Methoden

Bianca Wiesmayr*, Felix Roithmayr und Alois Zoitl

Ein toolgestützter Ansatz für die benutzerfreundliche Definition von Funktionsbaustein-Einschränkungen

A tool-assisted approach for user-friendly definition of FB constraints

https://doi.org/10.1515/auto-2023-0173
Empfangen September 18, 2023; angenommen Dezember 6, 2023

Zusammenfassung: An Steuerungsprogramme werden hohe Anforderungen hinsichtlich Korrektheit und Robustheit gestellt. Moderne Werkzeuge können Entwickler*innen bei der Erstellung von Programmen unterstützen. Der Schwerpunkt dieses Artikels liegt auf der expliziten Modellierung von implizitem Wissen der Entwickler*innen, da diese Informationen für die automatische Überprüfung von Programmen nötig sind. In diesem Artikel stellen wir ein Konzept zur Definition von Einschränkungen an der Schnittstelle von Funktionsbausteinen dar. Diese sollen Annahmen über das Verhalten anderer Komponenten oder der physischen Umgebung festhalten. Diese Einschränkungen werden für die Erzeugung von Überwachungsbausteinen genutzt.

Schlagwörter: Verhaltensmodelle; IEC 61499; Werkzeuge

Abstract: High demands regarding correctness and robustness are placed on control programs. Modern tools can support developers in the creation of programs. The focus of this article is on the explicit modeling of implicit knowledge of developers, as this information is necessary for the automatic validation of programs. In this article, we present a concept for defining constraints at the interface of function modules. These are intended to capture assumptions about the behavior of other components or the physical environment and form the basis for generated monitoring blocks.

Keywords: behavior models; IEC 61499; modeling tool

***Korrespondenzautor: Bianca Wiesmayr**, LIT CPS Lab, Johannes Kepler Universität Linz, Linz, Österreich, Austria, E-mail: bianca.wiesmayr@jku.at. https://orcid.org/0000-0002-9512-5249
Felix Roithmayr, Johannes Kepler Universität Linz, Linz, Österreich, Austria, E-mail: felix.roithmayr@jku.at
Alois Zoitl, LIT CPS Lab, CDL VaSiCS, Johannes Kepler Universität Linz, Linz, Austria, E-mail: alois.zoitl@jku.at

1 Einleitung

Konzepte wie Modularisierung, Abstraktion und ereignisgesteuerte Programmierung helfen, die Komplexität von Software in cyber-physischen Produktionssystemen zu beherrschen [1]. Vor diesem Hintergrund wurden domänenspezifische Modelliersprachen entwickelt, beispielsweise für eingebettete Systeme [1,2]. Verschiedene Ansätze erlauben weiters die Entwicklung von verteilten Anwendungen in der Automatisierungstechnik. Auch die in der IEC 61499[1] definierte Modelliersprache legt den Schwerpunkt auf die Verteilbarkeit und die hierarchische Strukturierung von Steuerungssoftware. Die wiederverwendbaren Komponenten (sogenannte Funktionsbausteine, FB) können wiederum aus einer beliebigen Anzahl von FB bestehen. Alternativ kann die Funktionalität eines FB als Zustandsdiagramm oder in einer der Programmiersprachen aus IEC 61131-3[2] implementiert werden. IEC 61499 sieht keine globalen Variablen vor und lässt Interaktionen zwischen FB nur über deren Schnittstellen zu, was die Wiederverwendbarkeit fördert [2]. Die Funktionalität der Steuerungssoftware wird nach IEC 61499 im Applikationsmodell definiert, das sich aus Instanzen von Bibliotheksbausteinen sowie aus Verbindungen zusammensetzt (Funktionsbausteindiagramm). Die Norm definiert die ereignisgesteuerte Ausführung des Applikationsmodells, wodurch eine gezielte Steuerung der Ausführungsreihenfolge von FB ermöglicht wird. Implizite Annahmen über die korrekte Verwendung eines FB werden derzeit jedoch nicht systematisch erfasst, weshalb die (Wieder-)verwendung von komplexen Bibliotheksbausteinen eine umfangreiche Dokumentation oder Expert*innenwissen erfordert. Das Fehlen solcher Informationen erschwert auch die Bereitstellung effizienter

1 International Electrotechnical Commission (IEC), IEC 61499 Part 1 – Function Blocks, Edition 2, 2012.

2 International Electrotechnical Commission (IEC), IEC 61131 Part 3 – Programming Languages, 2013.

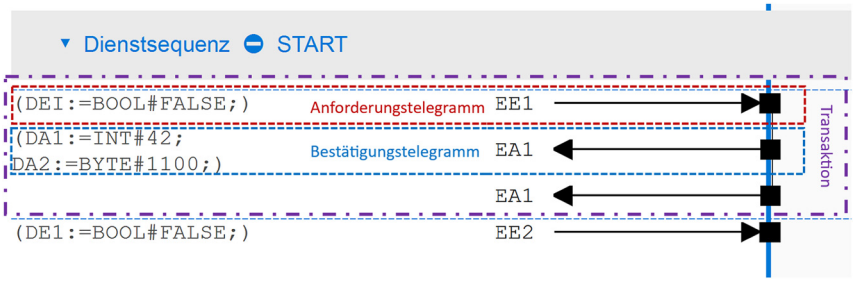

Abbildung 1: Struktur einer Dienstsequenz, die den Ereignisfluss an der äußeren Schnittstelle eines FB erfasst. Die Ereigniseingänge EE1, EE2 und der Ereignisausgang EA1 sind erfasst. Jede Transaktion beinhaltet ein Anforderungstelegramm und eine beliebige Anzahl an Bestätigungstelegrammen. Jedes Telegramm besteht aus einem Ereignis und Parametern.

Entwicklungswerkzeuge. Die Wiederverwendung funktionierender Teillösungen erleichtert allgemein die effiziente Erstellung korrekter Software [3].

Die IEC 61499 unterstützt die Modellierung des erwarteten Verhaltens eines FB im Dienstsequenzdiagramm. Diese beschreiben jene Ereignisse, die an der Schnittstelle beobachtet werden können. Eine Beispielsequenz ist in Abbildung 1 dargestellt. Jedes Ereignis wird als Telegramm modelliert. Die IEC 61499 beschreibt ein Anforderungstelegramm als Aufruf eines im FB enthaltenen Algorithmus. Es ist Teil einer Transaktion, zusammen mit einem oder mehreren Bestätigungstelegrammen. Telegramme können grundsätzlich auch Datenwerte enthalten (sogenannte Parameter). Die normierte Semantik für Parameter beschränkt sich auf Ereignisse, die mit einer booleschen Eingangsvariable kombiniert werden. Für andere Ereignisse sollen die Namen der relevanten Variablen als Parameter des Telegramms spezifiziert werden.

Derzeit sind Modelle des erwarteten Bausteinverhaltens nicht in den Entwicklungsprozess von IEC 61499-Software integriert, sondern dienen bestenfalls als Dokumentation. Die in der Praxis seltene Verwendung von Dienstsequenzdiagrammen könnte durch (halb)automatische Erfassung von Verhaltensdiagrammen verbessert werden. Daher wollen wir Entwickler*innen dabei unterstützen, geforderte oder verbotene Abläufe für einen FB festzulegen. Dazu verwenden wir eine im Vergleich zum Standard IEC 61499 angepasste Variante von Dienstsequenzmodellen. Dadurch unterstützen wir die bereits vorgestellten Konzepte, die auf Dienstsequenzmodellen beruhen und diese Anpassungen teilweise bereits umgesetzt haben. Basierend auf zufällig generierten und/oder manuell definierten Eingangsereignissen analysieren wir das Verhalten des FB und visualisieren es als Dienstsequenz. Entwickler*innen klassifizieren die so erstellten Modelle. Die komponentenorientierte Architektur erlaubt weiters bei Bedarf eine Erweiterung auf andere Notationen von Sequenzdiagrammen.

Unsere Hauptbeiträge sind (i) eine Adaption der IEC 61499, um die Klassifizierung von Dienstsequenzen zu erlauben, (ii) der vorgestellte Ansatz zur benutzerfreundlichen Definition von FB-Einschränkungen als Dienstsequenz und (iii) die Möglichkeit zur automatischen Erzeugung von Überwachungsbausteinen aus Dienstsequenzen. Der Ansatz basiert auf unserer Infrastruktur zur Interpretation von IEC 61499-Modellen [4] und ist kompatibel mit allen in der IEC 61499 definierten Arten von FB. Der vorliegende Artikel erweitert den Konferenzbeitrag [5]. Alle entwickelten Werkzeuge werden als Teil des Open-Source-Projekts Eclipse 4diac™[3] veröffentlicht.

2 Wissenschaftlicher Hintergrund

In der Automatisierungstechnik verwendete Modellierspracheh sind die Unified Modeling Language (UML), die davon abgeleitete Systems Modeling Language (SysML) sowie domänenspezifische Modellierspracheh [6]. Die Adaption einer bekannten Modellierspracheh bringt Vorteile hinsichtlich der Akzeptanz durch Nutzer*innen und der Verfügbarkeit von Modellierwerkzeugen im Vergleich zur Einführung gänzlich neuer Ansätze [6]. Die in diesem Artikel vorgestellte Methode zur Definition von FB-Einschränkungen verwendet daher eine Variante der in der IEC 61499 normierten Dienstsequenzdiagramme.

Dienstsequenzdiagramme können bei der Erstellung von IEC 61499-Modellen für Synthese [7], Test [8] oder Überwachung [9] von Funktionalität genutzt werden. Die Diagramme beschreiben in erster Linie, wie ein Dienst aufzurufen ist. Dorofeev et al. [7] synthetisierten einen Orchestrierungs-FB, welcher untergeordnete Funktionalitäten gemäß der in den jeweiligen Bausteintypen spezifizierten Dienstsequenzen aufruft. Hametner et al. [8] realisierten ein Konzept zur Nutzung von Dienstsequenzmodellen als

3 4diac IDE 3.0 (Entwicklungsversion), www.eclipse.org/4diac.

Grundlage für Softwaretests. Erwartete Datenwerte werden im Parameterfeld festgelegt, wobei eine im Vergleich zur Norm leicht angepasste Schreibweise verwendet wird. Die Autoren merkten die eingeschränkte Ausdrucksfähigkeit von Dienstsequenzdiagrammen an und schlugen eine Erweiterung mit zusätzlichen Sprachelementen vor. Die vorgeschlagenen Möglichkeiten zum Aufruf von Unterprogrammen, Schleifen und Verzweigungen sind zum Beispiel in den Sequenzdiagrammen der UML[4] vorhanden. Wenger et al. [9] beschrieben eine Methode zur Überwachung des Bausteinverhaltens direkt in der Ausführungsumgebung, wobei das erwartete Verhalten ebenfalls als Dienstsequenzmodell vorliegt. Wiederum enthält das Parameterfeld erwartete Datenwerte. Darüber hinaus werden auch Wertebereiche und Zeitangaben modelliert. Alle vorgestellten Arbeiten gehen von einer manuellen Erstellung der Dienstsequenzdiagramme aus. Eine Reduktion des Modellierungsaufwands erhöht daher auch die Nutzbarkeit bestehender Ansätze. Mithilfe von Werkzeugen ermittelte Einschränkungen könnten daher einen wertvollen Beitrag zu den beschriebenen Methoden leisten.

Steuerungssysteme sind für die Interaktion mit der realen Welt konzipiert. Bei solchen Systemen hängen realistische Ausführungsszenarien von der Umgebung ab. Infolgedessen schlugen Prenzel und Steinhorst [10] Szenariomodelle als Grundlage für die Planung von Software-Updates für IEC 61499-Anwendungen vor. Darüber hinaus ist die Ermittlung von Parametern wie der Worst-Case-Ausführungszeit (WCET) unerlässlich für zeitkritische Systeme. Szenarien bieten ein Mittel zur Abschätzung der WCET, da sie die Reaktion auf eine Dienstanforderung erfassen. Lednicki et al. [11] haben derartige Transaktionen nicht als Dienstsequenz visualisiert, die in ihrem WCET-Modell erfassten Informationen sind aber ähnlich.

Allgemein können Sequenzdiagramme Nutzungsszenarien von Systemen und FB beschreiben. In einem modellbasierten Ansatz nutzen Panjaitan und Frey [12] UML-Diagramme, aus denen sie automatisiert IEC 61499-Software generieren. Sie setzen Sequenzdiagramme sowohl zur Beschreibung von Schnittstellen zu physischen Komponenten als auch zur Beschreibung von Szenarien der Ausführung in einer bestimmten Betriebsart ein. Das Sequenzdiagramm zeigt in diesem Fall die Interaktionen zwischen Komponenten für einen bestimmten Ablauf. Sequenzdiagramme können Informationen speichern, die nicht Teil der ausführbaren Software werden.

Unser Konzept zielt auf Steuerungssoftware ab, die sich noch in der Entwicklung befindet, weshalb die nötigen Informationen in erster Linie durch Simulation gewonnen werden. Die Visualisierung als Dienstsequenz wird gewählt, um die Möglichkeiten der IEC 61499 auszuschöpfen. Die komponentenbasierte Architektur ermöglicht grundsätzlich jedoch auch, dass die Ausführungsprotokolle in anderen Diagrammen dargestellt werden.

3 Identifizierte Herausforderungen

Die Norm IEC 61499 enthält keine umfassende Sprachdefinition für Dienstsequenzmodelle. Anforderungen für verschiedene Verwendungen wurden in einer früheren Arbeit zusammengefasst [13] und werden hier diskutiert, sofern sie die Definition von FB-Einschränkungen betreffen.

H1 **Modellierung der erwarteten Datenwerte** Die Norm berücksichtigt nur die Anforderung und Bestätigung von Diensten. Selbst bei dieser engen Auslegung ist die Angabe von erwarteten Datenwerten für die korrekte Anforderung eines Dienstes mit Parametern unerlässlich. Für die Namen der einem Ereignis zugeordneten Variablen ist das Parameterfeld vorgesehen. Datenwerte sind nicht berücksichtigt.

H2 **Modellierung von Vorbedingungen** In Dienstsequenzdiagrammen ist keine Angabe von Vorbedingungen vorgesehen. Vordefinierte Anfangszustände sind besonders nützlich bei der Modellierung von Szenarien, die nach dem Auftreten eines Fehlers beginnen. Manche Arbeiten gehen davon aus, dass jede Dienstsequenz mit dem Anfangzustand beginnt (z. B. [14]), was für die Definition von FB-Einschränkungen nicht praktikabel ist. Für einen FB könnte zum Beispiel gelten, dass zwei Ereignisse immer direkt nacheinander auftreten. Eine solche verpflichtende Bedingung wäre in allen Zuständen des FB gültig, nicht nur als Teil einer bestimmten Dienstsequenz. Falls die Ausgangsereignisse vom Zustand des FB abhängen, müssen sie als separate Dienstsequenzen mit den entsprechenden Vorbedingungen angegeben werden.

H3 **Klassifizierung von Szenarien** Dienstsequenzen definieren mögliche Szenarien für die Verwendung eines FB. Es ist unklar, ob ein Szenario für alle Ausführungen, nur für eine Teilmenge davon oder für keine der Ausführungen gültig ist. Die Speicherung dieser Informationen ist für die Verwendung von Dienstsequenzen als Mechanismus zur Erfassung von Beschränkungen unerlässlich.

H4 **Hoher Modellierungsaufwand** Die Erstellung von Dienstsequenzen für bestehende FB bedeutet einen hohen Aufwand für Entwickler*innen. Obwohl dessen Funktionalität bereits implementiert ist, soll das

4 Unified Modeling Language, ed. 2.5.

erwartete Verhalten spezifiziert werden. Das gleiche Verhalten wird also aus verschiedenen Sichten modelliert. Einige der verfügbaren Entwicklungsumgebungen erzwingen eine Reihenfolge der Operationen bei der Modellierung von Funktionsblöcken; nur Ereignisse, die in der Schnittstelle angegeben wurden, können als Ereignis für das Telegramm ausgewählt werden. Dies kann sicherstellen, dass die erstellten Modelle gültig sind, kann aber die Nützlichkeit von Dienstsequenzen für die anfängliche Spezifikation des erwarteten Verhaltens weiter verringern.

H5 **Nötige Softwareentwicklungs-Kenntnisse** Die zunehmende projektübergreifende Wiederverwendung von Komponenten erhöht auch den Nutzen von Dokumentation, Einschränkungen und Testfällen, die für eine Komponente spezifiziert werden. Die Definition von FB-Einschränkungen erfordert jedoch Erfahrung in der Softwareentwicklung. Viele Annahmen an das Umgebungsverhalten erscheinen offensichtlich und werden daher möglicherweise nicht gespeichert. Dennoch könnten sie nützlich sein, insbesondere wenn eine Komponente in anderen Kontexten wiederverwendet wird als ursprünglich vorgesehen.

4 Szenariomodelle

Um die im vorigen Abschnitt genannten Herausforderungen vollumfänglich zu adressieren, sind einerseits geringfügige Erweiterungen der IEC 61499 erforderlich, andererseits werden bessere Entwicklungswerkzeuge benötigt. Wir diskutieren nun Szenariomodelle sowohl in allgemeinen Modelliersprachen als auch in der domänenspezifischen IEC 61499, um daraus Erweiterungen für die Norm zu identifizieren, die für die Spezifikation von Einschränkungen geeignet sind.

4.1 Allgemeine Szenariosprachen

Szenarien sind vor allem in frühen Entwicklungsphasen von Vorteil, wenn isolierte Fragmente des Gesamtverhaltens modelliert werden können. Für die Entwicklung kompletter Systeme mit zahlreichen Szenarien, die Teilergebnisse modellieren, ist eine effiziente Werkzeugunterstützung zur Identifizierung der gültigen und verbotenen Ausführungsszenarien eines Gesamtsystems unerlässlich [15].

Sequenzdiagramme beschreiben Szenarien und ermöglichen die Spezifikation von verbotenen Verhaltensweisen oder Testfällen. Sie zeigen die Interaktionen, die zwischen Komponenten entlang deren Lebenslinien stattfinden. Es wurden verschiedene Semantiken und Notationen für Sequenzdiagramme vorgeschlagen. Message Sequence Charts (MSCs) sind in der UML definiert. Sie enthalten Sprachelemente für Alternativen, Synchronisationen und Schleifen. Obwohl die grafische Notation normiert ist, sind verschiedene Interpretationen ihrer Bedeutung (Semantik) möglich, sodass unterschiedliche Anwendungsfälle unterstützt werden. Live Sequence Charts (LSCs) wurden zur Erfassung von Anforderungen entwickelt. Im Gegensatz zu MSCs erlauben sie die Definition von verpflichtendem und verbotenem Verhalten [16]. In LSCs zeigt ein verpflichtendes Verhalten an, dass ein bestimmtes Szenario letztendlich eintreten muss. Wendet man dies auf verschiedene Betriebsarten wie Initialisierung oder Normalbetrieb eines Systems an, könnte festgelegt werden, dass die Initialisierung letztendlich abgeschlossen werden muss. Der Normalbetrieb würde daher ein verpflichtendes Verhalten darstellen. Weiters können zeitliche Einschränkungen eine minimale und maximale Verzögerung zwischen zwei Aktionen in einem LSC angeben. Eine Zeitüberschreitung weist auf einen Fehler hin. Auch diese Art von unerwünschtem Verhalten kann als verbotenes Szenario modelliert werden.

4.2 Dienstsequenzen in der IEC 61499

Im Gegensatz zu LSCs definieren die Dienstmodelle der IEC 61499 mögliche Szenarien einer Software ohne weitere Klassifizierung. Wie in der Einleitung beschrieben, konzentrieren sich Dienstsequenzen auf Dienstanfragen und -antworten. Sie werden als eine Reihe von Diensttransaktionen modelliert. Jede Diensttransaktion beginnt mit dem Anforderungstelegramm, bestehend aus einem Eingabeereignis und den zugehörigen Datenvariablen (vgl. Abbildung 1). Anschließend werden in einem Bestätigungstelegramm die Ausgangsereignisse, die als Reaktion auf dieses Eingangsereignis ausgelöst werden und ihre zugehörigen Datenvariablen aufgeführt. Eine Dienstsequenz legt die Reihenfolge der Ereignisse fest, aber keine absoluten Zeitvorgaben.

Die Lösung der in Abschnitt 3 (**H1**–**H5**) dargestellten Herausforderungen erfordert die Erweiterung der Notation von Dienstsequenzen um Vorbedingungen, einen Mechanismus zur Angabe von Daten und die Klassifizierung, ob das Verhalten erwünscht ist.

Das Feld **Parameter** wird für die Angabe von Datenwerten (**H1**) als Teil von Anforderungs- und Bestätigungstelegrammen verwendet. Die Norm gibt an, dass die Namen der relevanten Variablen aufgeführt werden sollten, beschränkt die Verwendung des Parameterfeldes jedoch nicht ausdrücklich auf Namen. Infolgedessen haben [8,9] eine Notation von Variablenzuweisungen für das `parameter`-Feld eines Telegramms eingeführt, wie zum

Beispiel `DI1:= FALSE;DI2:= 5`. Diese Notation wurde auch in dem Beispiel in Abbildung 1 verwendet, aber alle Werte wurden zusätzlich nach den Regeln für IEC 61131-3-Literale formatiert. 4diac IDE behandelt das Parameterfeld als Textfeld und ermöglicht dies.

Vorbedingungen (H2) enthalten initiale Datenwerte und den Anfangszustand des FB. Wir führen ein eigenes Attribut als XML-Element ein, um den Namen des Startzustands anzugeben. Darüber hinaus können die gewünschten Anfangswerte als Parameter des ersten Anforderungstelegramms angegeben werden. Ein eigenes XML-Element zur Speicherung der Vorbedingung jeder Dienstsequenz ist wünschenswert. Dies ermöglicht die Trennung von Datenwerten, die aus der Umgebung des FB empfangen werden (die Teil des Anforderungstelegramms sind), von anderen Variablen (die eine Voraussetzung für die Gültigkeit der Dienstsequenz sind).

Wir schlagen die Verwendung der **Klassifikations**-Tags "verpflichtend", "erlaubt", "verboten" und "bedingt" vor (**H3**). Eine verpflichtende Sequenz muss irgendwann eintreten, damit ein System gültig ist, während eine verbotene Sequenz niemals eintreten darf. Bedingte Dienstsequenzen müssen gelten, wenn die Vorbedingungen erfüllt sind. Eine mögliche Sequenz ist der Standardfall, der die aktuelle Definition der Norm widerspiegelt. Die Vorbedingungen der Sequenz sind unbekannt oder nicht modelliert und es ist auch möglich, dass die Sequenz in der Praxis nie auftritt.

Damit Entwickler*innen Dienstsequenzen als Modellierungswerkzeug in Betracht ziehen (**H4** and **H5**), muss der Nutzen der Erstellung den Aufwand überwiegen. In unserer früheren Arbeit haben wir einen Modellinterpreter für FB entwickelt und ihn für die automatische Aufzeichnung von Ausführungsprotokollen und deren Konvertierung in Dienstsequenzmodelle eingesetzt, um einen ersten Schritt in Richtung einer werkzeuggestützten Generierung von Verhaltensmodellen zu erreichen [4]. Um eine Sequenz aufzuzeichnen, müssen Entwickler*innen eine Liste von Eingangsereignissen und den gewünschten Anfangszustand des Bausteins angeben. Solche Empfehlungen des Werkzeugs reduzieren das erforderliche Wissen der Entwickler*innen über Softwareentwicklungsmethoden.

5 Architektur zur Klassifikation

Mit den im vorigen Abschnitt vorgeschlagenen Erweiterungen können Dienstsequenzen als Notation für Einschränkungen des erwarteten Verhaltens einer FB-Schnittstelle dienen. Eine werkzeuggestützte Spezifikation kann den Entwicklungsaufwand für die Definition solcher Einschränkungen erheblich reduzieren (**H4**). Die in diesem Abschnitt vorgeschlagene Architektur ermöglicht die werkzeuggestützte Aufzeichnung von Einschränkungen mit benutzerdefinierten Eingangswerten für einen Bausteintyp. Ein Überblick

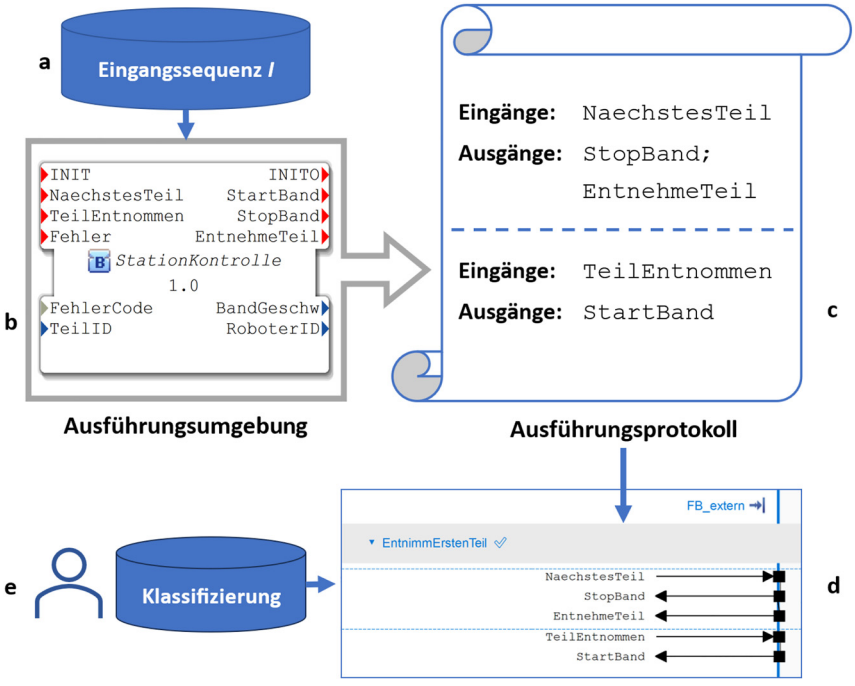

Abbildung 2: Methode zur Klassifikation einer Dienstsequenz. Für eine *a* Abfolge an Eingangsdaten berechnet eine *b* Laufzeitumgebung die *c* Ergebnisse der Ausführung. Basierend auf den *d* Ausführungsprotokollen wird das Verhalten als *Dienstsequenzmodell* visualisiert. Ein*e Entwickler*in *e* klassifiziert die Dienstsequenz, welche dann als Teil des Bausteintyps gespeichert wird.

über die Architektur ist in Abbildung 2 dargestellt. Zur Analyse des Bausteinverhaltens ist eine Ausführungsumgebung für IEC 61499-Modelle erforderlich.

Entsprechend der Nomenklatur von Dubinin und Vyatkin [17] kann die Schnittstelle eines Bausteintyps formal als ein Tupel (EI^0, EO^0, VI^0, VO^0, IW, OW) beschrieben werden. Hierbei bezeichnet:

$\mathrm{EI}^0 = \{ei_1^0, ei_2^0, \ldots, ei_k^0\}$ die Menge an Ereigniseingängen;

$\mathrm{EO}^0 = \{eo_1^0, eo_2^0, \ldots, eo_l^0\}$ die Menge an Ereignisausgängen;

$\mathrm{VI}^0 = \{vi_1^0, vi_2^0, \ldots, vi_m^0\}$ die Menge an Dateneingängen;

$\mathrm{VO}^0 = \{vo_1^0, vo_2^0, \ldots, vo_n^0\}$ die Menge an Datenausgängen.

IW und OW sind Mengen sogenannter WITH-Verbindungen zwischen Ereigniseingängen und Dateneingängen, beziehungsweise Ereignisausgängen und Datenausgängen. Ein Eingangsereignis $\mathrm{ei}_{\mathrm{trig}}^j$ ist ein Signal, das am Ereigniseingang ei_j^0 auftritt. Ein Eingangsereignis $\mathrm{ei}_{\mathrm{trig}}^j$ oder eine Folge an Eingangsereignissen,

$$\mathrm{EI}_{\mathrm{trig}} = \left\{ \mathrm{ei}_{\mathrm{trig},0}^{j_0}, \mathrm{ei}_{\mathrm{trig},1}^{j_1}, \ldots, \mathrm{ei}_{\mathrm{trig},n}^{j_n} \right\},$$

zusammen mit den zugehörigen Dateneingängen $\mathrm{vi}_i \subseteq \mathrm{VI}$, dienen als Eingangsinformationen für das Klassifikationssystem:

$$I = \{i_0, i_1, \ldots, i_o\}, \quad i_i = \left(\mathrm{ei}_{\mathrm{trig},i}^j, \mathrm{vi}_i\right).$$

Jeder Ereigniseingang hat zugehörige Dateneingänge, die sich wie Parameter eines Methodenaufrufs verhalten und als Teil der Menge IW angegeben werden. Die zu den Dateneingängen vi_i zugehörigen Werte müssen für jedes Eingangsereignis angegeben werden.

In einer Laufzeitumgebung kann die Reaktion eines FB auf eine Eingangssequenz erfasst werden. Hierbei können durch ein einzelnes Eingangsereignis auch mehrere Ausgangsereignisse erzeugt werden. Nachfolgend werden mögliche Quellen für die Eingangssequenz erläutert:

Benutzerdefinierte Eingangsdaten Kombinationen aus Eingangsereignissen und Datenwerten können manuell festgelegt werden, um bekannte, wichtige Szenarien zu erfassen. Außerdem können so bestimmte Einschränkungen bereits anhand des unfertigen Bausteintyps definiert werden.

Zufällige Eingangsdaten Für jeden Datensatz wird ein Eingangsereignis zusammen mit Werten für die zugehörigen Dateneingänge zufällig ausgewählt. Zufällige Eingabesequenzen helfen bei der Identifizierung von Szenarien, die für Entwickler*innen intuitiv sind und daher bei der manuellen Spezifikation nicht berücksichtigt werden. Darüber hinaus sind einige Szenarien spezifisch für den Anwendungsfall einer Komponente und können in dem Kontext, für den eine Komponente ursprünglich entwickelt wurde, nicht relevant sein. Dies gilt insbesondere für verbotene Szenarien, wenn Entwickler*innen aus Erfahrung wissen, dass eine bestimmte Reihenfolge von Ereignissen im realen System nicht vorkommt. Daher eignen sich zufällige Eingaben am besten zur Erfassung von Szenarien für unerwartete Anwendungsfälle einer Komponente und von Szenarien, die Entwickler*innen nicht in Betracht ziehen (aufgrund von Unvollständigkeit der Spezifikation, mangelnder Schulung oder Zeit). Die identifizierten Szenarien können jedoch auch weniger relevant sein.

Laufzeitanalyse Software von lauffähigen Gesamtsystemen kann während ihrer Ausführung überwacht werden. Die Laufzeitanalyse eignet sich am besten für die Identifizierung von Szenarien, die in einem bestimmten Anwendungsfall häufig auftreten, erfordert aber vollständige und betriebsbereite Systeme. Im Gegensatz zur zufälligen Generierung kann sie Fehlerszenarien nicht schon vor ihrem Auftreten simulieren.

Gemäß der Norm muss die Ausführungsumgebung des FB sicherstellen, dass immer nur ein Ereignis an einem FB eintrifft. Weiters müssen die in Abschnitt 4.2 eingeführten Vorbedingungen berücksichtigt werden. Der in der Vorbedingung definierte Ausgangszustand muss eingestellt werden, bevor das erste Eingangsereignis zugestellt wird.

Ausführung in einer Laufzeitumgebung Viele Entwicklungsumgebungen exportieren Bausteintypen als Code in Allzwecksprachen. Nach dem Kompilieren dieses Codes können die Bausteininstanzen in der Laufzeitumgebung ausgeführt werden. Der Export der Ausführungsprotokolle muss möglich sein.

Ausführung in einem Interpreter Ein Interpreter modifiziert das Modell direkt ohne vorherigen Export. Er ermöglicht die Ausführung unvollständiger Modelle und den Zugriff auf interne Daten im Gegensatz zu einer Laufzeitumgebung wie 4diac FORTE. So kann beispielsweise ein bestimmter Startzustand gemäß der definierten Vorbedingungen eingestellt werden. Durch Interpretieren eines teilweise fertigen Modells kann die Definition von FB-Einschränkungen frühzeitig erfolgen. Ist beispielsweise die Einschaltlogik noch nicht fertig implementiert, kann durch Setzen des entsprechenden Zustands häufig dennoch eine Analyse der weiteren Betriebszustände erfolgen.

Zur Modellerstellung sind *Ausführungsprotokolle* erforderlich, in denen die Ausgangsereignisse festgehalten werden, die durch eine bestimmte Eingangssequenz I erzeugt wurden. Diese Ausführungsprotokolle müssen in ein Modell umgewandelt werden, das die Einschränkungen in einem Szenariomodell grafisch darstellt. In unserem Kontext werden Dienstsequenzen verwendet, eine Erweiterung auf andere Szenariomodelle, wie die in Abschnitt 4.1 besprochenen, ist aber grundsätzlich möglich. Schließlich wird das grafische Modell angezeigt und durch den Entwickler oder die Entwicklerin *klassifiziert*.

6 Werkzeugunterstützung

Wir haben eine Werkzeugunterstützung für die benutzerfreundliche Definition von FB-Einschränkungen als Sicht in 4diac IDE implementiert, die in Abbildung 3 gezeigt ist. Während im Bausteineditor oben die Schnittstelle oder die Implementierung angepasst werden, erlaubt die untere Sicht die Konfigurierung und Klassifikation von Dienstsequenzen. Durch Auswahl der entsprechenden Kategorie wird die aktuelle Dienstsequenz gespeichert und eine neue wird geladen und angezeigt. Eine Dienstsequenz kann auch übersprungen werden, da insbesondere bei zufällig generierten Modellen die Redundanz hoch ist.

Die Implementierung folgt der zuvor beschriebenen komponentenorientierten Architektur. Dadurch bleiben alle Elemente austauschbar und können beliebig erweitert werden. Folgende Alternativen wurden zunächst realisiert:

Eingangssequenzen können manuell definiert oder zufällig erzeugt werden. Auch manuell definierte Teilsequenzen ergänzt durch Zufallssequenzen sind möglich. Die Länge der Eingangssequenz ist auswählbar.

Als Laufzeitumgebung wird ein Modellinterpreter ohne Echtzeitfähigkeit eingesetzt.

Ausführungsprotokolle liegen zunächst als Datenstruktur vor. Sie werden dann gemäß den Übersetzungsregeln aus einer früheren Arbeit [4] in Dienstsequenzmodellen grafisch visualisiert.

Die Klassifizierung erfolgt über die entwickelte Sicht auf Basis der vorgeschlagenen Kategorien.

Die Dienstsequenzen mit Ereignissen und Daten können direkt im normierten XML-Format gespeichert werden. Die Norm IEC 61499 definiert auch einen Namen und ein Kommentarfeld für diese Sequenzen. Die Klassifizierung und die Vorbedingung (d. h. der Startzustand) sind jedoch benutzerdefinierte Erweiterungen, die für unseren Prototyp implementiert wurden. Für die Speicherung dieser Informationen haben wir den Mechanismus der Attribute gewählt. Das XML-Element `Attribute` ist für die Speicherung von werkzeugspezifischen Erweiterungen, wie grafischen Visualisierungshinweisen, vorgesehen. Die aktuelle Implementierung schränkt daher die Portabilität der Typdefinition ein. Andere Entwicklungsumgebungen würden verbotene Szenarien nicht korrekt erkennen. Die Verwendung der Dienstsequenzen ohne weitere Bearbeitung würde zu fehlgeschlagenen Testfällen führen. Für die Anwendung unserer Methode sind neue XML-Elemente für Vorbedingungen und die Klassifizierung in weiteren Ausgaben der Norm erforderlich.

Nach der Speicherung können Entwickler*innen Dienstsequenzen im grafischen Typeneditor manuell bearbeiten. Dies erlaubt die Verallgemeinerung von Dienstsequenzen. In manchen Fällen können die spezifischen Datenwerte entfernt werden, sodass der gesamte Wertebereich des Datentyps abgedeckt ist.

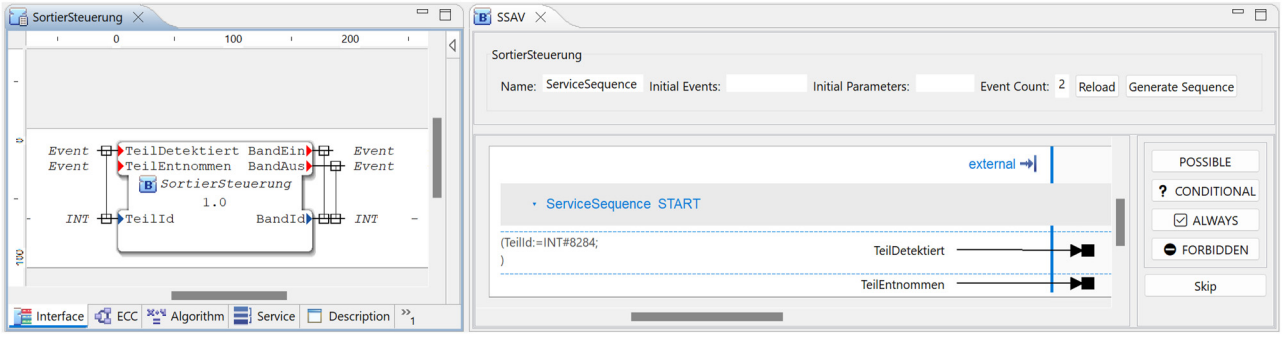

Abbildung 3: Überblick über 4diac IDE. Der Editor (*links*) zeigt die Schnittstellenbeschreibung des FB `SortierSteuerung`. Die Klassifikationssicht (*rechts*) erlaubt die Aufzeichnung, Beurteilung und Speicherung von Dienstsequenzen dieses Bausteins. Die Sicht kann als eigenes Fenster herausgelöst werden, um die gleichzeitige Ansicht verschiedener Informationen zu erlauben. Entwickler*innen können eine gewünschte initiale Ereignisabfolge oder Datenwerte angeben.

7 Anwendungsfall: Überwachung der Ausführung

Der Nutzen von FB-Einschränkungen ergibt sich vor allem aus der Integration dieser Informationen in den Softwareentwicklungsprozess für FB. Sie können beispielsweise zur Fehlererkennung genutzt werden.

Wenn eine verbotene Dienstsequenz während der Ausführung beobachtet wird, liegt möglicherweise ein Fehler des Steuerungssystems vor. Aus Arbeiten zur Softwarearchitektur von Steuerungssoftware ist bekannt, dass die Fehlererkennung von der Implementierung des Verhaltens getrennt werden sollte. Diese kann als eigener FB realisiert werden [18]. Um eine Abfolge von Ereignissen zu erkennen, muss die Historie als Zustand des FB gespeichert werden. Dafür sind insbesondere Basisfunktionsbausteine geeignet, da ihre Funktionalität als Zustandsdiagramm implementiert wird.

Als möglicher Anwendungsfall wird hier die automatische Generierung von Bausteinen zur Überwachung des FB-Verhaltens während der Ausführung besprochen. Die als verboten klassifizierten Dienstsequenzen dürfen nicht auftreten. Soweit dies durch die Implementierung des FB selbst sichergestellt ist, dienen die definierten Szenarien als Ausgangsbasis für Testfälle. Zum Teil handelt es sich aber auch um Annahmen über den Einsatz des FB. Ein Beispiel stellt eine als verboten klassifizierte Folge von Eingangsereignissen dar, die durch physische Komponenten ausgelöst werden könnten. Sollte eine solche verbotene Abfolge dennoch auftreten, liegt ein Fehler vor, auf den das System reagieren muss.

Anhand eines einfachen Beispiels zeigen wir für verbotene Sequenzen die Erstellung eines Basisfunktionsbausteintyps zur Überwachung der Ausführung (Abbildung 4). Der FB `SortierSteuerung` soll das zur `TeilId` gehörige Förderband aktivieren. Ein Teil der Kategorie 1 soll

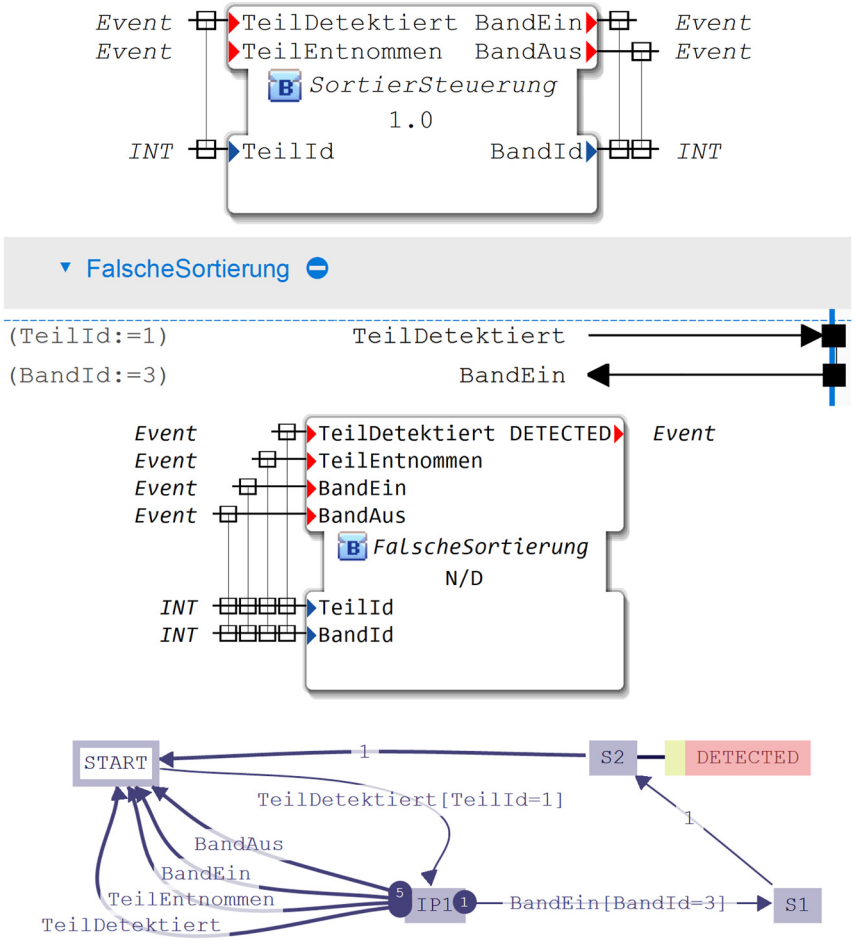

Abbildung 4: Generierung eines Bausteins zur Erkennung einer falschen Sortierung in Zusammenhang mit der SortierSteuerung. Das verbotene Szenario ist in der Dienstsequenz modelliert. Der Überwachungsbaustein und die Implementierung des Zustandsdiagramms (*unten*) wurden automatisch erzeugt.

demnach nicht an Band 3 verteilt werden, was als falsche Sortierung modelliert wird (Dienstsequenz). Für dieses Szenario soll nun ein Überwachungsbaustein erzeugt werden. In Abbildung 4 sind die Schnittstelle und das Zustandsdiagramm des FB `FalscheSortierung` gezeigt.

Folgende Umformungsregeln für die Erzeugung von Überwachungsbausteinen wurden identifiziert:

- Es wird ein Basisfunktionsbausteintyp erstellt. Als Typname für den Überwachungsbaustein wird der Name der Dienstsequenz gewählt.
- Für jedes Ein- oder Ausgangsereignis des zu überwachenden FB wird ein Ereigniseingang erzeugt. Auch Ereignisse, die nicht in der Sequenz modelliert wurden, unterbrechen diese und müssen demzufolge berücksichtigt werden.
- Ein Ausgangsereignis `DETECTED` zeigt an, dass das gewählte Szenario eingetreten ist.
- Für die in der Dienstsequenz angeführten Datenwerte müssen Dateneingänge vorgesehen werden.
- Im Zustandsdiagramm muss ein Zustand für jedes Telegramm erzeugt werden.
- Das im Telegramm enthaltene Ereignis wird als Bedingung einer Transition angegeben. Die Transition wird zwischen den Zuständen vor und nach dem angegebenen Ereignis erzeugt. Die Bedingung wird durch Datenwerte ergänzt, sofern diese in der Dienstsequenz modelliert wurden.
- Jeder interne Zustand muss weiters für alle anderen Ereignisse, die auftreten könnten, Transitionen zum Startzustand vorsehen. Diese unterbrechen das in der Dienstsequenz modellierte Szenario. Eine Transition mit der Bedingung 1 ist immer wahr und der Zielzustand wird daher sofort aktiv. Eine solche Bedingung wäre daher nicht geeignet, um in einer einzelnen Transition alle Ereignisse zu identifizieren, die nicht Teil der Sequenz sind und den internen Zustand daher zurücksetzen müssen.

Nach Auswahl der zu identifizierenden Sequenz erlaubt 4diac IDE die automatische Erzeugung eines Überwachungsbausteins nach diesen Umformungsregeln. Der generierte Baustein muss zusammen mit dem ursprünglichen FB verwendet werden. Dazu kann auch ein hierarchischer Baustein inklusive der notwendigen Verbindungen automatisch erzeugt werden, um den Aufwand insbesondere bei mehreren zu überwachenden Sequenzen zu verringern.

8 Evaluierung

Als Evaluierungsgrundlage dienen einerseits Experimente zur Modellierung von FB-Einschränkungen. Andererseits wird mit der automatischen Erkennung verbotener Szenarien eine Anwendungsmöglichkeit diskutiert, die Einsparungspotenziale des Entwicklungsaufwands aufzeigt.

8.1 Test des Klassifikationssystems

Wir haben unsere Realisierung der Architektur in Abbildung 2 mit denselben Bausteintypen wie [4] evaluiert. Dazu gehören die in der Norm definierten Basisfunktionsbausteintypen und ein eigener Bausteintyp für die Orchestrierung einer Station in einem Produktionssystem (`SortierSteuerung`). Die von uns implementierten Erweiterungen ermöglichen die manuelle Klassifizierung bestehender Dienstsequenzen. Im Typen-Editor haben wir Vorbedingungen und die Klassifikation für die manuell definierten Dienstsequenzen des Bausteins `SortierSteuerung` ausgewählt.

Das System ermöglicht weiters die Generierung neuer Dienstsequenzen, die direkt klassifiziert werden. Ausgewählte Dienstsequenzen, die wir automatisch für den Baustein `SortierSteuerung` generiert haben, sind in Abbildung 5 dargestellt. Im Vergleich zur manuellen Spezifikation von Dienstsequenzen erzeugt der automatisierte Prozess eine größere Vielfalt an Einschränkungen (unerwartete Varianten) und spart Zeit. Da der `SortierSteuerung`-FB sehr spezifische Datenwerte erfordert, war jedoch nur ein kleiner Teil der vollständig generierten Einschränkungen

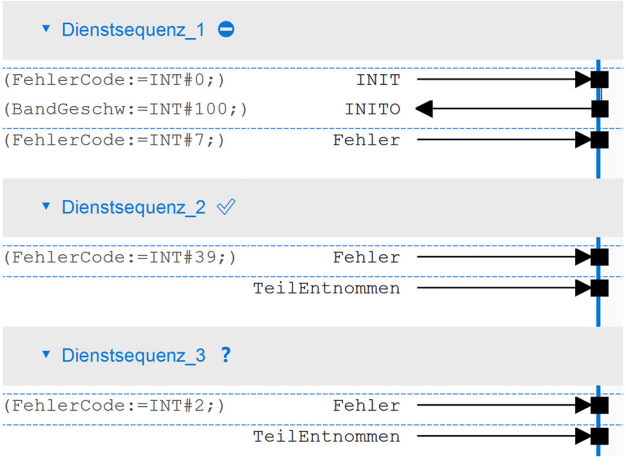

Abbildung 5: Automatisch aufgezeichnete Dienstsequenzen für einen FB. Sie sind als verboten, verpflichtend und bedingt klassifiziert.

relevant, was auf notwendige Verbesserungen der Generierung von Eingangssequenzen hinweist. Durch die Angabe des `FehlerCode` als Anfangsparameter konnte die Relevanz erhöht werden (aufgezeichnet wurden zehn Sequenzen mit bis zu drei Eingangsereignissen).

8.2 Nutzerfreundlichkeit

Bei der Entwicklung wurden Empfehlungen für die nutzerfreundliche Gestaltung von Modellierungswerkzeugen beachtet. Entsprechend des in [19] aufgezeigten Evaluierungsprozesses wurde zunächst eine Bewertung anhand definierter Kriterien durch Expert*innen vorgenommen, wovon im Folgenden einige Beispiele angeführt werden. Die Definition der Einschränkungen auf Schnittstellenebene erhöht die *Abstraktion*, da nicht auf die interne Implementierung eingegangen wird. Die in den Auswahlschaltern eingefügten Piktogramme sind geeignet, deren Funktionsweise besser auszudrücken als einfacher Text. Jedoch sind die Piktogramme für unerfahrene Nutzer*innen voraussichtlich nicht intuitiv verständlich. Durch die Implementierung als Sichten können die einzelnen Teilfenster flexibler angeordnet werden, als dies in einem Dialog der Fall wäre. Die Komplexität ist durch die vielen Einstellmöglichkeiten hoch, was jedoch teilweise durch die Aufgabe selbst bedingt wird. Eine umfassende Studie mit Anwender*innen ist Teil zukünftiger Arbeiten.

8.3 Überwachungsbausteine

Die werkzeuggestützte Erstellung eines Überwachungsbausteins ist effizienter und weniger fehleranfällig und bietet daher eine Unterstützung für Entwickler*innen. Die in Abbildung 4 gezeigte Sequenz kann in 4diac IDE manuell mit acht Interaktionen (Erzeugung der Teilelemente) oder generiert mit maximal 3 Interaktionen erzeugt werden (bei Angabe des Eingangsereignisses `TeilDetektiert` und des gewünschten Parameters). Daraus können automatisch der Überwachungsbaustein und die Komposition mit dem zu überwachenden FB erzeugt werden. Deren manuelle Erstellung erfordert mehr als 40 Interaktionen. Daraus ergibt sich eine deutliche Effizienzsteigerung.

8.4 Diskussion der Ergebnisse

Nach der Erstellung einer Dienstsequenz dokumentiert diese nicht nur implizite Annahmen über die Umgebung eines Bausteins, sondern kann auch als Testfall dienen [8]. Die Implementierung wird in der Regel später bearbeitet, z. B. um ein Refactoring durchzuführen oder einen Fehler zu beheben. Die Testfälle helfen bei der Überprüfung, dass die Änderungen nicht gegen frühere Annahmen verstoßen.

Alle Informationen zur Klassifikation werden in unserer Implementierung als Attribute gespeichert, um eine normkonforme XML-Datei zu gewährleisten. Andere Entwicklungsumgebungen sind jedoch nicht in der Lage, diese Attribute richtig zu verstehen und zeigen möglicherweise verbotene Szenarien neben gültigen an. Wir schlagen daher vor, Vorbedingungen und Klassifikationen in die nächste Ausgabe der Norm zu integrieren.

Da der Modellinterpreter ein Kernbestandteil unserer Implementierung ist, würde seine Erweiterung auch die Anwendbarkeit unseres Systems zur Definition von FB-Einschränkungen erweitern. Dann wäre beispielsweise die Analyse von zusammengesetzten FB, die ihrerseits ein Netzwerk von anderen FB enthalten, machbar. Für viele Bausteintypen ist die Ausführungssemantik in der Norm leider nicht genau definiert. Daher können die erzeugten Ausführungsprotokolle von der Ausführungsumgebung abhängen, was die Plattformunabhängigkeit der erzeugten Schnittstellenmodelle verringert.

Hinsichtlich der Erzeugung von Überwachungsbausteinen verringert die Implementierung als separater FB die Komplexität der einzelnen FB. Die beiden FB müssen jedoch über ein Netzwerk von FB verknüpft werden. Die Ausführungssemantik von FB Netzwerken ist in der Norm nicht genau bestimmt. Daher kann nicht allgemein garantiert werden, in welcher Reihenfolge mehrere gleichzeitig gesendete Ereignisse dem Überwachungs-FB zugestellt werden. Unsere Implementierung erkennt eine Dienstsequenz daher schon dann, wenn gleichzeitig gesendete Ausgangsereignisse in beliebiger Reihenfolge gesendet werden. Bezogen auf eine bestimmte Ausführungsplattform, deren Semantik bekannt ist, könnten hier Anpassungen vorgenommen werden, um die Genauigkeit des Überwachungsbausteins zu verbessern. Außerdem begrenzt diese Implementierung die Anzahl der Bestätigungstelegramme, die durch eine einzelne Anforderung hervorgerufen werden können. Derzeit können maximal sechs parallele Ereignisse detektiert werden, da die Anzahl der zu erzeugenden Zustände exponentiell wächst.

Die vorgestellten Ansätze sind insbesondere für die Definition von Einschränkungen von FB geeignet, die Abläufe in einem Steuerungssystem vorgeben. Wenn Anpassungen im System notwendig sind, sind oft auch Änderungen im Zustandsdiagramm nötig [18]. Dienstsequenzen können bereits während der initialen Entwicklung oder im Zuge der notwendigen Änderungen aufgezeichnet werden. In ihrer

Funktion als Testfälle stellen sie sicher, dass Funktionalität erhalten bleibt. Durch die automatische Generierung von Programmteilen zur Fehlerbehandlung werden einerseits Anpassungen der Funktionalität erleichtert, da die Komplexität des Steuerungsbausteines verringert wird. Andererseits sind durch die automatische Generierung keine manuellen Anpassungen notwendig, beispielsweise werden die Verbindungen bei einer notwendigen Anpassung der Schnittstelle automatisch erneuert.

9 Zusammenfassung

Dienstsequenzen können als Modell des beobachtbaren Verhaltens an der Schnittstelle eines FB dienen, was die Automatisierung von Testfällen oder die Überwachung ermöglichen kann. Für die Spezifikation von FB-Einschränkungen als Dienstsequenzen werden Erweiterungen vorgeschlagen. Diese erlauben die Modellierung von Vorbedingungen, einer Klassifikation und spezifischer Datenwerte.

Das vorgeschlagene Werkzeug ermöglicht die Generierung von Dienstsequenzen, die von dem/der Entwickler*in als verbotenes, mögliches oder obligatorisches Verhalten einer Komponente klassifiziert werden. Die Entwickler*innen stützen sich dabei auf ihr implizites Wissen über die beabsichtigte Verwendung der Komponente und die Umgebung des Systems. So kann beispielsweise eine bestimmte Reihenfolge von Ereignissen unerwartet sein und in einem funktionierenden System niemals auftreten. Die modulare Architektur unseres Klassifikationssystems ermöglicht es, es an individuelle Bedürfnisse anzupassen. Solche Anpassungen können zusätzliche oder andere Quellen von Eingabedaten oder die direkte Ausführung in einer Laufzeit umfassen, wenn genaue Zeitinformationen erforderlich sind. Derzeit können Dienstsequenzen nur für Basisfunktionsbausteintypen generiert werden. Sobald andere Bausteintypen vom Interpreter unterstützt werden, wird unser System die Klassifizierung von Sequenzdiagrammen für jede Art von FB ermöglichen.

Künftige Arbeiten werden darauf abzielen, die Relevanz der erzeugten Dienstsequenzen zu erhöhen. Die generierten Eingaben könnten besser auf den FB zugeschnitten werden, wenn mehr Informationen über die erwarteten Daten verfügbar wären. So könnte beispielsweise die Definition von Teilbereichen oder einer Reihe von diskreten erwarteten Werten die Anzahl der zu erfassenden Eingaben begrenzen. Darüber hinaus können einige Szenarien auf der Grundlage der bereits verfügbaren Informationen klassifiziert werden. Wenn eine Dienstsequenz ein Unterszenario enthält, das als verboten markiert wurde, muss auch das längere Szenario verboten sein. Eine solche Unterstützung durch das Werkzeug kann Ingenieur*innen bei der Definition von Einschränkungen und dem erwarteten Verhalten einer Komponente unterstützen.

Danksagung: Die Autor*innen danken für die finanzielle Unterstützung durch die Europäische Kommission im Rahmen des Projekts 1-SWARM (grant agreement 871743).
Research ethics: Not applicable.
Informed consent: Not applicable.
Author contributions: The authors have accepted responsibility for the entire content of this manuscript and approved its submission.
Competing interests: The authors declare no conflicts of interest.
Research funding: Parts of this work were funded by the European Commission in the project 1-SWARM (grant agreement 871743).
Data availability: Not applicable.
Software availability: Developed tools are available open source as part of the Eclipse 4diac project.

References

[1] J. Wiklander, J. Eliasson, A. Kruglyak, P. Lindgren, and J. Nordlander, "Enabling component-based design for embedded real-time software," *J. Comput.*, vol. 4, no. 12, pp. 1309—1321, 2009.
[2] A. Zoitl and V. Vyatkin, "Different perspectives [face to face] IEC 61499 architecture for distributed automation: the "glass half full" view," *IEEE Ind. Electron. Mag.*, vol. 3, no. 4, pp. 7—23, 2009.
[3] A. Fay, B. Vogel-Heuser, T. Frank, K. Eckert, T. Hadlich , and C. Diedrich, "Enhancing a model-based engineering approach for distributed manufacturing automation systems with characteristics and design patterns," *J. Syst. Software*, vol. 101, no. 1, pp. 221—235, 2015.
[4] B. Wiesmayr, A. Zoitl, A. Garmendia, and M. Wimmer, "A model-based execution framework for interpreting control software," in *2021 26th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2021.
[5] B. Wiesmayr, F. Roithmayr, and A. Zoitl, "A tool-assisted approach for user-friendly definition of fb constraints," *IFAC World Congress*, vol. 56, no. 2, 2023. https://www.sciencedirect.com/science/article/pii/S2405896323019390.
[6] B. Vogel-Heuser, D. Schütz, T. Frank, and C. Legat, "Model-driven engineering of manufacturing automation software projects — a SysML-based approach," *Mechatronics*, vol. 24, no. 7, pp. 883—897, 2014.
[7] K. Dorofeev, S. Bergemann, T. Terzimehić, J. Grothoff, M. Thies, and A. Zoitl, "Generation of the orchestrator code for skill-based automation systems," in *2021 26th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2021.
[8] R. Hametner, I. Hegny, and A. Zoitl, "A unit-test framework for event-driven control components modeled in IEC 61499," in *2014 IEEE Int. Conf. on Emerging Technology and Factory Automation (ETFA)*, 2014.

[9] M. Wenger, A. Zoitl, and J. O. Blech, “Behavioral type-based monitoring for IEC 61499,” in *2015 IEEE 20th Int. Conf. on Emerging Technologies & Factory Automation (ETFA)*, 2015.
[10] L. Prenzel and S. Steinhorst, “Automated dependency resolution for dynamic reconfiguration of IEC 61499,” in *2021 26th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2021.
[11] L. Lednicki, J. Carlson, and K. Sandstrom, “Device utilization analysis for IEC 61499 systems in early stages of development,” in *2013 IEEE 18th Int. Conf. on Emerging Technologies & Factory Automation (ETFA)*, 2013.
[12] S. D. Panjaitan and G. Frey, “Designing generic/reusable functionality based controllers for distributed control using UML,” in *2007 IEEE Int. Conf. on Robotics and Automation (ICRA)*, IEEE, 2007.
[13] B. Wiesmayr and A. Zoitl, “Requirements for a dynamic interface model of IEC 61499 Function Blocks,” in *2020 25th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2020, pp. 1069—1072.
[14] H. Prähofer and A. Zoitl, “Verification of hierarchical IEC 61499 component systems with behavioral event contracts,” in *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, 2013, pp. 578—585.
[15] H. Kugler, M. J. Stern, and E. J. A. Hubbard, “Testing scenario-based models,” in *Fundamental Approaches to Software Engineering, Lecture Notes in Computer Science*, vol. 4422, M. B. Dwyer and A. Lopes, Eds., Berlin, Heidelberg, Springer, 2007, pp. 306—320.
[16] M. Brill, W. Damm, J. Klose, B. Westphal, and H. Wittke, “Live sequence charts,” in *Integration of Software Specification Techniques for Applications in Engineering, Lecture Notes in Computer Science*, vol. 3147, H. Ehrig, Ed., Berlin, Heidelberg, Springer, 2004, pp. 374—399.
[17] V. N. Dubinin and V. Vyatkin, “On definition of a formal model for IEC 61499 function blocks,” *EURASIP J. Embed. Syst.*, vol. 2008, no. 1, pp. 1—10, 2008.
[18] L. Sonnleithner, B. Wiesmayr, V. Ashiwal, S. Sharma, A. Zoitl, and J. Walter, “Architectural concepts for IEC 61499-based machine controls: beyond normal operation handling,” in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2022, pp. 1—8.
[19] B. Wiesmayr, A. Zoitl, and R. Rabiser, “Assessing the usefulness of a visual programming ide for large-scale automation software,” *Software Syst. Model*, vol. 22, no. 5, pp. 1619—1643, 2023.

Bionotes

Bianca Wiesmayr
LIT CPS Lab, Johannes Kepler Universität Linz, Linz, Österreich, Austria
bianca.wiesmayr@jku.at
https://orcid.org/0000-0002-9512-5249

Bianca Wiesmayr ist wissenschaftliche Mitarbeiterin des LIT Cyber-Physical Systems Lab an der Johannes Kepler Universität Linz, Österreich. Sie hat ihr Doktoratsstudium an dieser Universität abgeschlossen und forscht an Softwareentwicklungsmethoden und -werkzeugen für Steuerungsprogramme sowie der Nutzerfreundlichkeit von Modellierungswerkzeugen.

Felix Roithmayr
Johannes Kepler Universität Linz, Linz, Österreich, Austria
felix.roithmayr@jku.at

Felix Roithmayr studiert an der Johannes Kepler Universität Linz Computer Science mit einem Fokus auf Computational Engineering und IT Security. Er arbeitet derzeit am Institute for Complex Systems an verschiedenen Projekten mit Bezug auf Open Hardware und RISC-V.

Alois Zoitl
LIT CPS Lab, CDL VaSiCS, Johannes Kepler Universität Linz, Linz, Austria
alois.zoitl@jku.at

Alois Zoitl ist Universitätsprofessor und stv. Leiter des LIT Cyber-Physical Systems Lab und auch Leiter des Christian Doppler Labors VaSiCS an der Johannes Kepler Universität Linz, Österreich. Seine Forschungsthemen beinhalten adaptive Produktionssysteme, verteilte Steuerungsarchitekturen, dynamische Rekonfiguration von Steuerungsprogrammen sowie Softwareentwicklungs- und Software-Qualitätssicherungsmethoden für die industrielle Automatisierungstechnik.