway simulation. This requirement needed to be kept in mind throughout all of the model development activities, as control and data flow are refined starting from interface definition, continuing with flat or deeply nested modeled structures and behavior which may use broadcasting or direct links, down to the use of more or less efficient data structures and algorithms on code level. Fortunately, real-time conditions for signaling systems are not that hard (usual time for a functionality in the overall signaling system is one to several seconds) and it turned out that so far there were no difficulties at all to meet the real-time requirement. However, this may become of interest again when the model is used for multi-train/multi-interlocking operation and further functionalities are included.

There are also some practical issues when running a railway component such as loading configuration data, getting diagnostic information and accessing the component for maintenance. Such issues are solved by manufacturers for real components operating in the real railway environment, but a different solution may be needed for models as part of a simulation. For easy configuration of the model, we decided to provide two separate files: One settings file containing identifier/value pairs for initialization during start-up, and one file containing infrastructure data simply in the form of program code which is included at compile time, creating several infrastructure objects. This way no additional data format needed to be defined and implemented. For diagnostic information, a flexible logging

mechanism has been included in the model. For interaction a simple command line menu has been prepared which provides a convenient way to establish or close RBC connections to individual components during runtime; for initial connections to be established automatically at start-up, default settings can be passed to the RBC via the settings file.

## 5 Code Generation

In principle, the code generation should be a push button activity and independent of the integration into the railway simulation, except that it is a first necessary step to transform the model into an executable form. In practice, it nevertheless is important to understand how the model is transformed to program code for various reasons:

First of all, code generation may only support part of the model diagrams and elements available in a language or modeling tool. The modeler needs to know this and to stick to that subset.

In case of a SysML model, there is no completely defined semantics available, but it is given to the model during code generation. For example, the execution order of state chart transitions may depend on the code generator. Thus understanding the code generator semantics is necessary if one wants to validate the model by running it in a railway simulation — it may be the case that some unexpected behavior originates from the code generator semantics rather than from a model error.

Another questions regards the handwritten code that has been added to the model: Is it integrated into the code generated from the model's diagrams as expected by the modeler? For example, under which conditions are initial and final parentheses generated for a code snippet, and is the handwritten code generated verbatim or is it processed in some way?

We have also seen cases where obviously the code generator produced erroneous code; sometimes the errors result in a compiler message, but sometimes they stay hidden and may be a subtle source of unexpected behavior. We utilized scripts in order to quickly, reliably and reproducibly remove such errors from the code files after each code generation. However, those scripts may require maintenance when the model is changed, so it is desirable to retrieve an updated code generator as soon as possible.

Finally, code generators may not only be capable of generating the mere product code, but also instrumented code for simulation of the model. We decided to use such a feature in order to have animated state charts available during the railway simulation. We did not use it for manipulation of the running model, which might have caused model behavior which is otherwise impossible to stimulate. Another issue is to make sure that the additional code in the instrumented version does not change the behavior of the model (except for intended manipulations by the user) in any way; in our case, the code generator documentation claimed this.

## 6 Preparation of Lab Interfaces

We have discussed the interfaces between model and lab from the model side already in Sect. 4. Of course, the lab needs to provide the counterpart of those interfaces. Since for a lab reuse of such interfaces is desirable, they should follow widespread standards, if available. As mentioned above, in our case for both relevant lab-RBC interfaces such standards exist; the ETCS standard had already been implemented in the lab and was then made available externally via TCP/IP, and the SCI RBC standard has been newly implemented for the RBC lab integration. The interface to the PTC Integrity Modeler was provided by PTC as part of the tool and of the instrumented code (RBC model interface side), based on TCP/IP communication as well.

For the RBC train interface further measures in order to ensure interoperability between lab and RBC have been necessary. This was due to an old version of the ETCS on-board unit which was incompatible with the RBC which was developed according to the newest version. For message decoding on the RBC side, an existing component (external to the model, see Fig. 3) was used which supports the different versions anyway. However, for further processing a version converter was needed. Although officially not foreseen in the ETCS standard, such a conversion was inferred from another ETCS specification ([SS16b], where data forwarding between RBC and a neighboring RBC of different versions is specified).

The use of the external message decoder component brought up the need for specification of that particular interface. It was decided to realize this on TCP/IP basis as well (which did not cause additional asynchronism problems since the messages passed to the decoder are received by the RBC from the on-board unit via TCP/IP right before). Messages were wrapped, adding IDs in order for the RBC to be able to distinguish which message coming back from the decoder has been received from which train previously.

## 7 Preparing a Railway Simulation

In order to run a railway simulation, a particular railway line (track topology and signaling equipment) and operational scenario (train type, train control system, route, course of events) have to be identified (cf. Fig. 2). In our case the single track line from Braunschweig main station to Braunschweig Gliesmarode station and the scenario of a train ride starting in ETCS Level 0 with later entry to Level 2 was chosen since for the latter a connection between train and RBC is set up and used. Since that particular line is not equipped with ETCS in reality, this has been done (mainly positioning of balises and assignment of balise messages) according to the applicable DB guidance document [Ril14].

The line and scenario data are then used to prepare the different parts of the simulation: Configuration files for the RBC model and the lab railway simulation are created which contain distances, signals, points, balises, speed and gradient profiles, etc. The 3D-world for visualization of the line has been available already but was adjusted to include the ETCS

balises. All of those preparations must happen in a consistent way; otherwise the intended scenario will not work properly e.g. due to triggering of automatic train stops or a time shift between simulation visualization and logic. The built-in configuration consistency check that was implemented in the RBC helps to detect faulty RBC configurations.

The connection set-up between the different components from Fig. 3 is dependent on the line and simulation scenario as well. For example, the number of interlockings connected to an RBC may vary. It is also dependent on the deployment of components in the lab; in our case, we needed to configure the IP addresses and ports for the TCP/IP connections between the computers involved, and ensure a suitable network and firewall configuration of all computers.

## 8 Running the Simulation



Fig. 4: Visual output of the railway simulation (driver's view and ETCS DMI on the left) and of the PTC Integrity Modeler state animation (upper right) as well as the current position of the train on the map (lower right). The state chart shows in red the current RBC internal state ("stopping aspect") of the red signal visible in the driver's view.

For running the RBC model as part of the railway simulation, at least all of the components from Fig. 3 that are not purely grey (pertain not to mere visualization) need to be started

up. In general, it is favorable that the component implementations do not create too many dependencies regarding the order of start-up and connect automatically to each other using predefined connection settings; for particular connections, exceptions from that rule may make sense. Afterwards, the components need to be brought to the correct initial state for the simulation scenario. In our case, that meant that a "Start of Mission" procedure bringing the ETCS on-board unit to Level 0 operation had to be executed via the ETCS DMI and that initial information on current signal and point states was transferred automatically from the interlocking module to the RBC after start-up of both components. For finally executing the scenario, in our set-up the train had to be driven manually (speed control lever in mock-up and interaction with ETCS DMI) and also routes had to be set manually (interaction with interlocking module). This offered an easy possibility to try out slight modifications of the original operational scenario. In this context, DMI and front view visualization become mandatory, of course. However, the lab could have been set-up to simulate the precise scenario automatically, if that had been desired.

The simulation can be run with or without the RBC state chart animation. In the first case, the instrumented version of the RBC model code must be generated, compiled and used. The PTC Integrity Modeler must be started (requiring the PTC license) and the RBC model must be loaded (requiring the PTC repository). Coherence must be ensured between the model versions loaded in Integrity Modeler and the one from which the running RBC model was generated. Then Integrity Modeler simulation mode can be started; starting up the RBC model as well will result in a connection between those two which is used to transfer information about newly created objects and changed states from the running model to Integrity Modeler. The latter visualizes that by opening and coloring state charts for those objects. Fig. 4 shows the visual outputs of the lab and the state animation for one scene of the running operational scenario.

## 9 Validation Results and Scope

Here we shortly describe our experiences with the RBC model validation by running it as part of the RailSiTe® railway simulation. We used the visual lab output as well as the state animation and the RBC logging function in order to check whether the scenario proceeded as expected. This way we could detect errors from a wide range of sources: errors in the model, in the added code, in the code generator, in the newly implemented lab interfaces and in the configuration data. Most often it was not difficult to locate the component where the problem originated from; in order to support this, we ran the generated RBC code in a debugging tool. And we could do so step by step, each time advancing a bit further in the simulation scenario. Thus we conclude that validation by simulation is an effective tool which can provide evidence that the generic model as well as the configuration data used for the integration are valid. During validation it pays off if restarting the railway simulation and the model is easily possible.



The clear majority of errors we found was in hand-coded parts (e.g. the code added to the model). Since a good portion of the generated RBC code was generated from the modeled diagrams (estimation: nearly half of the code lines), this confirms that modeling and using code generation increases software quality. We found only very few errors in the modeled diagrams, e.g. a missing state chart transition; here the modeling proved to be advantageous again since it eased the location and correction of such errors. Problems that involved the code generation (different interpretation of model by user and code generator) were comparatively difficult to locate, but occurred only seldom.

## 10 Reflection on the Model in Simulation Approach

As stated in the paper’s abstract, a railway simulation is one possible environment for model validation; other alternatives may include using stubs or running the model in a real environment. Tab. 1 provides a compact comparison. While working with stubs may

Criterion	Stubs	Simulation	Real Environment
Availability	high — automatic generation possible	low — special purpose software	medium — real hardware and software
Integration effort	low — mostly automatic	medium to high — integration effort, depending on available interfaces	medium — installation effort
Testing effort	high — in-/outputs on detailed level	low to medium — in-/outputs on high level	low — stimulation/read-off at external interfaces
Flexibility	high — direct manipulation	medium to high — indirect manipulation on different levels	low — manipulation only at external interfaces
Suited for complex scenarios	low — inputs too detailed	high — supports manipulation on high level	high — supports manipulation on high level
Closeness to reality	low — depends on user	high — realistic components	high — real components

Tab. 1: Advantages and disadvantages of different validation environments for railway signaling component models.

be a good idea for unit and integration testing, they do not provide (realistically) the environment’s logic. On the other side, a real environment does, but may not be available or too costly, especially when it comes to a railway signaling environment. So all in all we think that for the validation of signaling component models railway simulations are a favorable choice.

## 11 Conclusion and Future Work

In this article, we have reported on the validation of a railway signaling component model by integrating it into a railway simulation. More precisely, a SysML model of an RBC has been integrated and validated into DLR's RailSiTe® laboratory.

A first conclusion, from the mere fact that the model could be integrated and ran well in the simulation environment, is the general feasibility of modeling railway signaling components generically and for realistic applications. Further steps to support this statement would be to try the same approach with different components, with different configurations, and with complete component models. Some work remains to be done if the development approach is to be applied for real signaling components: A detailed semantics for SysML (which may be part of the upcoming SysML 2.0 standardization work) would be needed as well as a concept for migration to the new approach and a certified code generator. Most probable, this will imply further restrictions on the model artifacts and programming language constructs used. At least the model at hand was developed to result in deterministic code (i.e. without concurrency) and avoiding circular control flow between the state charts.

Depending on the interfaces available and the configuration of the lab for a particular operational scenario, the integration of a model into a railway simulation may require some effort; certainly, there is potential for more efficient or automated ways to distribute configuration data from a single source to the different components of the simulation. However, we were pleased to see that the combination of model and railway simulation allowed for efficient validation: The model obviously resulted in a high initial quality of the component and provided the possibility for visual inspection during simulation by using the state animation feature; and the simulation allowed for convenient stimulation and output retrieval on the high-level interfaces like DMI and line visualization, while offering the possibility to dig deeper when erroneous behavior was observed. It also turned out that the inherent parallel validation of the configuration data and the newly implemented lab interfaces was not much of a problem since the assignment of errors to the different possible sources was quickly found in most cases, whereas tracking errors related to code generation was more difficult. All in all, we conclude that validation of models by running them in railway simulations is a practically feasible and may even be the favorable approach for overall component validation. Provided that some particular requirements of the approach, which we aimed to describe in this paper, are taken into consideration early, lab integration can be organized in a smooth way.

In the future, we would like to investigate and exploit the benefits of a validated model. It may be used as a reference for real signaling components, or for test case generation for such components. It would be nice to see whether the quality of the component can be increased further e.g. due to the extra model coverage criteria generated test cases can fulfill. We also expect a clear decrease in maintenance effort of test cases (simply change model and generate again).

## References

- [EUL18] EULYNX Website, 2018, URL: <https://www.eulynx.eu>, visited on: 02/19/2018.
- [Ril14] LST-Anlagen planen: Grundsätze zur Erstellung der Ausführungsplanung PT1 für ETCS Level 2, tech. rep. Richtlinie 819.1344, draft, DB Netze AG, Apr. 24, 2014.
- [SCI14] SCI-RBC – FAS TAS TAV Schnittstelle ESTW-RBC V2. Baseline 0.19, tech. rep., DB Netze AG, June 6, 2014.
- [SH16] Schwencke, D.; Hungar, H.: Development of Reference Models of Signaling Components: the Example of the Radio Block Center. *Signal+Draht* 108/9, pp. 24–32, 2016.
- [SHC17] Schwencke, D.; Hungar, H.; Caspar, M.: Between Academics and Practice: Model-based Development of Generic Safety-Critical Systems. In (Huhn, M.; Hungar, H.; Riebisch, M.; Voss, S., eds.): *Modellbasierte Entwicklung eingebetteter Systeme XIII* (Proceedings of the Dagstuhl MBEES 2017 workshop). fortiss GmbH, Munich, pp. 1–18, 2017, URL: [http://download.fortiss.org/public/mbees/mbees2017\\_proceedings.pdf](http://download.fortiss.org/public/mbees/mbees2017_proceedings.pdf).
- [SS16a] SUBSET-026 – System Requirements Specification. Version 3.6.0, tech. rep., ERTMS, June 15, 2016, URL: <http://www.era.europa.eu/Core-Activities/ERTMS/Pages/Set-of-specifications-3.aspx>.
- [SS16b] SUBSET-039 - FIS for the RBC/RBC Handover. Version 3.2.0, tech. rep., ERTMS, June 15, 2016, URL: <http://www.era.europa.eu/Core-Activities/ERTMS/Pages/Set-of-specifications-3.aspx>.